

10.0

IBM MQ 용 애플리케이션 개발



참고

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, [1275 페이지의 『주의사항』](#)에 있는 정보를 확인하십시오.

본 판은 버전 9 릴리스 4에 적용되며, 새 판에서 달리 명시될 IBM® MQ 때까지 이후의 모든 릴리스 및 수정 사항에도 적용됩니다. IBM은 귀하가 IBM으로 보낸 정보를 귀하의 권리를 침해하지 않는 범위 내에서 적절하다고 생각하는 방식으로 사용하거나 배포할 수 있습니다.

© Copyright International Business Machines Corporation 2007년, 2026.

목차

애플리케이션 개발	5
애플리케이션 개발 개념	6
애플리케이션에서 수행할 수 있는 조치	7
애플리케이션, 애플리케이션 이름 및 애플리케이션 인스턴스	9
MQI를 사용하는 애플리케이션 프로그램	11
다중 IBM MQ 큐 관리자에 연결하도록 클라이언트 연결 사용	12
유연하고 확장 가능한 클라이언트 애플리케이션 개발	14
객체 지향 애플리케이션	15
IBM MQ 메시지	18
Microsoft Transaction Server 애플리케이션 준비 및 실행	52
IBM MQ 애플리케이션에 대한 설계 고려사항	53
지원되는 프로그래밍 언어로 애플리케이션 이름 지정	55
메시지를 위한 설계 기술	61
애플리케이션 설계 및 성능 고려사항	62
고급 애플리케이션을 위한 기술 설계	64
IBM i 애플리케이션에 대한 설계 및 성능 고려사항	65
Linux on Power Systems - Little Endian 애플리케이션에 대한 설계 고려사항	67
z/OS 애플리케이션에 대한 설계 및 성능 고려사항	67
IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션	71
개발 JMS/Jakarta Messaging 및 Java 애플리케이션	81
사용 IBM MQ classes for JMS/Jakarta Messaging	83
IBM MQ classes for Java 사용	320
리소스 IBM MQ 어댑터 사용	386
IBM MQ 및 WebSphere Application Server 함께 사용	470
IBM MQ 헤더 패키지 사용	484
설정하기 IBM MQ에 IBM i와 Java 함께 JMS	486
Maven 저장소를 사용한 Java 애플리케이션 개발	493
C++ 애플리케이션 개발	494
C++ 샘플 프로그램	497
C++ 언어 고려사항	501
C++의 메시징	505
IBM MQ C++ 프로그램 빌드	511
.NET 애플리케이션 개발	520
설치 IBM MQ classes for .NET	521
설치 IBM MQ classes for .NET Framework	527
IBM MQ classes for .NET를 큐 관리자에 연결하는 옵션	527
.NET용 샘플 애플리케이션	528
TCP/IP 클라이언트 연결을 승인하도록 큐 관리자 구성	530
.NET의 분산 트랜잭션	531
IBM MQ .NET 프로그램 작성 및 배치	542
XMS .NET 애플리케이션 개발	581
XMS에서 지원되는 메시징 스타일	582
XMS 오브젝트 모델	583
XMS 메시지 모델	585
설치 IBM MQ classes for XMS .NET	585
메시징 서버 환경 설정	590
XMS 샘플 애플리케이션 사용	595
XMS .NET 애플리케이션 작성	598
XMS .NET 관례 대상 오브젝트에 대한 작업	620
애플리케이션에서 신규 XMS 버전 사용 금지	627
XMS 애플리케이션의 통신 보안 설정	627
XMS 메시지	632

AMQP 클라이언트 애플리케이션 개발.....	641
MQ LightApache, Qpid JMS 및 AMQP(고급 메시지 큐잉 프로토콜).....	643
AMQP 1.0 지원.....	643
AMQP 채널에서의 포인트-투-포인트 지원.....	645
AMQP 및 IBM MQ 메시지 필드 매핑.....	646
메시지 전달 안정성.....	653
IBM MQ가 있는 AMQP 클라이언트의 토폴로지.....	657
IBM MQ AMQP 리스너 제어 속성.....	663
IBM MQ를 사용하여 REST 애플리케이션 개발.....	664
REST API를 사용한 메시징.....	665
IBM MQ를 사용하여 MQI 애플리케이션 개발.....	676
IBM MQ 데이터 정의 파일.....	677
사용자 엑시트, API 엑시트 및 IBM MQ 설치 가능 서비스.....	687
큐잉을 위한 프로시저 애플리케이션 작성.....	743
클라이언트 프로시저 애플리케이션 작성.....	917
프로시저 애플리케이션 빌드.....	937
멀티플랫폼용 샘플 절차적 프로그램 사용.....	971
RPG 샘플 프로그램을 사용하여 IBM i.....	1067
다음의 샘플 절차적 프로그램을 사용하여 z/OS.....	1079
절차에 따른 프로그램 오류 핸들링.....	1133
멀티캐스트 프로그래밍.....	1138
C 언어로 ‘MQI’ 앱 개발하기.....	1143
COBOL 에서 MQI 앱 개발하기.....	1162
어셈블리어로 ‘MQI’ 앱 개발하기.....	1177
PL/I 에서 MQI 앱 개발하기.....	1192
RPG로 ‘MQI’ 앱 개발하기.....	1205
Managed File Transfer용 애플리케이션 개발.....	1205
MFT와 함께 실행할 프로그램 지정.....	1205
MFT과(와) 함께 Apache Ant 사용.....	1207
사용자 엑시트를 사용하여 MFT 사용자 정의.....	1211
에이전트 명령 큐에 메시지를 추가하여 MFT 제어.....	1224
IBM MQ Telemetry용 애플리케이션 개발.....	1225
MQ Telemetry Transport 샘플 프로그램.....	1225
MQTT 클라이언트 프로그래밍 개념.....	1227
MQTT 서비스 제어 속성.....	1246
개발하는 MicrosoftWindows Communication Foundation 애플리케이션 IBM MQ.....	1247
.NET 를 사용하는 WCF용 IBM MQ 사용자 정의 채널 소개.....	1247
WCF용 IBM MQ 사용자 정의 채널 사용.....	1252
WCF 샘플 사용.....	1269
주의사항.....	1275
프로그래밍 인터페이스 정보.....	1276
상표.....	1276

IBM MQ용 애플리케이션 개발

메시지를 송신하고 수신하며, 큐 관리자와 관련 자원을 관리하기 위한 애플리케이션을 개발할 수 있습니다. IBM MQ는 많은 다양한 언어와 프레임워크로 작성된 애플리케이션을 지원합니다.

IBM MQ용 애플리케이션 개발의 새로운 기능

애플리케이션 개발에 대해 알아보려면 다음을 방문하세요 IBM Developer IBM MQ:

- [IBM MQ 개발자 필수 과정](#) (기본 개념 학습, 데모 실행, 앱 코딩, 고급 튜토리얼 수강)
- [IBM MQ 개발자용 다운로드](#) (무료 개발자용 에디션 및 체험판 포함)

다음 절에 설명된 개념에 익숙해지면 사용자의 애플리케이션을 쉽게 개발할 수도 있습니다.

- [6 페이지의 『애플리케이션 개발 개념』](#)
- [53 페이지의 『IBM MQ 애플리케이션에 대한 설계 고려사항』](#)

객체 지향 언어 및 프레임워크에 대한 지원

IBM MQ는 다음 언어 및 프레임워크로 개발된 애플리케이션에 대한 핵심 지원을 제공합니다.

- [JMS](#)
- [Java](#)
- [C++](#)
- [.NET](#)

[15 페이지의 『객체 지향 애플리케이션』](#)도 참조하십시오.

.NET은 다양한 언어로 개발된 애플리케이션을 지원합니다. 큐에 IBM MQ 접근하기 .NET 위한 클래스 IBM MQ 사용을 설명하기 위해, 제품 IBM MQ 문서에는 다음 언어에 대한 정보가 포함되어 있습니다:

- [C# 예제 코드 및 샘플 애플리케이션](#)
- [C++ 샘플 애플리케이션](#)
- [Visual Basic 샘플 애플리케이션](#)

[542 페이지의 『IBM MQ .NET 프로그램 작성 및 배치』](#)의 내용을 참조하십시오.

IBM MQ 지원합니다. Windows 환경에서의 애플리케이션을 IBM MQ 9.1.1 위한 코어 .NET 및 환경에서의 Linux® 애플리케이션을 IBM MQ 9.1.2 위한 코어. 자세한 정보는 [521 페이지의 『설치 IBM MQ classes for .NET』](#)의 내용을 참조하십시오.

 IBM MQ에서는 OASIS AMQP 1.0 프로토콜을 구현하는 AMQP 클라이언트도 지원합니다.

MQ Light, Apache Qpid 클라이언트(예: Apache Qpid Proton 및 Apache Qpid JMS API)는 이 프로토콜을 기반으로 합니다.

MQ Light API는 [IBM MQ Light](#)에서 사용할 수 있습니다.

Apache Qpid 클라이언트는 [QPId Proton](#)에서 사용할 수 있습니다.

다음 언어 바인딩이 그대로 제공됩니다.

- [Go 바인딩](#)
- [Node.js 애플리케이션과 함께 작동하는 JavaScript API 구현](#)

프로그래밍 방식 REST API에 대한 지원

IBM MQ에서 메시지를 보내고 받기 위한 다음 프로그래밍 방식 REST API에 대한 지원을 제공합니다.

- [IBM MQ messaging REST API](#)

-  [IBM z/OS Connect EE](#)
- [IBM Integration Bus](#)
- [DataPower®IBM 게이트웨이](#)

또한 IBM MQ 섹션의 튜토리얼 '[IBM DeveloperIBM MQ 메시징 REST API 시작하기](#)'를 참조하십시오 [664 페이지](#)의 『[IBM MQ를 사용하여 REST 애플리케이션 개발](#)』. 이 학습서에는 IBM MQ messaging REST API에서 사용할 수 있는 다음 언어로 된 예제(있는 그대로 제공됨)가 포함되어 있습니다.

- 메시징 REST IBM MQ API를 사용하는 예제 실행
- HTTPS 모듈을 사용하는 Node.js 예제
- Promise 모듈을 포함하는 Node.js 예제

프로시저 프로그래밍 언어 지원

IBM MQ는 다음 프로시저 프로그래밍 언어로 개발된 애플리케이션에 대한 지원을 제공합니다.

- [C](#)
- [COBOL](#)
-  [Visual Basic\(Windows 시스템 전용\)](#)
-  [어셈블러\(IBM MQ for z/OS 전용\)](#)
-  [PL/I\(IBM MQ for z/OS 전용\)](#)
-  [RPG\(IBM MQ for IBM i 전용\)](#)

이러한 언어는 메시지 큐잉 시스템에 액세스하기 위해 메시지 큐 인터페이스(MQI)를 사용합니다. [676 페이지](#)의 『[IBM MQ를 사용하여 MQI 애플리케이션 개발](#)』의 내용을 참조하십시오. 오브젝트 지향 언어와 프레임워크에서 사용된 IBM MQ 오브젝트 모델이 MQI를 사용하는 프로시저 언어에 대해 사용할 수 없는 추가 기능을 제공하는 것을 주의하십시오.

애플리케이션 이름 지정



IBM MQ 9.1.2 이전에는 Java 또는 JMS 클라이언트 애플리케이션에 애플리케이션 이름을 지정할 수 있었습니다. IBM MQ 9.1.2부터는 더 많은 프로그래밍 언어로 애플리케이션 이름을 지정할 수 있습니다. 자세한 정보는 [55 페이지](#)의 『[지원되는 프로그래밍 언어로 애플리케이션 이름 지정](#)』의 내용을 참조하십시오.

관련 태스크

[1225 페이지](#)의 『[IBM MQ Telemetry용 애플리케이션 개발](#)』

[1247 페이지](#)의 『[개발하는 MicrosoftWindows Communication Foundation 애플리케이션 IBM MQ](#)』

WCF Windows Communication FoundationMicrosoft (Windows Communication Foundation) 사용자 지정 채널은 WCF 클라이언트와 서비스 IBM MQ 간에 메시지를 송수신합니다.

관련 참조

[1205 페이지](#)의 『[Managed File Transfer용 애플리케이션 개발](#)』

Managed File Transfer와 함께 실행할 프로그램을 지정하고, Managed File Transfer와 함께 Apache Ant 를 사용하고, 사용자 엑시트와 함께 Managed File Transfer 를 사용자 정의하고, 에이전트 명령 큐에 메시지를 넣어 Managed File Transfer 를 제어하십시오.

애플리케이션 개발 개념

사용자는 원하는 절차적 또는 객체 지향 언어를 사용하여 IBM MQ 애플리케이션을 작성할 수 있습니다. IBM MQ 애플리케이션을 디자인하고 작성하기 전에 기본 IBM MQ 개념을 숙지하십시오.

IBM MQ에 대해 작성할 수 있는 애플리케이션의 유형에 대한 정보는 [5 페이지](#)의 『[IBM MQ용 애플리케이션 개발](#)』 및 [7 페이지](#)의 『[애플리케이션에서 수행할 수 있는 조치](#)』의 내용을 참조하십시오.

관련 개념

53 페이지의 『IBM MQ 애플리케이션에 대한 설계 고려사항』

애플리케이션에서 사용 가능한 플랫폼과 환경을 이용할 수 있는 방법을 결정한 경우 IBM MQ에서 제공한 기능의 사용 방법을 결정해야 합니다.

애플리케이션에서 수행할 수 있는 조치

비즈니스 프로세스를 지원하는 데 필요한 메시지를 송신 및 수신하기 위한 애플리케이션을 개발할 수 있습니다. 또한 큐 관리자 및 관련 자원을 관리하도록 애플리케이션을 개발할 수도 있습니다.

애플리케이션이 IBM MQ for Multiplatforms에서 수행할 수 있는 조치

Multi

다중 플랫폼에서는 다음 조치를 수행하는 애플리케이션을 작성할 수 있습니다.

- 동일한 운영 체제 아래에서 실행 중인 다른 애플리케이션에 메시지를 송신합니다. 애플리케이션은 동일한 시스템 또는 다른 시스템에 있을 수 있습니다.
- 다른 IBM MQ 플랫폼에서 실행하는 애플리케이션에 메시지를 송신합니다.
- 다음 시스템에 대해 CICS®에서 메시지 큐잉을 사용하십시오.
 -  TXSeries® for AIX®
 -  IBM i
 -  Windows
- 다음 시스템에 대해 Encina에서 메시지 큐잉을 사용하십시오.
 -  AIX
 -  Windows
- 다음 시스템에 대해 Tuxedo에서 메시지 큐잉을 사용하십시오.
 -  AIX
 - AT&T
 -  Windows
- IBM MQ 작업 단위 내에서 외부 자원 관리자가 작성한 업데이트를 조정하는 트랜잭션 관리자로 IBM MQ를 사용합니다. 다음 외부 자원 관리자가 지원되며 X/OPEN XA 인터페이스를 준수합니다.
 - Db2®
 - Informix®
 - Oracle
 - Sybase
- 커밋되거나 백아웃될 수 있는 단일 작업 단위로 여러 메시지를 함께 처리합니다.
- 전체 IBM MQ 환경에서 실행하거나, IBM MQ 클라이언트 환경에서 실행합니다.

애플리케이션이 IBM MQ for z/OS에서 수행할 수 있는 조치

z/OS

z/OS에서는 다음 조치를 수행하는 애플리케이션을 작성할 수 있습니다.

- CICS 또는 IMS 내에서 메시지 큐잉을 사용합니다.
- 각 기능에 가장 적합한 환경을 선택하여 배치, CICS 및 IMS 애플리케이션 간에 메시지를 송신합니다.
- 다른 IBM MQ 플랫폼에서 실행하는 애플리케이션에 메시지를 송신합니다.
- 커밋되거나 백아웃될 수 있는 단일 작업 단위로 여러 메시지를 함께 처리합니다.

- IMS 브릿지의 도움으로 IMS 애플리케이션에 메시지를 송신하고 애플리케이션과 상호 작용합니다.
- RRS에서 조정하는 작업 단위에 참여합니다.

z/OS 내의 각 환경에는 고유 특성, 장점 및 단점이 있습니다. IBM MQ for z/OS의 장점은 애플리케이션이 하나의 환경과 관련되지 않았지만 각 환경의 이점을 이용하기 위해 분배될 수 있다는 점입니다. 예를 들어, TSO 또는 CICS를 사용하여 일반 사용자 인터페이스를 개발할 수 있으며, z/OS 배치에서 프로세스 집약적인 모듈을 실행할 수 있으며, IMS 또는 CICS에서 데이터베이스 애플리케이션을 실행할 수 있습니다. 모든 경우에서 애플리케이션의 다양한 부분은 메시지와 큐를 사용하여 통신할 수 있습니다.

IBM MQ 애플리케이션의 설계자는 이러한 환경에 내재되어 있는 차이점과 제한사항을 알고 있어야 합니다. 예를 들면, 다음과 같습니다.

- IBM MQ는 큐 관리자 간에 상호통신을 허용하는 기능을 제공합니다(분산 큐잉이라고 알려짐).
- 커밋 및 백아웃 메소드 변경사항은 배치와 CICS 환경 간에 다릅니다.
- IBM MQ for z/OS는 IMS 환경에서 온라인 메시지 처리 프로그램(MPP), 대화식 빠른 경로 프로그램(IFP) 및 배치 메시지 처리 프로그램(BMP)에 대한 지원을 제공합니다. 배치 DL/I 프로그램을 작성하는 경우, 배치 프로그램에 z/OS 대한 753 페이지의 『z/OS 배치 고려사항』 및 958 페이지의 『배치 애플리케이션 z/OS 구축 z/OS』 과 같은 항목에서 제공되는 지침을 따르십시오.
- 단일 z/OS 시스템에 여러 인스턴스가 IBM MQ for z/OS 존재할 수 있지만, 리전은 CICS 한 번에 하나의 큐 관리자만 연결할 수 있습니다. 그러나 하나 이상의 CICS 영역을 동일한 큐 관리자에 연결할 수 있습니다. IMS 및 z/OS 배치 환경에서 프로그램은 하나 이상의 큐 관리자에 연결할 수 있습니다.
- IBM MQ for z/OS를 통해 로컬 큐를 큐 관리자 그룹에서 공유할 수 있으며 처리량 및 가용성이 개선됩니다. 이러한 큐를 공유 큐라고 부르며 큐 관리자는 동일한 공유 큐에서 메시지를 처리할 수 있는 큐 공유 그룹을 작성합니다. 배치 애플리케이션은 특정 큐 관리자 이름 대신에 큐 공유 그룹 이름을 지정하여 큐 공유 그룹 내의 여러 큐 관리자 중 하나에 연결할 수 있습니다. 이는 그룹 배치 연결 또는 더 간단히 그룹 연결이라고 합니다. 공유 큐 및 큐 공유 그룹을 참조하십시오.

z/OS 지원되는 환경 간의 차이점과 해당 제한사항은 895 페이지의 『IBM MQ for z/OS의 애플리케이션 사용 및 작성』에서 자세히 설명됩니다.

관련 개념

6 페이지의 『애플리케이션 개발 개념』

사용자는 원하는 절차적 또는 객체 지향 언어를 사용하여 IBM MQ 애플리케이션을 작성할 수 있습니다. IBM MQ 애플리케이션을 디자인하고 작성하기 전에 기본 IBM MQ 개념을 숙지하십시오.

53 페이지의 『IBM MQ 애플리케이션에 대한 설계 고려사항』

애플리케이션에서 사용 가능한 플랫폼과 환경을 이용할 수 있는 방법을 결정한 경우 IBM MQ에서 제공한 기능의 사용 방법을 결정해야 합니다.

743 페이지의 『큐잉을 위한 프로시저 애플리케이션 작성』

이 정보를 사용하여 큐잉 애플리케이션 작성, 큐 관리자에 연결 및 연결 끊기, 발행/구독 및 오브젝트 열기 및 닫기에 대해 알아보십시오.

917 페이지의 『클라이언트 프로시저 애플리케이션 작성』

프로시저 언어를 사용하여 IBM MQ에서 클라이언트 애플리케이션을 작성할 때 알아야 할 사항입니다.

83 페이지의 『사용 IBM MQ classes for JMS/Jakarta Messaging』

IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 는 와 함께 제공되는 IBM MQ메시징 Java 공급자입니다. 이 메시징 공급자들은 및 Jakarta Messaging JMS 사양에서 정의된 인터페이스를 구현하는 것 외에도, 메시징 Java API에 두 가지 확장 기능을 추가합니다.

320 페이지의 『IBM MQ classes for Java 사용』

사용 IBM MQJava 환경에서 사용하십시오. IBM MQ classes for Java를 사용하면 Java 애플리케이션을 IBM MQ 클라이언트로 IBM MQ에 연결하거나 IBM MQ 큐 관리자에 직접 연결할 수 있습니다.

494 페이지의 『C++ 애플리케이션 개발』

IBM MQ에서는 IBM MQ 오브젝트에 해당하는 C++ 클래스와 배열 데이터 유형에 해당하는 추가 클래스를 제공합니다. MQI를 통해 사용할 수 없는 여러 기능을 제공합니다.

937 페이지의 『프로시저 애플리케이션 빌드』

여러 프로시저 언어 중 하나로 IBM MQ 애플리케이션을 작성하고 여러 다른 플랫폼에서 애플리케이션을 실행할 수 있습니다.

관련 태스크

971 페이지의 『멀티플랫폼용 샘플 절차적 프로그램 사용』

이 샘플 프로시저 프로그램은 제품과 함께 전달됩니다. 샘플은 C 및 COBOL로 작성되며 MQI(Message Queue Interface)의 일반 사용을 보여줍니다.

520 페이지의 『.NET 애플리케이션 개발』

IBM MQ classes for .NET 애플리케이션이 로 IBM MQ MQI client IBM MQ 연결하거나 .NET IBM MQ 서버에 직접 연결할 수 있도록 허용합니다.

1247 페이지의 『개발하는 Microsoft Windows Communication Foundation 애플리케이션 IBM MQ』

WCF Windows Communication Foundation Microsoft (Windows Communication Foundation) 사용자 지정 채널은 WCF 클라이언트와 서비스 IBM MQ 간에 메시지를 송수신합니다.

보안 설정

Multi

애플리케이션, 애플리케이션 이름 및 애플리케이션 인스턴스

애플리케이션을 디자인하고 작성하기 전에 애플리케이션, 애플리케이션 이름 및 애플리케이션 인스턴스에 대한 기본 개념을 숙지하십시오.

애플리케이션

Multi

큐 관리자에 대한 연결은 연결에서 동일한 애플리케이션 이름을 제공하는 경우 동일한 애플리케이션으로부터의 연결로 간주됩니다. 애플리케이션 이름은 DISPLAY CONN(*) TYPE CONN 명령의 APPLTAG 속성으로 표시됩니다.

참고:

1. 버전보다 IBM MQ client 이전 버전을 IBM MQ 9.1.2 사용하는 애플리케이션의 경우, 애플리케이션 이름은 자동으로 설정됩니다 IBM MQ client. 해당 값은 애플리케이션 프로그래밍 언어 및 애플리케이션이 실행 중인 플랫폼에 따라 다릅니다. 자세한 정보는 PutApplName의 내용을 참조하십시오.
2. 버전 IBM MQ client 10 IBM MQ 9.1.2 이상을 사용하는 애플리케이션의 IBM MQ client 경우 애플리케이션 이름을 특정 값으로 설정할 수 있습니다. 대부분의 경우 여기에서는 애플리케이션 코드를 변경할 필요가 없거나 애플리케이션을 다시 컴파일할 필요가 없습니다. 자세한 정보는 57 페이지의 『지원되는 프로그래밍 언어로 애플리케이션 이름 사용』의 내용을 참조하십시오.

애플리케이션 인스턴스

Multi

연결은 추가적으로 애플리케이션 인스턴스로 나뉩니다. 애플리케이션 인스턴스는 해당 애플리케이션에 대해 하나의 '실행 단위'를 제공하는 밀접하게 관련된 연결들의 집합이다. 일반적으로 이는 여러 스레드 및 연관된 IBM MQ 연결이 있을 수 있는 단일 운영 체제 프로세스입니다.

IBM MQ for Multiplatforms에서 애플리케이션 인스턴스는 특정 연결 태그와 연관됩니다. 큐 관리자는 관련되어 있음을 알 수 있는 경우 기존 애플리케이션 인스턴스와 새 연결을 자동으로 연관시킵니다.

참고:

- 클라이언트 연결을 사용하는 경우 이러한 프로세스는 하나 이상의 실행 중인 채널을 통해 큐 관리자에 연결할 수 있습니다.
- 애플리케이션에서 JMS 기본적으로 애플리케이션 인스턴스는 특정 JMS 연결 및 관련 JMS 세션 전체에 매핑됩니다. 자세한 정보는 10 페이지의 『에 IBM MQ classes for JMS 있는 애플리케이션 인스턴스 Jakarta Messaging』의 내용을 참조하십시오.

애플리케이션 인스턴스는 균일 클러스터 자동 애플리케이션 부하 분산을 사용할 때 IBM MQ for Multiplatforms 특히 중요합니다. IBM MQ for Multiplatforms 플랫폼에서는 DISPLAY APSTATUS 명령을 사용하여 현재 연결된 애플리케이션 인스턴스를 볼 수 있습니다.

일부의 경우 큐 관리자는 특히 다음의 경우에 애플리케이션 인스턴스 연관에 대한 연결을 올바르게 수행할 수 없습니다.

- 서로 다른 애플리케이션 이름을 사용하여 동일한 프로세스의 공유 대화에 여러 연결을 작성하는 경우.
- 이전 레벨 클라이언트 라이브러리가 사용 중인 경우. 예를 들어, 및 이전 IBM MQJMSIBM MQ 9.1.2 버전의 클라이언트 설치.

이러한 상황에서 애플리케이션이 자신을 다시 연결 가능으로 정의하지 않으면 이는 허용은 되지만 일부 애플리케이션 인스턴스 그룹화가 잘못 될 수 있습니다. 임의의 연결이 MQCNO_RECONNECT로 선언되면 이는 애플리케이션 밸런싱에 상당히 부정적인 영향을 미치므로 MQCONN 호출은 MQCNO_RECONNECT_INCOMPATIBLE로 거부됩니다.

관련 개념

55 페이지의 『지원되는 프로그래밍 언어로 애플리케이션 이름 지정』

IBM MQ 9.2.0 이전에 이미 Java 또는 JMS 클라이언트 애플리케이션에 애플리케이션 이름을 지정할 수 있었습니다. IBM MQ 9.2.0부터 이 기능은 IBM MQ for Multiplatforms의 다른 프로그래밍 언어로 확장되었습니다.

Multi /에 IBM MQ classes for JMS 있는 애플리케이션 인스턴스Jakarta Messaging

V 1000 기본적으로 및 IBM MQ classes for Jakarta Messaging 프로바이더는 IBM MQ classes for JMS 동일한 JMSConnection 또는 JMSContext 객체를 위해 생성된 모든 IBM MQ 클라이언트 연결을 단일 애플리케이션 인스턴스로 취급합니다.

개체 JMSConnection 아래에 할당된 각 JMSSESSION 개체에 대해 개별 IBM MQ 연결이 생성되지만, 이러한 연결들은 동일한 소켓을 공유하며 (SHARECNV 설정이 허용하는 경우) 동일한 CONNTAG 값을 할당받습니다. 일관된 클러스터가 사용 중인 경우, 애플리케이션 인스턴스가 새로운 큐 관리자로 재조정되면 이러한 연결들은 하나의 단위로 이동합니다.

개별 JMS 연결은 서로 관련이 없는 것으로 간주됩니다. 각 JMSConnectionJMSContext 객체는 개별 애플리케이션 인스턴스를 나타냅니다. 이러한 동작은 단일 대형 Java 가상 머신 프로세스 내에서 여러 개의 별도 애플리케이션 스레드를 실행하는 애플리케이션 서버 환경과 같은 일부 상황에서 바람직합니다.

JVM 모드

에서 IBM MQ 10.0.0, 전체 Java 프로세스(가상 Java 머신)의 모든 IBM MQ 연결을 연결 친화성과 JMS 관계없이 단일 애플리케이션 인스턴스로 처리하는 동작 모드를 지정할 수 있습니다. 이러한 동작은 상대적으로 적은 수의 연결 간에 암묵적 결합이 존재하며 항상 동일한 Java 가상 머신 내에서 실행되는 특정 스타일의 마이크로서비스 애플리케이션에 더 적합할 수 있습니다.

다음 방법 중 하나를 사용하여 이 작동 모드를 지정하십시오:

- 다음 환경 변수를 설정하십시오: AMQ_APP_INSTANCE_MODE=JVM.
- 응용 프로그램을 실행할 때 명령줄에서 다음 Java 시스템 속성을 설정하십시오: com.ibm.mq.cfg.jmqi.AppInstanceMode=JVM. 예를 들면, 다음과 같습니다.

```
java -Dcom.ibm.mq.cfg.jmqi.AppInstanceMode=JVM MyApplication
```

두 방법을 모두 사용하는 경우 시스템 속성이 우선 적용됩니다. 기본 설정으로 작동 모드를 되돌리려면(예: 기본 JVM 속성 파일이 모드를 로 JVM 설정하고 명령줄에서 이를 재정의하려는 경우), 값을 JMSConnection사용하십시오.

프로세스를 Java 시작한 후에도 애플리케이션이 시스템 Java 속성을 수정할 수 있습니다. 그러나 애플리케이션은 MQ 연결을 설정한 후에는 해당 속성을 수정해서는 안 됩니다.

이 모드를 사용할 때 큐 관리자는 추가 IBM MQ 연결 핸들을 표시합니다. 이 핸들은 애플리케이션 인스턴스의 제어 연결을 나타냅니다. 이 핸들은 및 JMSSESSION 객체를 JMSConnection 대신하여 생성된 핸들에 추가되며, 이 애플리케이션의 마지막 JMSConnection 객체가 연결을 끊을 때까지 유지됩니다.

JVM 모드 제한

이 작동 모드는 애플리케이션이 생성하는 연결에 IBM MQJMS 제한을 가합니다. 연결이 단일 애플리케이션 인스턴스로 처리되려면, 해당 연결의 모든 관련 옵션이 호환되어야 합니다:

- 모든 연결은 동일한 큐 관리자 이름(**QMANAGER**)을 지정해야 합니다. 연결이 큐 관리자 그룹 스타일 이름(***qMgrName**, 여기서 * 뒤에 큐 관리자 이름이 따름)을 사용하는 경우(일관된 클러스터에서 흔히 사용됨), 모든 연결은 동일한 그룹 이름을 지정해야 합니다.
- 모든 연결은 재연결 가능해야 하며 동일한 재연결 옵션 값(**QMR** 또는 **ANY**, 에 설명된 269 페이지의 『속성을 CLIENTRECONNECTOPTIONS 사용하여 클라이언트의 자동 재연결을 활성화합니다. IBM MQ classes for JMS』 대로)을 지정해야 합니다.
- 모든 연결은 동일한 애플리케이션 이름(**APPLICATIONNAME**, 일부 환경에서는 **APPLTAG**로도 알려짐)을 지정해야 합니다.

두 연결의 옵션이 일치하지 않을 경우, 후속 연결은 거부됩니다.

관련 정보

[IBM MQ JMS/Jakarta Messaging 클래스](#)

[환경 변수 설명](#)

MQI를 사용하는 애플리케이션 프로그램

IBM MQ 애플리케이션 프로그램을 성공적으로 실행하려면 특정 오브젝트가 필요합니다.

11 페이지의 [그림 1](#)에서는 큐에서 메시지를 제거하고, 메시지를 처리한 후 일부 결과를 동일한 큐 관리자의 다른 큐에 전송하는 애플리케이션을 보여줍니다.

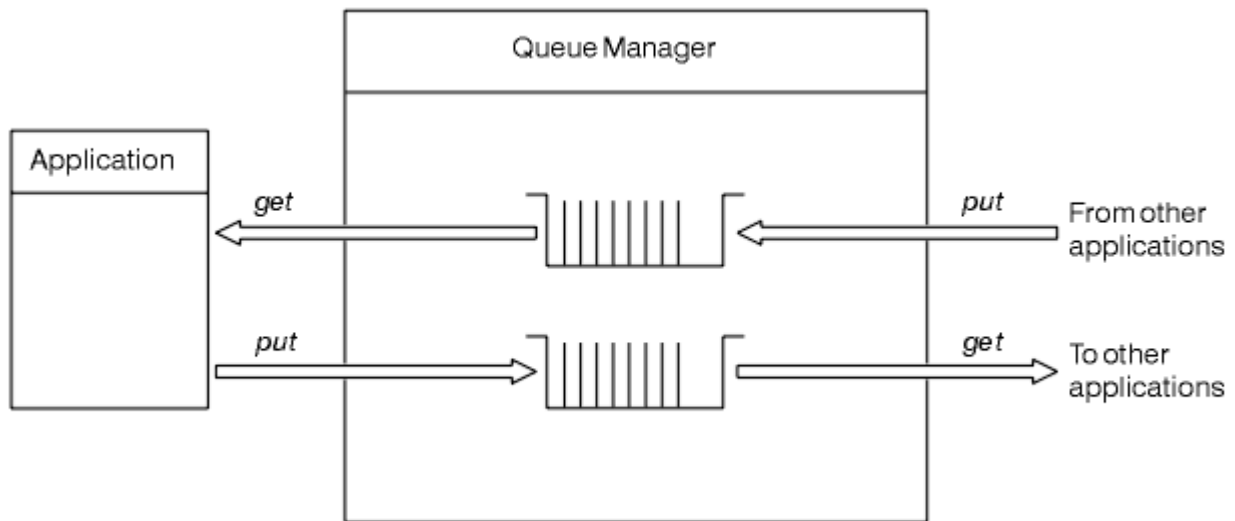


그림 1. 큐, 메시지 및 애플리케이션

애플리케이션에서 로컬 또는 리모트 큐에 메시지를 넣을 수 있는 반면(MQPUT 사용), 로컬 큐에서만 직접 메시지를 가져올 수 있습니다(MQGET 사용).

이 애플리케이션을 실행할 수 있으려면 다음 조건이 충족되어야 합니다.

- 큐 관리자가 존재하고 실행 중이어야 합니다.
- 메시지가 제거될 첫 번째 애플리케이션 큐가 정의되어 있어야 합니다.
- 애플리케이션이 메시지를 넣는 두 번째 큐도 정의되어 있어야 합니다.
- 애플리케이션이 큐 관리자에 연결할 수 있어야 합니다. 이를 수행하려면 IBM MQ에 링크되어 있어야 합니다. 937 페이지의 『프로시저 애플리케이션 빌드』의 내용을 참조하십시오.
- 첫 번째 큐에 메시지를 넣는 애플리케이션도 큐 관리자에 연결해야 합니다. 리모트인 경우 전송 큐 및 채널을 사용하여 애플리케이션을 설정해야 합니다. 시스템의 이 부분은 11 페이지의 [그림 1](#)에 표시되지 않습니다.

다중 IBM MQ 큐 관리자에 연결하도록 클라이언트 연결 사용

클라이언트 연결 애플리케이션은 부하 분산 또는 서비스 가용성 목적으로 하나 이상의 큐 매니저에 연결되도록 구성할 수 있습니다.

클라이언트에서 IBM MQ 이를 달성하기 위한 주요 메커니즘은 클라이언트 채널 정의 테이블([클라이언트 채널 정의 테이블 구성 참조](#)) 또는 연결 목록의 사용입니다.

외부 부하 분산 제품을 사용하거나 호스트 이름 또는 IP 주소를 리디렉션할 수 있는 '스텝'으로 연결 IBM MQ 코드를 감싸는 방식으로 유사한 동작을 구현할 수도 있습니다.

이러한 각 기술에는 몇 가지 제한 사항이 따르며, 특정 애플리케이션 요구 사항에 따라 적합성이 다소 다를 수 있습니다. 다음 섹션들은 비록 완전하지는 않지만, 고려해야 할 특정 측면들과 이러한 다양한 접근법들이 해당 측면들에 미치는 영향을 설명합니다.

IBM MQ 균일 클러스터([균일 클러스터에 관하여 참조](#))는 CCDT의 기본 메커니즘을 기반으로 다중 목적지를 제공함으로써, 여러 큐 관리자 간 애플리케이션의 수평적 확장성을 달성하는 강력한 메커니즘을 제공합니다. 일관된 클러스터는 기본 IBM MQ 프로토콜을 인식하지 못하는 외부 로드 밸런서를 사용할 때 가능한 것 이상의 기능을 제공할 수 있으며, 이 주제 후반부에 논의될 일부 문제를 피할 수 있으므로, 적용 가능한 경우 다른 기술보다 일관된 클러스터 사용을 우선적으로 고려하십시오.



주의: 일부 애플리케이션 환경에서는 연결이 부하 IBM MQ 분산될 때 특별한 주의가 필요합니다. 다음과 같은 경우 로드 밸런싱은 정의되지 않거나 신뢰할 수 없는 동작을 유발할 수 있습니다. 이러한 경우 중 하나에서 로드 밸런싱을 사용 중이며 IBM 지원팀에 도움을 요청할 경우, 연결 리디렉션 없이 문제를 재현해 달라는 요청을 받을 수 있습니다.

- 큐 관리자에 여러 연결을 생성하는 애플리케이션. 이러한 애플리케이션에는 일반적으로 사용 시 여러 IBM MQ 연결을 생성하는 JMS 클래스 또는 Jakarta IBM MQ IBM MQ Messaging 클래스를 사용하는 모든 애플리케이션이 포함됩니다. 외부 로드 밸런서나 사용자 정의 코드 스텝을 사용하는 경우, 동일한 애플리케이션 인스턴스에서 오는 연결은 항상 동일한 큐 매니저로 라우팅되어야 합니다.
- XA의 트랜잭션 관리 기능을 사용하거나 트랜잭션 API(JTA)를 Java 사용하는 애플리케이션. 이러한 애플리케이션은 항상 동일한 큐 관리자에 일관되게 연결하거나 재연결해야 합니다. 사용 중인 부하 분산 기술이 이러한 일관성을 보장할 수 없다면, JTA(XA)를 안정적으로 사용할 수 없습니다.
- 균일한 클러스터 애플리케이션 부하 분산. 이 기능은 클라이언트가 간섭 없이 특정 큐 관리자에 재연결하도록 지시할 수 있는 능력에 의존합니다. 따라서 외부 부하 분산을 균일 클러스터 사용과 결합하려는 시도는 권장되지 않습니다.

가능한 경우, 외부 부하 분산 기술 대신 균일 IBM MQ 클러스터를 사용하는 것을 고려하십시오.

이 정보에 사용된 용어

CCDT- 다중 QMGR

이는 동일한 그룹에 다중 클라이언트 연결(CLNTCONN) 채널을 포함하는 CCDT 파일을 의미합니다. 이는 파일 관리자 이름 클라이언트 연결(QMNAME CLNTCONN) 속성이며, 서로 다른 CLNTCONN 항목은 서로 다른 큐 관리자로 해석됩니다.

이는 동일한 다중 인스턴스 큐 관리자에 대한 서로 다른 IP 주소 또는 호스트 이름에 해당할 뿐인 다중 CLNTCONN 항목을 포함하는 CCDT 파일(코드 스텝과 결합할 수 있는 접근 방식)과는 다릅니다.

CCDT 다중 큐 관리자 접근 방식을 선택하는 경우 항목의 우선순위를 지정할 것인지 무작위 작업 로드 관리(WLM)를 사용할 것인지 선택해야 합니다.

우선순위

CLNTWGHT(1) 및 AFFINITY(PREFERRED) 속성을 통해 알파벳순으로 정렬된 다중 항목을 사용하여 마지막 정상 연결을 기억합니다.

무작위

CLNTWGHT(1) 및 AFFINITY(NONE) 속성을 사용합니다. CLNTWGHT를 조정하여 서로 다르게 확장된 IBM MQ 서버에서 WLM 가중치를 조정할 수 있습니다.

참고: 채널 간 CLNTWGHT에서 큰 차이를 방지해야 합니다.

로드 밸런서

다중 IBM MQ 큐 관리자의 TCP/IP 리스너에 대한 포트 모니터링에서 가상 IP 주소(VIP)를 구성한 네트워크 어플라이언스를 말합니다. 네트워크 어플라이언스에서 VIP를 구성하는 방법은 사용하는 네트워크 어플라이언스에 따라 달라집니다.

다음 선택사항은 메시지를 전송하는 애플리케이션 또는 동기식 메시징 요청 및 응답을 시작하는 애플리케이션에만 적용됩니다. 예를 들어, 이러한 메시지 및 요청을 지원하는 애플리케이션에 대한 고려사항의 경우 리스너는 완전히 별도의 항목이며, "큐에 메시지 리스너 연결"에서 자세히 설명합니다.

단일 큐 관리자에 연결하는 기존 애플리케이션에 필요한 코드 변경 범위

CONNNAME 목록, CCDT 다중-QMGR, 로드 밸런서

MQCONN("QMNAME")에서 MQCONN("*QMNAME")

큐 관리자 이름은 (Java EE) Java Platform, Enterprise Edition 애플리케이션의 JNDI(Naming and Java Directory Interface) 구성에 포함될 수 있습니다. 그렇지 않으면, 1자리 문자 코드 변경이 필요합니다.

코드 스텝

기존 JMS 또는 MQI 연결 논리를 코드 스텝으로 바꿉니다.

서로 다른 WLM 전략에 대한 지원

CONNNAME 목록

우선순위만 따릅니다.

코드에 부정적인 영향을 줄 수 있습니다.

CCDT 다중 QMGR

무작위 또는 우선순위에 따릅니다.

코드에 어떠한 영향도 주지 않을 수 있습니다.

로드 밸런서

임의(모든 메시지에 대한 각 연결 포함).

코드에 긍정적인 영향을 줄 수 있습니다.

코드 스텝

임의(모든 메시지에 대한 각 메시지 포함).

코드에 긍정적인 영향을 줄 수 있습니다.

1차 큐 관리자를 사용할 수 없는 동안 성능 오버헤드

CONNNAME 목록

항상 목록에서 첫 번째 항목을 시도합니다.

코드에 부정적인 영향을 줄 수 있습니다.

CCDT 다중 QMGR

마지막 정상 연결을 기억합니다.

코드에 긍정적인 영향을 줄 수 있습니다.

로드 밸런서

포트 모니터링으로 잘못된 큐 관리자를 피합니다.

코드에 긍정적인 영향을 줄 수 있습니다.

코드 스텝

마지막 정상 연결을 기억하고 지능적으로 재시도할 수 있습니다.

코드에 긍정적인 영향을 줄 수 있습니다.

XA 트랜잭션 지원

CONNNAME 목록, CCDT 다중-QMGR, 로드 밸런서

트랜잭션 관리자는 동일한 큐 관리자 자원에 다시 연결하는 복구 정보를 저장해야 합니다.

서로 다른 큐 관리자로 해석되는 MQCONN 호출은 일반적으로 이를 무효화합니다. 예를 들어, Java EE에서 단일 연결 팩토리는 XA를 사용할 때 단일 큐 관리자로 해석되어야 합니다.

코드에 부정적인 영향을 줄 수 있습니다.

코드 스텝

코드 스텝은 트랜잭션 관리자에 대한 XA 요구사항(예: 다중 연결 팩토리)을 충족할 수 있습니다.

코드에 긍정적인 영향을 줄 수 있습니다.

앱에서 인프라 변경사항을 숨기는 관리 유연성

CONNNAME 목록

DNS만.

코드에 부정적인 영향을 줄 수 있습니다.

CCDT 다중 QMGR

DNS 및 공유 파일 시스템 또는 공유 파일 시스템이나 CCDT 파일 푸시.

코드에 어떠한 영향도 주지 않을 수 있습니다.

로드 밸런서

동적 가상 IP 주소(VIP).

코드에 긍정적인 영향을 줄 수 있습니다.

코드 스텝

DNS 또는 단일 큐 관리자 CCDT 항목.

코드에 어떠한 영향도 주지 않을 수 있습니다.

계획된 유지보수에 따른 중단 방지

또 다른 상황을 고려하고 계획해야 합니다. 예를 들어, 큐 관리자의 계획된 유지보수 중 일반 사용자에게 표시되는 오류 및 제한시간 초과와 같이 애플리케이션에 대한 중단을 방지하기 위한 방법이 이에 해당합니다. 중단을 방지하는 최고의 방법은 중지하기 전에 큐 관리자에서 모든 작업을 제거하는 것입니다.

요청 및 응답 시나리오를 고려하십시오. 모든 인플라이트 요청을 완료하고 애플리케이션에서 응답을 처리하려고 하지만, 추가 작업을 시스템으로 제출하지는 않습니다. 단순히 큐 관리자를 정지해도 이 요구사항을 이행할 수 없습니다. 잘 코딩된 애플리케이션은 인플라이트 요청에 대한 응답 메시지를 수신하기 전에 리턴 코드 RC2161 MQRC_Q_MGR_QUIESCING 예외를 수신하기 때문입니다.

작업을 제출하는 데 사용된 요청 큐에서 PUT(DISABLED)을 설정하는 동시에, 응답 큐는 PUT(ENABLED) 및 GET(ENABLED) 모두로 둘 수 있습니다. 이러한 방법으로 요청 깊이, 전송, 응답 큐를 모니터링할 수 있습니다. 모두 안정화되면(즉, 인플라이트 요청이 완료되거나 제한시간이 초과된 경우) 큐 관리자를 중지할 수 있습니다.

그러나 PUT(DISABLED) 요청 큐를 처리하려면 요청 애플리케이션에서 좋은 코딩이 필요합니다. 이때 메시지를 전송하려고 할 때 리턴 코드 RC2051 MQRC_PUT_INHIBITED 오류가 발생합니다.

IBM MQ에 대한 연결을 작성하거나 요청 큐를 열 때는 예외가 발생하지 않습니다. MQPUT 호출을 사용하여 메시지를 실제로 송신하려고 시도하는 경우에만 예외가 발생합니다.

요청 및 응답 시나리오에 대한 이 오류 처리 논리를 포함하는 코드 스텝을 빌드하고 향후 이러한 코드 스텝을 사용하도록 애플리케이션 팀에 요청하면 일관된 동작으로 애플리케이션을 개발하는 데 도움이 될 수 있습니다.

유연하고 확장 가능한 클라이언트 애플리케이션 개발

내장애성 및 확장성을 위해 연결 옵션을 지원하는 클라이언트 애플리케이션을 균일 클러스터에 배치하면 애플리케이션의 인스턴스가 큐 관리자 사이에서 리밸런싱될 수 있습니다.

균일 클러스터의 개요는 [균일 클러스터](#) 정보를 참조하십시오.

이상적으로는 이러한 리밸런싱은 애플리케이션에 보이지 않지만 특정 유형의 애플리케이션만 이 유형의 배치에 적합하며 애플리케이션 설계에서 몇 가지 고려사항이 필요할 수 있습니다.

이러한 고려사항은 다음 두 가지 기본 범주로 분류됩니다.

- 재연결 가능한 애플리케이션에 대해 이미 존재할 수 있지만 균일 클러스터에 배치될 때 더 가능성이 높은 드문 오류 경로. 예를 들어, 재연결 후 인플라이트 작업 단위가 백아웃되고 찾아보기 커서가 재설정됩니다. 이는 현재 환경에서 재연결 가능한 애플리케이션에 대해 드문 경우일 수 있으므로 애플리케이션 코드를 통해 가능한 한 깔끔하게 처리되지 않습니다. 애플리케이션 로직을 검토하여 이러한 상황에 적절한 처리가 있는지 확인하면 예기치 않은 문제가 발생하지 않도록 방지하는 데 도움이 됩니다.
- 특정 큐 관리자에 대한 연관관계. 애플리케이션이 항상 동일하거나 특정한 큐 관리자에 다시 연결해야 한다는 것을 알고 있는 경우에는, 애플리케이션을 해당 큐 관리자에 다시 연결하도록 구성하거나 해당 큐 관리자에 대한 연결을 사용하지 않도록 설정해야 합니다. 하지만 이러한 연관관계는 일시적일 수 있습니다(예: 응답 메시지 대기). 애플리케이션 코드에서 이러한 연관관계를 설명하기 위해 밸런싱 알고리즘에 미치는 영향은 다음 섹션에서 논의됩니다. 이러한 옵션에 대한 자세한 내용 및 애플리케이션 코드가 아닌 구성을 통해 유사한 접근 방식을 수행하는 방법은 [균일 클러스터의 애플리케이션 리밸런싱에 미치는 영향](#)을 참조하십시오.

MQI의 재연결 옵션에 미치는 영향

MQCNO_RECONNECT에 대한 자세한 정보는 [재연결 옵션](#)을 참조하십시오.

애플리케이션이 항상 동일하거나 특정한 큐 관리자에 다시 연결해야 한다는 것을 알고 있는 경우에는, MQCNO_RECONNECT_Q_MGR 또는 MQCNO_RECONNECT_DISABLED로 애플리케이션을 구성해야 합니다.

MQI의 밸런싱 알고리즘에 미치는 영향

하지만 특정 유형의 애플리케이션 요구에 맞게 리밸런싱 작동을 제어하거나 영향을 미치기를 원할 수 있습니다. 예를 들어, 인플라이트 트랜잭션에 대한 인터럽트를 최소화하거나 요청자 애플리케이션이 이동하기 전에 응답을 수신하게 할 수 있습니다.

특정 바람직한 기본 작동은 [균일 클러스터의 애플리케이션 리밸런싱에 미치는 영향](#)에서 가정되고 논의됩니다. 이 주제에 설명된 대로 client.ini 파일을 통해 구성 또는 배치 시 특정 애플리케이션의 동작에 영향을 줄 수도 있습니다.

다른 상황에서는 밸런싱 작동 및 요구사항을 애플리케이션 로직의 일부로 만드는 것이 더 합리적일 수 있습니다. 이러한 경우에는 MQBNO(밸런싱 옵션)라는 구조에서, MQCONNX 호출로 큐 관리자에 연결할 때 애플리케이션의 동일한 관련 특성이 IBM MQ에 제공될 수 있습니다.

MQBNO 구조를 제공하면 이 구조는 애플리케이션에 다른 큐 관리자에 다시 연결하도록 요청해야 하는 방법 및 시기를 결정하기 위해 IBM MQ에 필요한 모든 정보를 제공해야 합니다.

다음은 제공해야 합니다.

- **Type** 신청의
- 인스턴스가 상태와 무관하게 재조정되는 시점 **Timeout**
- 특별한 **BalanceOptions**

이에 대한 예외는 필요한 경우 제한시간을 MQBNO_TIMEOUT_DEFAULT로 둘 수 있다는 점입니다. 이 경우 제한 시간은 client.ini 파일, 애플리케이션 또는 글로벌 스탠자의 값(제공된 경우)으로 해석되며 제공되지 않으면 기본 값인 10초로 해석됩니다.

이 구조의 형식에 대한 세부사항은 [MQBNO](#)를 참조하십시오.

.NET 애플리케이션의 경우, 자세한 내용은 [애플리케이션 재균형 조정 영향에 .NET](#) 관한 내용을 참조하십시오.

객체 지향 애플리케이션

IBM MQ은(는) JMS, Java, C++ 및 .NET에 대한 지원을 제공합니다. 이러한 언어 및 프레임워크는 IBM MQ 오브젝트 모델을 사용하며 IBM MQ 호출 및 구조와 동일한 기능을 제공하는 클래스를 제공합니다.

IBM MQ 오브젝트 모델을 사용하는 일부 언어 및 프레임워크에서는 메시지 큐 인터페이스(MQI)와 함께 프로시저 언어를 사용할 때 사용 가능하지 않은 추가 기능을 제공합니다.

이 모델에서 제공하는 클래스, 메소드 및 특성에 대한 자세한 내용은 [16 페이지의 『IBM MQ 오브젝트 모델』](#)의 내용을 참조하십시오.

JMS

IBM MQ 사양을 Jakarta Messaging 3.0 그리고 Java Message Service 2.0 구현하는 클래스를 제공합니다. 자세한 내용은 IBM MQ classes for JMS를 참조하십시오 [사용 IBM MQ classes for JMS](#). 와 IBM MQ classes for Java 의 차이점에 대한 IBM MQ classes for JMS 정보는 사용 시 선택에 도움이 되도록 에서 확인하십시오 81 페이지의 『개발 JMS/Jakarta Messaging 및 Java 애플리케이션』.

IBM MQ Message Service Client (XMS) for C/C++ 그리고 (JMS) Java Message Service API와 동일한 인터페이스 세트를 가진 라는 IBM MQ Message Service Client (XMS) for .NETXMS 애플리케이션 프로그래밍 인터페이스(API)를 제공합니다. 자세한 정보는 581 페이지의 『XMS.NET 애플리케이션 개발』 의 내용을 참조하십시오.

Java

에 대한 정보는 객체 IBM MQ 모델을 사용하여 Java코딩하는 [사용 IBM MQ classes for Java](#) 방법에 관한 내용을 참조하십시오.

Stabilized IBM은 IBM MQ classes for Java에 대해 추가적인 개선 계획이 없으며 IBM MQ 8.0에 제공된 레벨에서 기능적으로 안정되어 있습니다. IBM MQ classes for Java와 IBM MQ classes for JMS 사이의 차이점을 알면 사용할 클래스를 결정하는 데 도움이 될 것입니다. 차이점에 대한 자세한 정보는 81 페이지의 『개발 JMS/Jakarta Messaging 및 Java 애플리케이션』 의 내용을 참조하십시오.

C++

IBM MQ에서는 IBM MQ 오브젝트에 해당하는 C++ 클래스와 배열 데이터 유형에 해당하는 추가 클래스를 제공합니다. MQI를 통해 사용할 수 없는 여러 기능을 제공합니다. IBM MQ 오브젝트 모델을 사용하여 C++로 프로그램을 코딩하는 방법에 대한 정보는 C++ 사용의 내용을 참조하십시오. Message Service Clients for C/C++ 및 .NET은(는) Java Message Service(JMS) API와 동일한 인터페이스 세트가 있는 XMS라는 API(Application Programming Interface)를 제공합니다.

.NET

클래스를 IBM MQ.NET 사용한 코딩 .NET 프로그램에 대한 정보는 [를 애플리케이션 개발 .NET](#) 참조하십시오. C/C++ 및 .NET에 대한 메시지 서비스 클라이언트는 Java Message Service(JMS) API와 동일한 인터페이스 세트가 있는 XMS라는 API(Application Programming Interface)를 제공합니다.

관련 개념

676 페이지의 『IBM MQ를 사용하여 MQI 애플리케이션 개발』

IBM MQ에서는 C, Visual Basic, COBOL, 어셈블러, RPG, pTAL 및 PL/I에 대한 지원을 제공합니다. 이러한 절차적 언어는 메시지 큐 인터페이스(MQI)를 사용하여 메시지 큐잉 서비스에 액세스합니다.

기술 개요

6 페이지의 『애플리케이션 개발 개념』

사용자는 원하는 절차적 또는 객체 지향 언어를 사용하여 IBM MQ 애플리케이션을 작성할 수 있습니다. IBM MQ 애플리케이션을 디자인하고 작성하기 전에 기본 IBM MQ 개념을 숙지하십시오.

관련 참조

[애플리케이션 참조 개발](#)

IBM MQ 오브젝트 모델

IBM MQ 오브젝트 모델은 클래스, 메소드 및 특성으로 구성됩니다.

IBM MQ 오브젝트 모델은 다음으로 구성됩니다.

- 클래스 - 큐 관리자, 큐 및 메시지와 같은 익숙한 IBM MQ 개념을 나타냅니다.
- 메소드 - MQI 호출에 해당하는 각 클래스의 메소드.
- 특성 - IBM MQ 오브젝트의 속성에 해당하는 각 클래스의 특성.

IBM MQ 오브젝트 모델을 사용하여 IBM MQ 애플리케이션을 작성하는 경우 애플리케이션에서 이러한 클래스의 인스턴스를 작성합니다. 객체 지향 프로그래밍에서 클래스의 인스턴스를 오브젝트라고 합니다. 오브젝트가 작성된 경우 오브젝트 특성의 값을 조사하거나 설정하고(MQINQ 또는 MQSET 호출 발행과 동일한) 오브젝트에 대한 메소드 호출을 작성하여(다른 MQI 호출 발행과 동일한) 오브젝트와 상호작용합니다.

클래스

IBM MQ 오브젝트 모델은 다음의 클래스 기본 세트를 제공합니다.

모델의 실제 구현은 지원되는 다양한 객체 지향 환경 간에 다소 다를 수 있습니다.

MQQueueManager

MQQueueManager 클래스의 오브젝트는 큐 관리자에 대한 연결을 나타냅니다. Connect(), Disconnect(), Commit() 및 Backout()(MQCONN과 동일하거나 MQCONN, MQDISC, MQCMIT 및 MQBACK)에 대한 메소드가 있습니다. 큐 관리자의 속성에 해당하는 속성을 가지고 있습니다. 큐 관리자 속성 특성에 액세스하면 이미 연결되지 않은 경우 큐 관리자에 내재적으로 연결됩니다. MQQueueManager 오브젝트를 영구 삭제하면 큐 관리자의 연결이 내재적으로 끊깁니다.

MQQueue

MQQueue 클래스의 오브젝트는 큐를 나타냅니다. 큐로 또는 큐에서 메시지를 Put() 및 Get()하기 위한 메소드가 있습니다(MQPUT 및 MQGET과 동일함). 큐의 속성에 해당하는 특성을 가지고 있습니다. 큐 속성 특성에 액세스하거나 Put() 또는 Get() 메소드 호출을 발행하면 내재적으로 큐가 열립니다(MQOPEN과 동일함). 내재적으로 MQQueue 오브젝트를 영구 삭제하면 큐가 닫힙니다(MQCLOSE와 동일함).

MQTopic

MQTopic 클래스의 오브젝트는 주제를 나타냅니다. 주제로 또는 주제에서 메시지를 Put()(발행) 및 Get()(수신 또는 구독)하기 위한 메소드가 있습니다(MQPUT 및 MQGET과 동일함). 주제의 속성에 해당하는 특성을 가지고 있습니다. MQTopic 오브젝트는 발행 또는 구독에 대해 액세스할 수 있으며 동시에 둘 다 액세스할 수는 없습니다. 메시지를 수신하는 데 사용되는 경우 관리되지 않는 또는 관리되는 구독으로 MQTopic 오브젝트를 작성할 수 있으며, 지속 가능 또는 지속 불가능 구독자로서 이러한 다양한 시나리오를 위해 다중 오버로드된 구성자가 제공됩니다.

MQMessage

MQMessage 클래스의 오브젝트는 큐에 넣거나 큐에서 가져온 메시지를 나타냅니다. 버퍼를 포함하며 애플리케이션 데이터와 MQMD를 모두 캡슐화합니다. 다양한 유형의 사용자 데이터(예: 문자열, 긴 정수, 짧은 정수, 단일 바이트)를 버퍼에 쓰고 버퍼로부터 읽을 수 있는 메소드 및 MQMD 필드에 해당하는 특성이 있습니다.

MQPutMessageOptions

MQPutMessageOptions 클래스의 오브젝트는 MQPMO 구조를 나타냅니다. MQPMO 필드에 해당하는 특성을 가지고 있습니다.

MQGetMessageOptions

MQGetMessageOptions 클래스의 오브젝트는 MQGMO 구조를 나타냅니다. MQGMO 필드에 해당하는 특성을 가지고 있습니다.

MQProcess

MQProcess 클래스의 오브젝트는 프로세스 정의를 나타냅니다(트리거링과 함께 사용됨). 프로세스 정의의 속성을 나타내는 특성이 있습니다.

MQDistributionList

MQDistributionList 클래스의 오브젝트는 (단일 MQPUT으로 다수의 메시지를 전송하는 데 사용되는) 분배 목록을 나타냅니다. MQDistributionListItem 오브젝트의 목록을 포함합니다.

MQDistributionListItem

MQDistributionListItem 클래스의 오브젝트는 단일의 분배 목록 대상을 나타냅니다. MQOR, MQRR 및 MQPMR 구조를 캡슐화하고 이러한 구조의 필드에 해당하는 특성이 있습니다.

오브젝트 참조

MQI를 사용하는 IBM MQ 프로그램에서 IBM MQ는 연결 핸들과 오브젝트 핸들을 프로그램에 리턴합니다.

이러한 핸들은 후속 IBM MQ 호출 시 매개변수로 전달되어야 합니다. IBM MQ 오브젝트 모델을 사용하여 애플리케이션 프로그램에서 핸들을 숨깁니다. 대신 클래스에서 오브젝트를 작성하면 결과적으로 오브젝트 참조가 애플리케이션 프로그램에 리턴됩니다. 이 오브젝트는 오브젝트에 대한 특성 액세스 및 메소드 호출을 작성할 때 사용됩니다.

리턴 코드

메소드 호출 발행 또는 특성 값을 설정하면 결과적으로 리턴 코드가 설정됩니다.

이러한 리턴 코드는 완료 코드 및 이유 코드이며 오브젝트의 자체 특성입니다. 완료 코드 및 리턴 코드의 값은 MQI에 대해 정의된 코드와 동일하며 일부 추가 값은 객체 지향 환경에 특정합니다.

IBM MQ 메시지

IBM MQ 메시지는 메시지 특성과 애플리케이션 데이터로 구성됩니다. 메시지가 송신 및 수신 애플리케이션 간에 이동하는 경우 메시지 큐잉 메시지 디스크립터(MQMD)에는 애플리케이션 데이터를 수반하는 제어 정보가 포함되어 있습니다.

메시지의 부분

IBM MQ 메시지는 다음 두 파트로 구성됩니다.

- 메시지 특성
- 애플리케이션 데이터

18 페이지의 [그림 2](#)은 메시지를 나타내며 논리적으로 메시지 특성 및 애플리케이션 데이터로 나누어지는 방법을 표시합니다.

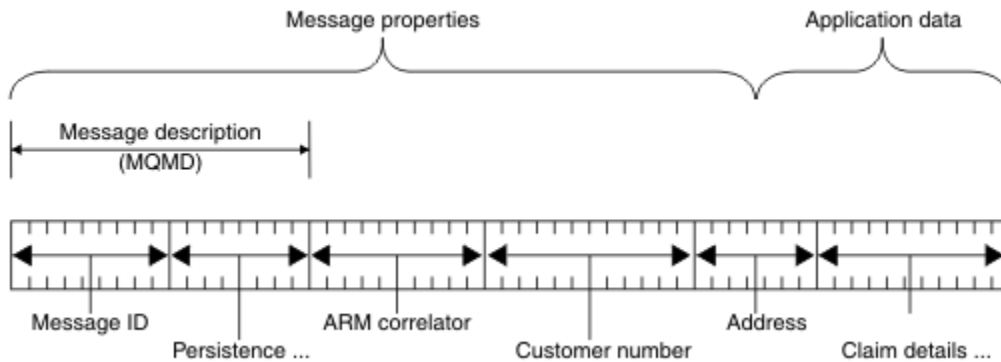


그림 2. 메시지의 표현

IBM MQ 메시지에 포함되는 애플리케이션 데이터는 데이터 변환이 수행되지 않는 한 큐 관리자에 의해 변경되지 않습니다. 또한 IBM MQ는 이 데이터의 콘텐츠에 어떠한 제한사항도 두지 않습니다. 각 메시지의 데이터 길이는 큐 및 큐 관리자 둘 다의 **MaxMsgLength** 속성 값을 초과할 수 없습니다.

➤ **ALW** AIX, Linux, and Windows에서는 큐 관리자 및 큐 **MaxMsgLength** 속성의 기본값이 4MB (4,194,304바이트)로 지정되고 필요한 경우 최대 100MB(104,857,600바이트)까지 변경할 수 있습니다.

➤ **IBM i** IBM i에서는 큐 관리자 및 큐 **MaxMsgLength** 속성의 기본값이 4MB (4,194,304바이트)로 지정되고 필요한 경우 최대 100MB(104,857,600바이트)까지 변경할 수 있습니다. IBM i에서 15MB보다 큰 IBM MQ 메시지를 사용하려는 경우 943 페이지의 [『IBM i에서 절차적 애플리케이션 빌드』](#)의 내용을 참조하십시오.

➤ **z/OS** z/OS에서는 큐 관리자의 **MaxMsgLength** 속성이 100MB로 고정되며 큐 **MaxMsgLength** 속성의 기본값은 4MB(4,194,304바이트)로 지정되고 필요한 경우 최대 100MB로 변경할 수 있습니다.

일부 환경에서는 **MaxMsgLength** 속성의 값보다 메시지를 약간 짧게 작성하십시오. 자세한 정보는 777 페이지의 [『메시지의 데이터』](#)의 내용을 참조하십시오.

MQPUT 또는 MQPUT1 MQI 호출을 사용하는 경우 메시지를 작성합니다. 이러한 호출에 대한 입력으로 메시지 우선순위 및 응답 큐의 이름과 같은 제어 정보 및 데이터를 제공하면 호출은 메시지를 큐에 넣습니다. 이러한 호출에 대한 자세한 정보는 [MQPUT](#) 및 [MQPUT1](#)을 참조하십시오.

메시지 디스크립터

메시지 디스크립터를 정의하는 MQMD 구조를 사용하여 메시지 제어 정보에 액세스할 수 있습니다.

MQMD 구조에 대한 자세한 설명은 [MQMD - 메시지 디스크립터](#)의 내용을 참조하십시오.

메시지의 원본에 대한 정보를 포함하는 MQMD 내에서 필드를 사용하는 방법에 대한 설명은 [51 페이지의 『메시지 컨텍스트』](#)의 내용을 참조하십시오.

여러 버전의 메시지 디스크립터가 있습니다. 메시지 그룹화 및 세그먼트화에 대한 추가 정보([48 페이지의 『메시지 그룹』](#) 참조)는 메시지 디스크립터(또는 MQMDE)의 버전 2에 제공됩니다. 이 내용은 버전 1 메시지 디스크립터와 동일하지만 추가 필드가 포함되어 있습니다. 이러한 필드는 [MQMDE - 메시지 설명자 확장](#)에서 설명됩니다.

메시지의 유형

IBM MQ에 의해 정의되는 메시지 유형에는 네 가지가 있습니다.

네 가지 메시지는 다음과 같습니다.

- [데이터그램](#)
- [요청 메시지](#)
- [응답 메시지](#)
- [보고 메시지](#)
 - [보고 메시지의 유형](#)
 - [보고 메시지 옵션](#)

애플리케이션은 처음 세 가지 유형의 메시지를 사용하여 애플리케이션 간에 정보를 전달할 수 있습니다. 네 번째 유형인 보고는 애플리케이션 및 큐 관리자가 오류 발생과 같은 이벤트에 관한 정보를 보고하는 데 사용합니다.

각 메시지 유형은 MQMT_* 값으로 식별됩니다. 사용자 고유 유형의 메시지를 정의할 수도 있습니다. 사용할 수 있는 값의 범위는 [MsgType](#)을 참조하십시오.

데이터그램

메시지를 수신하는(즉, 큐에서 메시지를 가져오는) 애플리케이션으로부터의 응답이 필요하지 않은 경우 데이터그램을 사용하십시오.

데이터그램을 사용할 수 있는 애플리케이션의 한 예는 공항 라운지에서 비행 정보를 표시하는 애플리케이션입니다. 메시지는 비행 정보의 전체 화면에 대한 데이터를 포함할 수 있습니다. 이러한 애플리케이션은 메시지가 전달되지 않은 경우 문제가 되지 않으므로 메시지의 수신확인을 요청할 가능성이 적습니다. 애플리케이션은 잠시 후 업데이트 메시지를 송신합니다.

요청 메시지

메시지를 수신하는 애플리케이션의 응답을 원할 경우 요청 메시지를 사용하십시오.

요청 메시지를 사용할 수 있는 애플리케이션의 한 예는 당좌 예금 계좌의 잔액을 표시하는 애플리케이션입니다. 요청 메시지는 계좌 번호를 포함할 수 있고 응답 메시지에는 계좌 잔액이 포함됩니다.

응답 메시지를 요청 메시지와 링크하려는 경우 두 가지 옵션을 사용할 수 있습니다.

- 요청 메시지를 핸들링하는 애플리케이션이 요청 메시지와 관련된 응답 메시지에 정보를 넣는지 확인하게 하십시오.
- 요청 메시지의 메시지 디스크립터에 있는 보고 필드를 사용하여 응답 메시지의 *MsgId* 및 *CorrelId* 필드의 콘텐츠를 지정하십시오.
 - 원래 메시지의 *MsgId* 또는 *CorrelId*를 응답 메시지의 *CorrelId* 필드에 복사되도록 요청할 수 있습니다(기본 조치는 *MsgId* 복사).
 - 응답 메시지에 대한 새 *MsgId*를 생성하거나 원래 메시지의 *MsgId*가 응답 메시지의 *MsgId* 필드에 복사되도록 요청할 수 있습니다(기본 조치는 새 메시지 ID 생성).

응답 메시지

다른 메시지에 응답할 경우 응답 메시지를 사용하십시오.

응답 메시지를 작성하는 경우 응답 중인 메시지의 메시지 디스크립터에 설정된 옵션을 준수하십시오. 보고서 옵션은 메시지 ID(*MsgId*) 및 상관 ID(*CorrelId*) 필드의 콘텐츠를 지정합니다. 이러한 필드를 통해 응답을 수신하는 애플리케이션에서 해당 응답을 원래 요청과 상관시킬 수 있습니다.

보고 메시지

보고 메시지는 메시지 처리 시 오류의 발생과 같은 이벤트에 대한 정보를 애플리케이션에게 알립니다.

다음에서 메시지를 생성할 수 있습니다.

- 큐 관리자
- 메시지 채널 에이전트(예를 들어, 메시지를 전달할 수 없는 경우) 또는
- 애플리케이션(예: 메시지에 있는 데이터를 사용할 수 없는 경우).

보고 메시지는 언제든지 생성될 수 있으며 애플리케이션에서 예상하지 못하는 시기에 큐에 도착할 수도 있습니다.

보고 메시지의 유형

메시지를 큐에 넣는 경우 다음을 수신하도록 선택할 수 있습니다.

- 예외 보고 메시지. 예외 플래그가 설정된 메시지에 대한 응답으로 송신됩니다. 메시지 채널 에이전트(MCA) 또는 애플리케이션에 의해 생성됩니다.
- 만기 보고 메시지. 이는 애플리케이션에서 해당 만기 임계값에 도달한 메시지를 검색하려고 시도했음을 표시합니다. 이 유형의 보고는 큐 관리자가 생성합니다.
- 도착 확인(COA) 보고 메시지. 메시지가 대상 큐에 도달했음을 표시합니다. 이 메시지는 큐 관리자가 생성합니다.
- 전달 확인(COD) 보고 메시지. 수신 애플리케이션에서 메시지를 검색했음을 표시합니다. 이 메시지는 큐 관리자가 생성합니다.
- 긍정적인 조치 알림(PAN) 보고 메시지. 요청이 성공적으로 서비스되었음을 표시합니다(즉, 메시지에서 요청된 조치가 성공적으로 수행됨). 이 유형의 보고는 애플리케이션에서 생성합니다.
- 부정적인 조치 알림(NAN) 보고 메시지. 요청이 성공적으로 서비스되지 않았음을 표시합니다(즉, 메시지에서 요청된 조치가 성공적으로 수행되지 않음). 이 유형의 보고는 애플리케이션에서 생성합니다.

참고: 각 보고 메시지 유형에는 다음 중 하나가 포함됩니다.

- 전체 원본 메시지
- 원본 메시지의 처음 100바이트 데이터
- 원본 메시지의 데이터 없음

큐에 메시지를 넣는 경우 두 가지 이상의 보고 메시지 유형을 요청할 수 있습니다. 전달 확인 보고 메시지 및 예외 보고 메시지 옵션을 선택하는 경우 메시지 전달에 실패하면 예외 보고 메시지를 수신합니다. 그러나 전달 확인 보고 메시지 옵션만 선택하고 메시지 전달에 실패하는 경우에는 예외 보고 메시지가 전달되지 않습니다.

특정 메시지 생성 기준이 충족되는 경우 요청하는 보고 메시지만 수신하게 됩니다.

보고 메시지 옵션

예외가 발생한 후 메시지를 제거할 수 있습니다. 제거 옵션을 선택하고 예외 보고 메시지를 요청한 경우 보고 메시지가 *ReplyToQ* 및 *ReplyToQMGr*로 전송되고 원래 메시지는 제거됩니다.

참고: 이러한 조치를 수행하면 데드-레터 큐로 전달되는 메시지의 수를 줄일 수 있습니다. 그러나 데이터그램 메시지만 송신하지 않는 한 애플리케이션이 리턴된 메시지를 처리해야 한다는 것을 의미합니다. 예외 보고 메시지가 생성되는 경우 원래 메시지의 지속성을 상속합니다.

요청 메시지를 전달할 수 없는 경우(예를 들어, 큐가 가득 찬 경우) 보고 메시지는 데드-레터 큐에 넣어집니다.

보고 메시지를 수신하려는 경우 *ReplyToQ* 필드에 응답 대상 큐의 이름을 지정하십시오. 그렇지 않으면 원래 메시지의 *MQPUT* 또는 *MQPUT1*이 *MQRC_MISSING_REPLY_TO_Q*로 실패합니다.

메시지의 메시지 디스크립터(MQMD)에 다른 보고서 옵션을 사용하여 메시지에 대해 작성되는 모든 보고 메시지의 *MsgId* 및 *CorrelId* 필드의 콘텐츠를 지정할 수 있습니다.

- 원래 메시지의 *MsgId* 또는 *CorrelId*가 보고 메시지의 *CorrelId* 필드에 복사되도록 요청할 수 있습니다. 기본 조치는 메시지 ID를 복사하는 것입니다. 메시지 송신자가 응답 또는 보고 메시지를 원래 메시지와 상관시킬 수 있으므로 *MQRO_COPY_MSG_ID_TO_CORRELID*를 사용하십시오. 응답 또는 보고 메시지의 상관 ID는 원래 메시지의 메시지 ID와 동일합니다.
- 보고 메시지에 대해 새 *MsgId*를 생성하거나 원래 메시지의 *MsgId*가 보고 메시지의 *MsgId* 필드에 복사되도록 요청할 수 있습니다. 기본 조치는 새 메시지 ID를 생성하는 것입니다. 시스템의 각 메시지가 서로 다른 메시지 ID를 갖고 있으며 시스템의 다른 모든 메시지와 분명하게 구분할 수 있으므로 *MQRO_NEW_MSG_ID*를 사용하십시오.
- 전문화된 애플리케이션은 *MQRO_PASS_MSG_ID* 또는 *MQRO_PASS_CORREL_ID*를 사용해야 할 수 있습니다. 단, 예를 들어, 큐에 동일한 메시지 ID를 갖는 여러 메시지가 포함되어 있는 경우 큐에서 메시지를 읽는 애플리케이션을 올바르게 작동하는지 확인할 수 있도록 설계해야 합니다.

서버 애플리케이션은 요청 메시지에서 이러한 플래그의 설정을 확인하고, 응답 또는 보고 메시지에서 *MsgId* 및 *CorrelId* 필드를 적절히 설정해야 합니다.

요청자 애플리케이션과 서버 애플리케이션 사이에서 중개자 역할을 하는 애플리케이션은 이러한 플래그의 설정을 확인할 필요가 없습니다. 이러한 애플리케이션은 일반적으로 *MsgId*, *CorrelId* 및 *Report* 필드를 변경하지 않고 서버 애플리케이션으로 메시지를 전달해야 하기 때문입니다. 이렇게 하면 서버 애플리케이션이 원래 메시지의 *MsgId*를 응답 메시지의 *CorrelId* 필드로 복사할 수 있습니다.

메시지에 대한 보고서를 생성할 때 서버 애플리케이션은 이러한 옵션이 설정되었는지 확인하기 위해 테스트를 수행해야 합니다.

보고 메시지 사용 방법에 대한 자세한 정보는 [보고서](#)의 내용을 참조하십시오.

보고서의 네이처를 표시하기 위해 큐 관리자는 다양한 피드백 코드를 사용합니다. 큐 관리자가 보고 메시지의 메시지 디스크립터 *Feedback* 필드에 이러한 코드를 넣습니다. 또한 큐 관리자는 *Feedback* 필드에 *MQI* 이유 코드를 리턴할 수도 있습니다. IBM MQ는 애플리케이션이 사용할 다양한 피드백 코드를 정의합니다.

피드백 및 이유 코드에 대한 자세한 정보는 [피드백](#)을 참조하십시오.

피드백 코드를 사용할 수 있는 프로그램의 한 예는 큐를 제공하는 다른 프로그램의 워크로드를 모니터링하는 프로그램입니다. 큐를 제공하는 프로그램의 인스턴스가 두 개 이상이고, 큐에 도착하는 메시지 수가 더 이상 이를 정당화하지 않는 경우, 이러한 프로그램은 제공 프로그램 중 하나에게 해당 프로그램이 활동을 종료해야 함을 알리는 보고 메시지(피드백 코드 *MQFB_QUIT* 포함)를 송신할 수 있습니다. (모니터링 프로그램은 *MQINQ* 호출을 사용하여 몇 개의 프로그램이 큐를 제공하고 있는지 알아낼 수 있습니다.)

Multi 보고서 및 세그먼트된 메시지

메시지가 세그먼트화되고 보고서 생성을 요청하는 경우, 메시지가 세그먼트화되지 않았던 경우보다 더 많은 보고서를 수신할 수 있습니다. 분할된 메시지에 대한 보고서는 멀티플랫폼에서만 이용 가능합니다.

세그먼트화된 메시지에 대한 설명은 [807 페이지의 『메시지 세그먼트화』](#)의 내용을 참조하십시오.

IBM MQ에서 생성하는 보고서의 경우

메시지를 세그먼트하거나 큐 관리자가 대신 수행하도록 허용하는 경우에만 전체 메시지에 대해 단일 보고서를 수신할 것으로 예상할 수 있습니다. 이는 COD 보고서만 요청했으며 가져오기 애플리케이션에 *MQGMO_COMPLETE_MSG*를 지정한 경우입니다.

다른 경우에는 애플리케이션이 여러 보고서를 처리하도록 준비되어 있어야 합니다. 보통 각 세그먼트당 하나씩입니다.

참고: 메시지를 세그먼트하고 원래 메시지의 처음 100바이트만 리턴해야 하는 경우 100개 이상의 오프셋이 있는 세그먼트에 대해 데이터가 없는 보고서를 요청하도록 보고서 옵션의 설정을 변경하십시오. 이를 수행하지 않고, 각 세그먼트에서 100바이트의 데이터를 요청하도록 설정을 그대로 둔 상태로 *MQGMO_COMPLETE_MSG*를

지정하는 단일 MQGET을 사용하여 보고 메시지를 검색하는 경우, 보고서는 각각의 해당 오프셋에서 100바이트의 읽기 데이터를 포함하는 대형 메시지를 작성합니다. 이러한 경우 대형 버퍼가 필요하거나 MQGMO_ACCEPT_TRUNCATED_MSG를 지정해야 합니다.

애플리케이션에서 생성하는 보고서의 경우

애플리케이션에서 보고서를 생성하는 경우 원본 메시지 데이터의 시작 부분에 표시되는 IBM MQ 헤더를 보고 메시지 데이터로 항상 복사하십시오.

그런 다음, 없음, 100바이트 또는 모든 원본 메시지 데이터(또는 일반적으로 포함하는 다른 양)를 보고 메시지 데이터에 추가하십시오.

MQMD로 시작하고 현재 헤더까지 계속되는 연속적인 형식 이름을 확인하여 복사되어야 하는 IBM MQ 헤더를 인식할 수 있습니다. 다음 Format 이름은 이러한 IBM MQ 헤더를 표시합니다.

- MQMDE
- MQDLH
- MQXQH
- MQIIH
- MQH*

MQH*는 문자 MQH으로 시작하는 모든 이름을 의미합니다.

Format 이름은 MQDLH 및 MQXQH에 대한 특정 위치에서 발생하지만 다른 IBM MQ 헤더의 경우 동일한 위치에서 발생합니다. 헤더의 길이는 MQMDE, MQIMS 및 모든 MQH* 헤더에 대한 동일한 위치에서 역시 발생하는 필드에 포함됩니다.

버전 1 MQMD를 사용 중이며 세그먼트, 그룹의 메시지 또는 세그먼트화가 허용되는 메시지에서 보고하는 경우 보고서 데이터는 MQMDE로 시작해야 합니다. *OriginalLength* 필드를 발견하는 IBM MQ 헤더의 길이를 제외한 원본 메시지 데이터의 길이로 설정하십시오.

보고서 검색

COA 또는 COD 보고서를 요청하는 경우 보고서가 MQGMO_COMPLETE_MSG로 리어셈블링되도록 요청할 수 있습니다.

MQGMO_COMPLETE_MSG가 포함된 MQGET은 하나의 완전한 원본 메시지를 표현하기 위해 충분한 보고 메시지(COA와 같은 단일 유형이고 동일한 *GroupId*가 있는 메시지)가 큐에 제공되는 경우에 충족됩니다. 이는 보고 메시지 자체에 전체 원본 데이터가 포함되어 있지 않은 경우에도 동일합니다. 데이터 자체가 없는 경우라도 각 보고 메시지의 *OriginalLength* 필드는 해당 보고 메시지에서 표현하는 원래 데이터의 길이를 제공합니다.

동일한 *Feedback* 코드를 갖는 경우에만 MQGMO_COMPLETE_MSG가 포함된 MQGET에서 보고 메시지를 리어셈블링하기 때문에 큐에 여러 다른 유형의 보고서 유형이 있는 경우(예를 들어, COA 및 COD 둘 다)에도 이 기술을 사용할 수 있습니다. 그러나 예외 보고서에는 이 기술을 사용할 수 없습니다. 일반적으로 예외 보고서에는 다른 *Feedback* 코드가 있기 때문입니다.

전체 메시지가 도착했다는 긍정적인 표시를 얻기 위해 이 기술을 사용할 수 있습니다. 그러나 대부분의 상황에서 다른 세그먼트가 예외(또는 종료, 허용한 경우)를 생성할 수 있는 동안 일부 세그먼트의 도달 가능성을 만족시켜야 합니다. 이러한 경우 MQGMO_COMPLETE_MSG를 사용할 수 없습니다. 일반적으로 다른 세그먼트에 대해서는 다른 *Feedback* 코드를 얻을 수 있으며 세그먼트에 대해 하나 이상의 보고서를 얻을 수 있기 때문입니다. 그러나 MQGMO_ALL_SEGMENTS_AVAILABLE은 사용할 수 있습니다.

이를 허용하려면, 보고서가 도착할 때 보고서를 검색해야 할 수 있으며 원래 메시지에 발생한 내용의 그림을 애플리케이션에서 빌드해야 할 수 있습니다. 보고 메시지의 *GroupId* 필드를 사용하여 원래 메시지의 *GroupId*와 보고서를 상관시키고 *Feedback* 필드를 사용하여 각 보고 메시지의 유형을 식별할 수 있습니다. 이를 수행하는 방법은 애플리케이션 요구사항에 따라 다릅니다.

한 가지 접근법은 다음과 같습니다.

- COD 보고서 및 예외 보고서를 요청하십시오.

- 정해진 시간이 지난 후 MQGMO_COMPLETE_MSG를 사용하여 전체 COD 보고서 세트가 수신되었는지 확인하십시오. 해당되는 경우 애플리케이션은 전체 메시지가 처리되었음을 인지하고 있습니다.
- 수신되지 않고 이 메시지와 관련된 예외 보고서가 있는 경우에는 세그먼트되지 않은 메시지에 대한 문제점을 처리하십시오. 그러나 일부 지점에서 고아(orphan) 세그먼트를 정리해야 합니다.
- 어떠한 종류의 보고서도 없는 세그먼트가 있다면 원래 세그먼트가 채널이 다시 연결되기를 기다리고 있거나 네트워크에서 일부 지점에 과부하가 발생했을 수 있습니다. 예외 보고서가 전혀 수신되지 않았다면(또는 임시 보고서만 수신되었다고 생각하는 경우) 애플리케이션을 좀 더 대기하도록 결정할 수 있습니다.

이전과 같이, 고아 세그먼트를 정리해야 할 수 있다는 점을 제외하고는 세그먼트되지 않은 메시지를 처리할 때의 고려사항과 유사합니다.

원본 메시지가 중요하지 않은 경우(예를 들어, 나중에 반복될 수 있는 메시지나 조회인 경우) 고아 세그먼트가 제거되었는지 확인하도록 만기 시간을 설정하십시오.

이전 레벨 큐 관리자

세그먼트화를 지원하는 큐 관리자가 보고서를 생성하지만 세그먼트화를 지원하지 않는 큐 관리자에서 보고서를 수신하는 경우 MQMDE 구조(보고서에 의해 표시되는 *Offset* 및 *OriginalLength*를 식별함)는 0(제로), 100바이트 또는 메시지의 원래 데이터 모두와 함께 항상 보고서 데이터에 포함됩니다.

그러나 메시지의 세그먼트가 세그먼트화를 지원하지 않는 큐 관리자를 통해 전달되는 경우 보고서가 해당 큐 관리자에 생성된다면 원래 메시지의 MQMDE 구조는 순수하게 데이터로 처리됩니다. 따라서 원래 데이터의 0바이트가 요청되었다면 보고서 데이터에 포함되지 않습니다. MQMDE 없이는 보고 메시지가 유용하지 않을 수 있습니다.

메시지가 이전 레벨 큐 관리자를 통해 이동할 가능성이 있는 경우 보고서에서 최소한 100바이트 이상의 데이터를 요청하십시오.

메시지 데이터 변환

환경 간에 문자 및 숫자를 서로 다른 방식으로 표현하는 경우 메시지를 IBM MQ 전송할 때 데이터 변환이 필요합니다. 예를 들어, 서로 다른 문자 인코딩 표준(Linux 예: ASCII 인코딩 표준과 EBCDIC 인코딩 표준) z/OS 을 사용하는 운영 체제 간에 메시지를 전송할 수 있습니다. 서로 다른 문자 집합을 사용하는 지역 설정을 가진 컴퓨터들 사이에서 메시지를 전송할 수 있습니다. 예를 들어 독일어와 일본어 같은 경우입니다. 또한 숫자에 대해 다른 바이트 순서를 사용하는 시스템 간에 메시지를 전송할 수도 있습니다. 또한 메시지의 형식이 데이터 변환에 영향을 미칠 수 있습니다. 예를 들어, IBM MQ 메시지에는 CICS 헤더와 애플리케이션 데이터가 CICS 포함되어 있으므로, 텍스트만 포함된 메시지와는 다르게 처리되어야 합니다.

IBM MQ 데이터 변환이 필요하지 않은 경우에도(예: 데이터 표현 방식이 동일한 두 시스템 간에 메시지를 전송하는 경우) 데이터 변환 루틴을 실행합니다. 이러한 루틴들은 작업이 필요하지 않다고 판단할 수 있지만, 메시지 데이터가 처리될 때마다 항상 xcsQueryCCSIDTypeIBM MQ 루틴 호출이 추적 IBM MQ 기록에 표시됩니다.

데이터 변환에 필요한 정보

데이터 변환이 필요한 경우, 메시지(또는 메시지를 처리하는 큐 IBM MQ 관리자) 내의 다음 정보들을 활용하여 수행할 변환 방식을 결정합니다.

코드화 문자 세트 ID

코딩된 문자 집합 식별자(CCSID)는 문자 데이터를 고유하게 식별하는 인코딩 체계 식별자, 문자 집합 식별자, 코드 페이지 식별자 및 기타 정보를 포함하는 16비트 숫자입니다. 이러한 식별자는 문자 데이터 식별 및 데이터 변환 과정 중 데이터 무결성 유지를 위한 통합 데이터 모델(IBM) 아키텍처의 일부입니다. 이 아키텍처에 대한 자세한 정보(등록된 CCSID 목록 포함)는 온라인에서 제공되는 [문자 데이터 표현 아키텍처 PDF](#)를 참조하십시오.

CCSID는 문자 및 비문자 데이터를 '00'부터 'FF'까지의 16진수 값에 할당합니다(또는 DBCS(이중 바이트 문자 집합) CCSID의 경우 '0000'부터 'FFFF'까지). DBCS CCSID는 한자처럼 256개 위치로 표현할 수 없는 문자를 위한 것입니다. CCSID 변환 테이블은 한 CCSID에서 다른 CCSID로 변환하는 데 사용되며, 일반적으로 문자 데이터와 제어 문자를 변환합니다. 잘 알려진 CCSID로는 1208(UTF-8)과 500(주로 EBCDIC CCSID로 사용됨) z/OS이 있습니다. 지원되는 CCSID 목록은 [코드 페이지 변환](#)을 참조하십시오.

기계 인코딩

데이터 변환의 맥락에서 인코딩이란 플랫폼이 숫자 데이터를 표현하는 데 사용하는 방식을 의미합니다. 숫자에서 최하위 바이트와 최상위 바이트의 순서는 플랫폼의 엔디안 방식에 따라 달라질 수 있습니다. 예를 들어, z/OS IBM Z 하드웨어에서는 빅엔디안(숫자의 최하위 바이트가 가장 높은 주소를 가지며 최상위 바이트가 가장 낮은 주소를 가짐)인 반면, x86 Windows 하드웨어에서는 리틀엔디안(빅엔디안과 비교하여 바이트 순서가 반대로 배열됨)입니다. 따라서, 숫자 437은 16진수 체계에서 X'01BF'로, z/OS 그리고 X'BF01'로 표현됩니다. Windows, IBM Power® 리틀 엔디안 또는 빅 엔디안일 수 있습니다. 따라서 동일한 CCSID를 사용하는 플랫폼 간에 전달되는 경우에도 숫자 데이터는 변환이 필요할 수 있습니다.

에서 IBM MQ 정수, 압축 십진수 및 부동 소수점 인코딩을 나타내는 값들이 결합되어 플랫폼의 기계 인코딩을 나타내는 단일 값을 생성합니다. 개별 값은 다음 기계 인코딩을 포함합니다:

- 일반 (빅엔디안)
- 역순 (리틀 엔디안)
- IEEE 표준 (표준 IEEE 부동 소수점 형식, 빅엔디안)
- IEEE 역순 (표준 IEEE 부동 소수점 형식, 리틀 엔디안)
- System/390 (IBM Z 부동 소수점 형식)

자세한 내용은 [기계 인코딩](#)을 참조하십시오.

형식

포맷 은 메시지 IBM MQ 발신자가 수신자에게 메시지 내 데이터의 성격을 알리기 위해 사용하는 명칭이다.

IBM MQ MQ로 시작하는 내장 메시지 형식이 몇 가지 있습니다. 예를 들어, MQFMT_STRING. 이러한 형식을 사용하여 메시지가 다음을 포함함을 나타낼 수 있습니다: 문자열, CICS 데이터, 이벤트에 IBM MQ 대한 정보 또는 수신 큐 관리자가 실행할 프로그래밍 가능 명령 형식(PCF) 명령 등. 사용자 정의 형식을 정의할 수도 있습니다.

큐 관리자는 메시지의 형식을 설정할 수도 있습니다. 예를 들어, 큐 관리자가 메시지를 전달할 수 없을 때, 메시지 전달 실패를 나타내기 위해 MQFMT_DEAD_LETTER_HEADER 형식의 새 헤더를 메시지 시작 부분에 추가한 후, 해당 메시지를 데드 레터(전달 실패 메시지) 큐에 배치합니다.

무엇이 IBM MQ 전환을 일으키며, 그 이유는 무엇인가

An IBM MQ 메시지는 다음 부분을 포함합니다:

- 메시지 디스크립터: 메시지를 식별하며, 우선순위 및 만료 시간과 같은 다양한 제어 정보, 메시지를 전송한 애플리케이션에 대한 정보, 메시지 내에 포함된 애플리케이션 데이터에 대한 정보(CCSID, 기계 인코딩 및 형식 포함)를 포함합니다.
- 응용 프로그램 데이터: 메시지가 전송하는 데이터로, 수신 응용 프로그램에서 처리될 데이터.

IBM MQ 다음과 같은 이유로 이 두 메시지 부분을 독립적으로 변환합니다:

채널 통신을 위해

각 큐 관리자는 CCSID를 가지며, 이는 큐 관리자 생성 시 자동으로 설정됩니다(이후 수정 가능). 자세한 내용은 '다중 플랫폼에서 큐 관리자 생성'을 참조하십시오. 두 IBM MQ 큐 관리자 간 채널이 시작될 때, 두 큐 관리자는 채널 통신에 사용할 CCSID 및 기계 인코딩을 협상합니다. 이 CCSID 및 인코딩 방식은 전송 세그먼트 IBM MQ 헤더(트레이스에서 IBM MQ 확인하면 큐 관리자 간에 전달되는 제어 IBM MQ 블록)와 메시지 설명자를 인코딩하는 데 사용됩니다. 메시지 디스크립터는 애플리케이션이 메시지를 큐에 배치할 때 애플리케이션에 의해 생성되고 채워집니다. 메시지 디스크립터의 데이터는 로컬 큐 관리자의 문자 집합 및 인코딩(큐 관리자의 CCSID와 로컬 머신의 인코딩에 의해 결정됨)으로 되어 있어야 합니다. 또는 애플리케이션이 클라이언트 IBM MQ MQI client 애플리케이션인 경우 클라이언트 시스템의 문자 집합 및 인코딩으로 되어 있어야 합니다(참조 918 페이지의 『클라이언트 또는 서버 CCSID 선택』). IBM MQ 메시지 디스크립터가 생성된 플랫폼과 관계없이 디스크립터의 내용을 이해해야 하므로, 메시지 IBM MQ 디스크립터를 자동으로 변환합니다.

메시지 디스크립터의 제어 정보는 모두 문자 데이터가 아닙니다. 일부 정보는 (사용 중인 CCSID에 따라) 문자 데이터로 변환되고, 일부는 (사용 중인 인코딩에 따라) 숫자 데이터로 변환되며, 일부는 변환되지 않은 상태로 전달됩니다. 다음 표에 그 내용을 나타냅니다:

필드 이름	필드 길이 (바이트 단위)	오프셋	변환됨
StrucId	4	'00'x	문자
버전	4	'04'x	숫자
보고서	4	'08'x	숫자
MsgType	4	'0C'x	숫자
만기	4	'10'x	숫자
Feedback	4	'14'x	숫자
Encoding	4	'18'x	숫자
CodedCharSetId	4	'1C'x	숫자
형식	8	'20'x	문자
Priority	4	'28'x	숫자
Persistence	4	'2C'x	숫자
MsgId	24 ('18'x)	'30'x	Byte
CorrelId	24 ('18'x)	'48'x	Byte
BackoutCount	4	'60'x	숫자
ReplyToQ	48 ('30'x)	'64'x	문자
ReplyToQMgr	48 ('30'x)	'94'x	문자
UserIdentifier	12 ('0C'x)	'C4'x	문자
AccountingToken	32 ('20'x)	'D0'x	Byte
ApplIdentityData	32 ('20'x)	'F0'x	문자
PutApplType	4	'110'x	숫자
PutApplName	28 ('1C'x)	'114'x	문자
PutDate	8	'130'x	문자
PutTime	8	'138'x	문자
ApplOriginData	4	'140'x	문자
GroupId	24 ('18'x)	'144'x	Byte
MsgSeqNumber	4	'15C'x	숫자
오프셋	4	'160'x	숫자
MsgFlags	4	'164'x	숫자
OrginalLength	4	'168'x	숫자

참고: MsgId, CorrelId, Accounting Token 및 GroupId 필드는 문자 데이터로 간주되지 않으므로 해당 방식으로 변환되지 않습니다. 많은 애플리케이션이 이러한 필드에 문자 데이터를 입력하지만, 메시지가 ASCII와 EBCDIC 플랫폼 간에 전송되고 수신 애플리케이션이 이러한 필드를 읽을 수 있어야 하는 경우, 애플리케이션 (또는 사용자 종료)이 이러한 필드를 변환해야 합니다. IBM MQ 이러한 필드의 데이터 형식을 제한하지 않습니다. 예를 들어, MsgId 필드 생성을 IBM MQ 허용하는 경우, 해당 필드는 문자 데이터로만 엄격히 제한되지 않습니다.

메시지 디스크립터에 대한 자세한 정보는 [MQMD - 메시지 디스크립터](#)의 내용을 참조하십시오.

수신 애플리케이션이 사용할 수 있도록 애플리케이션 데이터를 처리하기 위해

일반적으로 애플리케이션 데이터는 자동으로 변환되지 않습니다 IBM MQ . 변환되지 않은 상태로 전달해야 할 수 있기 때문입니다.(Java 애플리케이션은 예외이며, 이는 후술합니다). 따라서 애플리케이션 데이터의 변환을 요청해야 합니다.

보내는 애플리케이션은 메시지를 큐에 넣기 전에 적절한 메시지 설명자 필드를 설정하여 애플리케이션 데이터의 CCSID, 인코딩 및 형식을 지정합니다. 응용 프로그램은 실제 값이나 IBM MQ 특수 값을 지정할 수 있습니다. 특수 값은 문자열로, 예를 들어 데이터가 메시지를 전송하는 큐 관리자와 동일한 CCSID를 사용하거나, 데이터가 응용 프로그램이 실행 중인 머신과 동일한 인코딩을 사용함을 나타냅니다. 응용 프로그램 데이터에는 헤더가 포함될 수 있으며, 이 헤더는 뒤따르는 응용 프로그램 데이터에 대해 서로 다른 CCSID, 인코딩 및 형식 정보를 지정합니다. MQI 애플리케이션의 해당 필드에 대한 자세한 내용은 [MQMD 필드 세부 정보](#)의 ‘CodedCharSetId’, ‘인코딩’ 및 ‘형식’ 섹션을 참조하십시오. (다른 애플리케이션 유형에는 다를 수 있지만 동등한 옵션이 있을 수 있습니다. 예를 들어, 를 사용하는 [150 페이지의 『JMS 필드와 대응하는 MQMD 필드가 있는 특성』](#) 애플리케이션에는 에 나열된 대로 IBM MQ classes for JMS JMS_IBM_Character_Set 속성이 있습니다.)

다음 예시 IBM MQ 메시지는 메시지 내 애플리케이션 데이터의 CCSID가 850, 기계 인코딩이 546, 형식이 MQEVENT(이 메시지가 메시지 큐 이벤트(MQ)에 대한 정보를 포함함을 나타냄)임을 보여줍니다:

```
****Message descriptor****

StrucId  : 'MD ' Version : 2
Report   : 0 MsgType : 8
Expiry   : -1 Feedback : 0
Encoding : 546 CodedCharSetId : 850
Format   : 'MQEVENT '
Priority : 0 Persistence : 0
MsgId    : X'414D512073617475726E2E71756575650005D30033563DB8'
CorrelId  : X'0000000000000000000000000000000000000000000000000000000000000000'
BackoutCount : 0
ReplyToQ   : '
ReplyToQMgr : 'saturn.queue.manager'
** Identity Context
UserIdentifer : '
AccountingToken :
X'0000000000000000000000000000000000000000000000000000000000000000'
ApplIdentityData : '
** Origin Context
PutApplType : '7'
PutApplName : 'saturn.queue.manager'
PutDate : '19970417' PutTime : '15115208'
ApplOriginData : '

GroupId : X'0000000000000000000000000000000000000000000000000000000000000000'
MsgSeqNumber : '1'
Offset : '0'
MsgFlags : '0'
OriginalLength : '104'

**** Message ****

length - 104 bytes

00000000: 0700 0000 2400 0000 0100 0000 2C00 0000 '....&raarr;.....'
00000010: 0100 0000 0100 0000 0100 0000 AE08 0000 '.....'
00000020: 0100 0000 0400 0000 4400 0000 DF07 0000 '.....D.....'
00000030: 0000 0000 3000 0000 7361 7475 726E 2E71 '....0...saturn.q'
00000040: 7565 7565 2E6D 616E 6167 6572 2020 2020 'ueue.manager'
00000050: 2020 2020 2020 2020 2020 2020 2020 2020
00000060: 2020 2020 2020 2020
```

수신 애플리케이션은 큐에서 메시지를 수신할 때 데이터 변환을 요청할 수 있습니다. 예를 들어, MQI 애플리케이션에서는 [MQGET](#) 호출 시 [MQGMO_CONVERT](#) 옵션을 지정하여 요청 데이터를 변환할 수 있습니다. 큐 관리자는 변환이 필요한지 확인하고, 필요한 경우 데이터를 변환합니다. 이러한 검사에는 예를 들어, 메시지 설명자의 CCSID 및 인코딩이 수신 애플리케이션이 기대하는 CCSID 및 인코딩과 일치하는지 여부가 포함됩니다. 수신 측 `MQI` 애플리케이션은 [MQGET](#) 호출 시 `MsgDesc` 매개변수를 설정하여 예상하는 값을 지정합니다. MQ 가 메시지를 수신할 때 수행하는 검사 및 데이터 변환 프로세스에 대한 자세한 내용은 [‘변환 처리’](#)를 참조하십시오.

애플리케이션 데이터의 CCSID 및 인코딩이 지원되는 경우 IBM MQ ([코드 페이지 변환 및 기계 인코딩 참조](#)), 그리고 형식이 내장 IBM MQ 형식인 경우, 내장 변환 루틴을 사용하여 IBM MQ 데이터 변환을 수행할 수

있습니다. 그렇지 않으면 애플리케이션 프로그래머가 데이터 변환 출구 (data conversion exit)로 알려진 사용자 출구 프로그램에서 변환 코드를 제공해야 합니다. IBM MQ 이러한 사용자 출구를 생성하는 데 도움이 되는 기본 구조와 유틸리티를 제공합니다. 추가 정보는 [729 페이지의 『데이터 변환 엑시트 작성』](#)의 내용을 참조하십시오.

메시지 내 데이터 유형에 따른 데이터 변환 접근 방식의 예시는 다음과 같습니다:

메시지는 완전히 숫자 데이터입니다

응용 프로그램 데이터가 완전히 숫자 데이터이고, 메시지를 주고받는 두 플랫폼의 숫자 인코딩 방식이 동일하다는 것을 알고 있다면, 데이터 변환이 필요하지 않습니다. 두 플랫폼의 숫자 데이터 인코딩 방식이 다를 수 있다고 의심되거나, 향후 그러한 상황이 발생할 가능성을 대비하는 경우, 데이터를 변환하기 위해 IBM MQ 사용할 데이터 변환 출구(exit)를 제공하십시오.

메시지는 완전히 문자 데이터입니다

애플리케이션 데이터가 완전히 문자 데이터인 경우, 변환을 IBM MQ 수행할 수 있습니다. 메시지를 전송하는 MQI 애플리케이션에서 MQMD 메시지 설명자의 Format 필드를 MQFMT_STRING으로 설정하십시오. 이는 문자열 값 "MQSTR"과 동일한 상수입니다. 이 값은 메시지가 전적으로 문자열 데이터로 구성됨을 IBM MQ 나타냅니다.

메시지는 숫자와 문자 데이터가 혼합되어 있습니다

메시지가 숫자와 문자 데이터가 혼합된 경우, 데이터를 변환하기 위해 IBM MQ 사용할 데이터 변환 출구를 제공해야 합니다.

참고: 응용 프로그램에서, IBM MQ classes for JMS, 또는 IBM MQ classes for Java IBM MQ classes for Jakarta Messaging를 사용하는 경우, 데이터 변환은 코드에서 Java 자동으로 수행됩니다. 예를 들어 메시지 텍스트를 가져오기 위해 Java 메서드를 사용할 때와 같습니다. C 애플리케이션과 같은 다른 애플리케이션과의 이러한 차이는, 애플리케이션이 Java 숫자와 텍스트를 내부 표현과 메시지 사용을 위한 인코딩된 형식 사이에서 변환해야 하기 때문입니다. 대기열 관리자를 통해 데이터 변환을 요청할 수는 있지만, 데이터가 두 번 변환되므로 원하지 않을 수 있습니다. 메시지 처리에 IBM MQ classes for Java 대한 자세한 내용은 [참조하십시오 350 페이지의 『IBM MQ classes for Java의 메시지 핸들링』](#). 와 IBM MQ classes for JMS 에서 사용할 수 있는 데이터 변환에 IBM MQ classes for Jakarta Messaging대한 자세한 내용은 [참조하십시오 160 페이지의 『JMS 메시지 변환』](#).

대안으로, 시스템 관리자는 메시지 전송 시 발신자, 서버, 클러스터 수신자 IBM MQ 또는 클러스터 발신자 채널의 CONVERT 옵션을 YES로 설정하여 변환을 요청할 수 있습니다. 메시지는 수신 큐 관리자의 CCSID를 사용하여 변환됩니다. 메시지 채널을 통한 데이터 변환은 일반적으로 다른 형태의 변환을 사용할 수 없는 경우에만 사용됩니다.

메시지를 보내고 받을 때의 일반적인 관행

메시지가 문자, 숫자 또는 이 둘의 조합이든 상관없이, MQPUT 명령어를 사용하여 큐에 메시지를 삽입하거나 MQGET 명령어를 사용하여 메시지를 가져오는 MQI 애플리케이션을 작성할 때는 다음 설정을 일반적으로 적용합니다. 이러한 설정에 대한 자세한 내용은 MQMD의 필드 세부 정보를 참조하십시오.

- MQMD.CodedCharSetId 필드를 MQCCSI_Q_MGR로 설정하면 메시지가 애플리케이션이 사용하는 큐 관리자로 부터 CCSID를 상속받습니다. (MQCCSI_INHERIT는 동일한 의미를 가지지만 일부 시나리오에서는 사용할 수 없습니다.) 이 설정은 메시지 디스크립터의 'CodedCharSetId' 필드가 입력 및 출력 필드 모두이기 때문에 일반적인 관행입니다. 따라서 애플리케이션이 MQPUT 호출과 MQGET 호출 모두에 동일한 MQMD 구조를 사용하는 경우, 수신하는 모든 메시지의 IBM MQ CCSID를 이 필드에 저장합니다. 이러한 메시지 중 하나가 큐 관리자와 동일한 CCSID를 사용하지 않았고, 애플리케이션이 메시지를 변환하지 않은 경우, 이 필드는 애플리케이션이 다음 MQPUT 호출을 수행할 때 예상되지 않는 값으로 변경됩니다. 응용 프로그램이 수신 메시지를 변환하려고 시도했다라도, MQGET 호출이 예상치 못한 반환 코드(예: MQRC_TRUNCATED_MSG_ACCEPTED)를 반환한 경우 메시지는 변환되지 않은 상태로 유지되며, 메시지 코드 시스템(CodedCharSetId) 필드는 변환되지 않은 메시지의 CCSID로 설정됩니다.

참고: 클라이언트 애플리케이션에서 MQCCSI_Q_MGR은 원격 큐 관리자의 CCSID가 아닌 애플리케이션 로케일의 CCSID를 가리킵니다. 모든 경우에, 메시지 관리 코드(MQMD.CodedCharSetId) 필드를 MQCCSI_Q_MGR로 설정할 때, 애플리케이션은 메시지 내 문자열 데이터를 CCSID가 올바르게 표현하도록 보장해야 합니다.

응용 프로그램이 큐에서 메시지를 수신할 때, 메시지 CodedCharSetId 디스크립터의 값을 MQGET 호출 시 응용 프로그램이 'MsgDesc' 매개변수로 지정한 값과 비교해야 합니다. 두 값이 다를 경우, 애플리케이션은

메시지 내 문자 데이터를 변환하거나 사용 가능한 데이터 변환 출구(exit)가 있다면 이를 사용해야 할 수 있습니다.

- MQMD.Encoding 필드를 MQENC_NATIVE로 설정하십시오. 이 설정은 메시지 데이터의 인코딩이 애플리케이션이 실행 중인 시스템의 인코딩과 동일함을 지정합니다. 애플리케이션이 큐에서 메시지를 수신할 때, 메시지 디스크립터의 Encoding 필드 값을 애플리케이션이 실행 중인 머신의 상수 MQENC_NATIVE 값과 비교해야 합니다. 두 값이 다를 경우, 애플리케이션은 메시지 내의 숫자 데이터를 변환하거나 사용 가능한 데이터 변환 출구(exit)를 사용해야 할 수 있습니다.
- 메시지 내 데이터의 특성에 따라 'MQMD.Format' 필드를 설정하십시오. 내장된 형식(이름이 로 시작하는 MQ) 중 하나를 사용하거나, 적합한 형식이 없을 경우 사용자 정의 형식을 정의할 수 있습니다(이름은 로 시작해서는 MQ 안 됨). 사용자 정의 형식을 생성하고 사용할 때는, 애플리케이션이 MQGMO_CONVERT 옵션을 지정할 때 메시지를 수신할 수 있도록 데이터 변환 출구(exit)를 작성해야 합니다. 내장 형식 중 하나인 MQFMT_NONE은 데이터의 본질이 정의되지 않았음을 나타냅니다. 응용 프로그램이 MQGMO_CONVERT 옵션을 지정하여 요청 하더라도 이러한 메시지를 변환하지 않습니다 IBM MQ .
- 데이터를 전송할 때(예: MQI MQPUT 호출)가 아니라 수신할 때(예: MQI MQGET 호출) 변환하십시오. 메시지는 목적지에 도달하기 위해 서로 다른 플랫폼 간에 여러 번 전송될 수 있습니다. 각 전송 단계마다 데이터 변환을 수행하면 자원을 낭비할 수 있으며, 전송 중인 큐 관리자에서 데이터 변환이 실패할 경우 각 전송 단계마다 예상치 못한 채널 오류가 발생할 위험이 있습니다. 그러나 어떤 상황에서는 메시지를 보낼 때 데이터를 변환하는 것이 더 나을 수 있습니다. 예를 들면, 다음과 같습니다.
 - 수신 애플리케이션이 메시지를 변환할 수 없는 경우. 예를 들어, 애플리케이션이나 수신 큐 관리자가 올바른 문자 집합을 가지고 있지 않습니다.
 - 애플리케이션이 JMS 비(非)JMS 애플리케이션에 메시지를 보낼 때. 메시지에 텍스트나 숫자가 포함된 경우, 발신 애플리케이션은 어쨌든 해당 데이터를 메시지 내에서 사용하기 위한 인코딩된 형식으로 변환하기 위해 데이터의 원시 표현(JVM)을 변환해야 합니다. 이 변환으로 수신 애플리케이션에서 사용할 수 있는 데이터가 생성된다면, 수신 측에서의 변환이 필요하지 않으므로 이 접근 방식이 더 효율적이다.

메시지 전송 시 변환을 수행하고 사용자 정의 데이터 변환 종료 프로그램을 사용하는 경우, 해당 프로그램은 발신자 채널이 정의된 시스템에 존재해야 합니다.

- 애플리케이션이 다국어 지원이라면, 애플리케이션이 큐에서 수신하는 메시지의 CCSID를 사전에 알 수 없을 수 있습니다. 예를 들어, 한국어로 된 메시지와 독일어로 된 메시지를 모두 ASCII로 기계에 IBM i 전송하여 EBCDIC으로 변환할 수 있습니다. 그러나 독일어의 악센트 부호가 있는 문자나 한국어의 멀티바이트 문자는 EBCDIC에서 단일 CCSID로 표현할 수 없다. 한 가지 해결책은 각 언어별로 별도의 큐를 사용하는 것입니다. 이렇게 하면 애플리케이션이 메시지를 수신할 때 적절한 CCSID를 지정할 수 있습니다.

잠재적 문제점

- 제공된 IBM MQ 매크로는 모든 구조체가 압축되어 있을 것을 가정합니다. 그러나 대부분의 플랫폼에서 C 언어는 구조체 내 문자 데이터와 숫자 데이터 사이의 경계가 단어 경계에 맞지 않을 경우 패딩 문자를 삽입하는 경향이 있습니다. 이 문제를 애플리케이션에서 해결할 수 있습니다:

- 다음과 IBM MQ for z/OS같이 데이터 구조를 선언하십시오:

```
_Packed struct_name xxxx
```

자세한 내용은 사용 중인 버전의 문서에서 *_Packed* 한정자에 z/OS 관한 내용을 참조하십시오(예: [3.1.0](#)).

- 다음 컴파일러 옵션을 지정하십시오 IBM MQ for AIX:

```
-qalign=packed
```

자세한 내용은 사용 중인 버전의 문서에서 데이터 AIX 정렬을 참조하십시오(예: [16.1.0](#)).

- 다음 IBM MQ for Windows 중 하나의 줄을 코드에서 프래그마 지시문으로 사용하십시오:

```
#pragma _Packed or  
#pragma pack(1)
```

자세한 정보는 Windows 문서를 참조하십시오. 예를 들어, [pack 프래그마](#).

- INT 및 LONG 데이터 유형의 사용을 피하십시오. 이들의 길이는 아키텍처와 운영 체제에 따라 달라집니다. 대신 MQINT64 및 MQLONG과 같은 데이터 IBM MQ 유형을 사용하십시오.

- 스투딩 모델과 비스레딩 모델을 모두 지원하거나 여러 서로 다른 스투딩 모델을 지원하는 플랫폼에서는 UNIX 각 모델별로 데이터 변환 출구(exit)를 컴파일하십시오. 그렇지 않으면, 다른 애플리케이션이나 채널 IBM MQ 코드(송신자 채널 정의에서 CONVERT(YES)를 지정한 경우)에 의해 호출될 때 출구가 발견되지 않을 수 있으며, MQRC_FORMAT_ERROR(2110, x'83E') 오류나 해당 출구의 버전 _r (또는 _d 버전)을 찾을 수 없다는 메시지가 발생할 수 있습니다. 자세한 내용은 [AIX 또는 Linux 시스템용 IBM MQ 데이터 변환 종료 프로그램 작성하기](#)를 참조하십시오.
- 유니코드로 애플리케이션 데이터를 보내거나 받으려면 몇 가지 제한 사항이 적용됩니다. 응용 프로그램이 메시지를 넣거나 가져올 때 메시지 설명자에 유니코드 CCSID를 지정하여 데이터 변환을 트리거해야 합니다. 발신 채널의 [CONVERT](#) 옵션을 YES로 설정하여 데이터 변환을 트리거하지 마십시오. 이 변환은 항상 수신 큐 매니저의 CCSID로 수행되며, 큐 매니저 CCSID는 UTF-16 또는 일부 플랫폼에서 UTF-8 일 수 없습니다. 다국어 애플리케이션을 지원하기 위해 유니코드를 사용하는 두 가지 접근 방식은 다음과 같습니다:
 - 클라이언트 또는 서버에서 모든 메시지 데이터를 유니코드로 변환하거나 유니코드에서 변환하고, 전송 계층으로 IBM MQ 사용하십시오. 모든 클라이언트 또는 서버 처리는 유니코드로 수행됩니다. 이 접근 방식은 변환 IBM MQ 서비스를 사용하지 않으며, 유니코드와 운영 체제 코드 페이지(예: AIX 또는 Windows) 간에 우수한 운영 체제 변환기를 보유한 클라이언트 및 서버에 특히 적용됩니다.
 - 변환을 수행하려면 IBM MQ 를 사용하십시오. 이 접근 방식은 양 끝단의 큐 관리자가 유니코드 변환을 수행할 수 있어야 합니다.

다양한 플랫폼에서의 유니코드 지원에 IBM MQ 관한 정보는 [유니코드 변환 지원 항목](#)을 참조하십시오.

- EBCDIC와의 문자 변환 시(특히 느낌표) 문제가 발생하면, 큐 관리자의 z/OS CCSID가 애플리케이션이 데이터를 기대하는 코드페이지와 일치하는지 확인하십시오. 대체 CCSID인 500에는 다른 EBCDIC CCSID의 16진수 값과 다른 값을 가진 다양한 문자가 포함되어 있습니다. 이러한 차이점 목록은 [z/OS conversion support](#)를 참조하십시오. 이러한 문제가 발생할 경우, CSQ6SYSP 매크로를 사용하여 QMCCSID 매개변수를 설정함으로써 큐 z/OS 매니저의 CCSID를 변경하는 것을 고려하십시오. 기본 CCSID에 대한 설명 및 사용 중인 CCSID 확인 방법을 포함한 자세한 내용은 '[CSQ6SYSP 사용](#)'을 참조하십시오.
- 채널 메시지 표시 시 가끔 발생하는 변환 문제 보고를 받는 경우 AIX, 큐 관리자가 일본어와 같은 비라틴 로케일에서 실행 중이라면, C 로케일에서 실행 중인 **inetd** 데몬이 시작하는 리스너 프로세스 때문일 수 있습니다. 기본 데이터 변환을 활성화하거나 **inetd** 데몬이 호출하는 프로그램을 셸 스크립트로 수정함으로써 이 문제를 피할 수 있습니다. 셸 스크립트는 로케일 환경 변수를 설정한 후 리스너 IBM MQ 프로그램을 호출해야 합니다.
- Microsoft Microsoft 웹사이트의 '[Windows 앱에서 UTF-8 코드 페이지 사용](#)' 문서에 설명된 대로, 프로그래머는 UTF-8 코드 페이지를 사용해야 한다고 권장합니다. 그러나 응용 프로그램은 다른 코드 페이지를 사용할 수 있으며, 문자 표현이 다를 수 있는 두 Windows 그룹의 시스템 코드 페이지를 지원합니다: Windows 코드 페이지와 OEM 코드 페이지입니다. 잘못된 코드 페이지를 사용하면 데이터 변환 문제가 발생할 수 있습니다. 예를 들어, 애플리케이션에서 생성된 데이터를 Windows 코드페이지로 전송할 수 있지만, 해당 데이터를 전송하는 큐 매니저는 OEM 코드페이지로 생성되었기 때문에 MQPUT 호출 시 기본 변환에는 OEM CCSID가 사용됩니다. 메시지를 보내거나 받을 때 올바른 코드 페이지를 지정해야 합니다. 대안으로, 큐 관리자의 CCSID를 변경하는 것을 고려해 보십시오. 애플리케이션이 하나의 코드페이지 그룹에서만 메시지를 생성하는 경우 이 방법이 더 나은 접근 방식일 수 있습니다.

CCSID 변환 테이블

와 WindowsLinux 에서 변환표는 에 의해 제공됩니다 IBM MQ. 과 AIX 에서 IBM i, 변환 테이블은 과 운영 IBM MQ 체제 모두에서 제공됩니다. 제공되는 변환 테이블은 왕복 IBM MQ 무결성을 제공하는 경향이 있습니다. 즉, 데이터가 한 CCSID에서 다른 CCSID로 변환된 후 다시 원래 CCSID로 변환될 때 원본 데이터가 복원됩니다. 왕복 테이블은 모든 문자에 서로 다른 매핑을 할당합니다. 발신 CCSID에만 존재하고 수신 CCSID에는 존재하지 않는 문자가 있을 경우, 변환 과정이 왕복될 수 있도록 임의적이면서도 고유한 매핑이 할당됩니다. 변환 테이블을 구성하는 또 다른 방법은 송신 CCSID에는 존재하지만 수신 CCSID에는 존재하지 않는 모든 문자를 대체 문자로 매핑하는 것이다. 이러한 테이블은 왕복 무결성을 제공하지 않습니다. 데이터가 한 CCSID에서 다른 CCSID로 변환된 후 다시 원본 CCSID로 변환될 때, 원본 데이터가 복구되지 않을 수 있습니다. 운영 체제와 함께 제공되는 변환 테이블은 종종 이러한 유형입니다. 알 수 없는 문자에 대한 특정 처리가 필요한 경우, 요구 사항을 충족하는 변환 테이블과 알고리즘을 사용하여 애플리케이션 내에서 데이터 변환을 수행하는 것을 고려하십시오.

운영 체제의 CCSID 변환 테이블 위치는 운영 체제에 따라 다릅니다. 대부분의 UNIX 운영 체제에서는 일반적으로 이진 변환 테이블 파일을 특정 디렉터리에 추가함으로써 변환 테이블을 추가할 수 있습니다. 다른 운영 체제(예: IBM i)에서는 추가 변환 테이블을 추가할 수 없습니다.

IBM MQ for AIX

변환표는 AIX 에 있습니다 `/usr/lib/nls/loc/iconvTable`.

변환기는 코드셋 이름을 사용합니다 AIX. 이러한 이름은 일반적으로 형식으로 되어 있으며 `IBM-xxxx`, 여기서 `xxxx` 는 CCSID 번호입니다. 그러나 예외 사항이 존재하며, 이는 다음 표에 나열되어 있습니다:

CCSID	설명	코드셋 이름
819	라틴 서유럽	ISO8859-1
912	라틴 동유럽	ISO8859-2
915	키릴 문자	ISO8859-5
1089	아랍어	ISO8859-6
813	그리스어	ISO8859-7
916	히브리어	ISO8859-8
920	터키어	ISO8859-9
950	TraditionalChinese	big5
954 5050 33722	일본어	IBM-eucJP
970	한국어	IBM-eucKR
964	대만어	IBM-eucTW
1383	중국어	IBM-eucCN
1051	HP-UX 라틴어	IBM-1051
285	독일 EBCDIC	IBM-273

의 테이블은 `/usr/lib/nls/loc/iconvTable` SBCS 변환에만 사용됩니다. 이 테이블들은 명령어를 `AIXgenxlt` 사용하여 생성할 수 있습니다(이 명령어에 대한 자세한 내용은 사용 AIX중인 버전의 문서에서 관련 항목을 참조하십시오. 예를 들어 `genxlt` 명령). IBM MQ for AIX 또한 새로운 DBCS(더블 바이트 문자 집합) 및 일부 SBCS(싱글 바이트 문자 집합) 변환을 지원하기 위해 `uconv` 메서드를 사용합니다. `uconv` 메서드 테이블은 더 복잡합니다. 이 테이블들은 명령어를 `uconvdef` 사용하여 생성할 수 있습니다(자세한 내용은 문서(예: AIX `uconvdef` 명령)를 참조하십시오). `uconv` 메서드가 변환을 지원하는지 확인하려면 다음 검사를 수행하십시오:

- 디렉터리 `/usr/lib/nls/loc/uconvTable` (참고: `uconvTable` 가 아님 `iconvTable`)에 두 개의 파일이 있는지 확인하십시오. 하나는 출발 코드셋용이고, 다른 하나는 도착 코드셋용입니다.
- 해당 `/usr/lib/nls/loc/iconv` 디렉터리에 파일에서 `from-codeset_to-codeset` 파일으로의 심볼릭 링크가 포함되어 `/usr/lib/nls/loc/iconv/Universal_UCS_Conv` 있는지 확인하십시오.

테스트 목적으로 설치된 변환 테이블을 실행하려면 `iconv` 명령어를 사용할 수 있습니다. 다음 명령을 입력하십시오.

```
iconv -f fromcodeset -t tocodeset fromfile > tofile
```

는 `fromfilefromcodeset` 파일의 코드셋이고, `tocodeset` 는 파일이 `tofile` 속하기를 원하는 코드셋입니다.

IBM MQ for Linux

변환 테이블은 (여기서 `/opt/mqm` 는 기본 IBM MQ `/opt/mqm/lib/iconv` 설치 디렉터리)에 있습니다.

사용된 변환 코드는 다음과 유사합니다 Windows.

테이블명은 코드페이지 번호의 16진수 버전을 결합하여 명명되므로, 500에서 850으로의 변환은 테이블에 있습니다 `01F40352.tbl`. 변환 테이블 디렉터리 파일에는 제공되는 대부분의 테이블이 설명되어 있습니다 `readme.ccs`.

IBM MQ for Windows

변환 테이블은 (여기서 C:\mqm 는 기본 IBM MQ C:\mqm\conv\table 설치 디렉터리)에 있습니다.

Windows의 테이블은 코드 페이지 번호의 16진수 버전을 결합하여 명명되므로, 500에서 850으로의 변환은 테이블에 있습니다 01F40352.tbl. 변환 테이블 디렉터리에서 파일은 제공된 대부분의 readme.ccs 테이블을 설명합니다.

추적 (Windows, UNIX 및 IBM i)

추적 로그를 살펴보면, 함수 xcsCCSIDCompatible 에서 반환되는 xecX_W_INCOMPATIBLE_CCIDS 'x' 10806111'or 십진수 0276848913은 오류가 아닙니다. 이 코드는 함수에 전달된 두 CCSID가 유효하지만 객체 이름에 대해서도 데이터 변환이 필요함을 나타냅니다. 그런 다음 코드는 이 변환을 수행하는 함수를 호출해야 합니다.

변환 테이블을 찾을 수 없을 경우, 사용된 CCSID 및 코드 세트 이름이 표시됩니다. 초기화 중에 추적 기록은 기본 변환이 선택되었는지 여부와, 선택된 경우 어떤 코드 페이지가 선택되었는지 표시합니다.

기본 데이터 변환

기본 데이터 변환은 서로 다른 국가 언어로 작동하는 큐 관리자 네트워크가 존재하고 일부 CCSID 간 데이터 변환이 지원되지 않는 경우 유용할 수 있습니다.

앞서 설명한 바와 같이, 메시지를 전송하기 위해서는 일부 메시지 IBM MQ 데이터를 자동으로 변환해야 합니다. 두 CCSID가 단일 언어 또는 유사한 언어 그룹과 관련된 경우 CCSID 간 변환이 지원됩니다. CCSID가 서로 다른 국가 언어 그룹을 나타내는 경우, 변환은 일반적으로 채널이 시작되지 않고 메시지가 conversion not supported 표시되면서 실패합니다. 예를 들어, CCSID 850과 500은 모두 서유럽 IBM MQ 언어에 적용되므로 서로 변환할 수 있습니다. IBM MQ CCSID 850과 CCSID 1025(키릴 문자 집합을 사용하는 언어를 위한 EBCDIC 코드 문자 집합) 간 변환이 불가능합니다. 한 CCSID의 많은 문자가 다른 CCSID에서는 표현할 수 없기 때문입니다.

그러나 메시지 처리에 필요한 데이터, 즉 채널 간 연결 협상 시 교환되는 데이터는 매우 제한된 문자 집합으로 구성됩니다: 일반적으로 A-Z, a-z, 0-9 및 몇 가지 특수 문자로, 이는 네이밍 IBM MQ 객체에서 설명된 바와 같습니다. 이 데이터를 변환하려면, 데이터가 ASCII 또는 EBCDIC 문자 IBM MQ 인코딩 표준 중 어느 것을 사용하는지만 알면 됩니다. 그러면 단일 테이블을 사용하여 변환을 수행할 수 있습니다. 이러한 종류의 변환을 기본 데이터 변환이라고 합니다.

기본 ASCII CCSID 및 기본 EBCDIC CCSID를 정의하여 기본 데이터 변환을 활성화할 수 있습니다. 이러한 기본 값을 정의한 후, 플랫폼에 한 CCSID에서 다른 CCSID로의 변환 테이블이 없지만 두 CCSID 모두 큐 관리자에게 알려진 경우 다음과 같이 변환을 수행합니다 IBM MQ:

- CCSID가 둘 다 ASCII이거나 둘 다 EBCDIC인 경우, 데이터를 변환하지 않고 IBM MQ 수신 애플리케이션으로 전달합니다.
- 한쪽 CCSID가 ASCII이고 다른 쪽이 EBCDIC인 경우, IBM MQ 기본값을 사용하여 변환을 수행합니다. 따라서, 사용 중인 플랫폼에서 지원되는 ASCII/EBCDIC 쌍을 사용해야 합니다.

기본 데이터 변환을 활성화하려면, 플랫폼에 따라 파일 ccsid_part2.tbl 또는 파일을 ccsid.tbl 편집하여 EBCDIC 및 ASCII 기본값을 정의하는 두 줄의 주석을 해제하십시오. 권장 기본값은 500과 850이지만, 이 값들을 변경할 수 있으며, 서로 간의 변환이 지원되도록 해야 합니다. 자세한 내용은 파일 내의 지침을 참조하십시오. 이 변경 사항은 애플리케이션이 다음에 큐 관리자에 연결할 때 반영됩니다(예를 들어, 'MQI' 애플리케이션이 'MQCONN' 호출을 사용할 때).

다음 제한사항이 적용됩니다.

- 기본 데이터 변환은 애플리케이션 데이터 변환을 위한 것이 아니므로, 기대하는 변환이 이루어지지 않을 수 있습니다. 어쨌든 기본 데이터 변환을 사용하려면 이를 활성화하는 데이터 변환 출구를 제공해야 합니다. 예를 들어, 'MQI' 코드에서 MQXNCVC 호출 시 MQDCC_DEFAULT_CONVERSION 매개변수를 지정하십시오. 데이터 변환 종료 코드를 생성하는 데 사용할 수 있는 crtmqcvx 명령어에는 이 매개변수가 포함되어 있지 않다는 점에 유의하십시오.
- 기본 데이터 변환이 설정된 경우, MQFMT_STRING 메시지 형식의 메시지 변환을 포함한 내부 변환에 사용됩니다.

- EBCDIC 일본어 CCSID 290, 930, 1279 또는 5026을 사용한 기본 데이터 변환을 사용할 경우 주의하십시오. 이 CCSID에서 소문자 라틴 문자에 사용되는 코드는 다른 어떤 EBCDIC CCSID(CCSID 500 포함)와도 다릅니다. EBCDIC 기본값을 930 또는 5026으로 설정하고, ASCII 기본값은 930 또는 5026으로 변환할 수 있는 CCSID(예: 932, 94, 954 또는 5050)로 설정하는 것을 고려하십시오. 일본어 EBCDIC CCSID를 사용할 때의 또 다른 대안은 모든 객체 IBM MQ 이름이 오직 대문자 라틴 문자만을 사용하도록 하는 것입니다.

대안적인 접근 방식은 모든 큐 관리자의 CCSID를 단일 언어로 변경하는 것입니다. 응용 프로그램 데이터는 메시지 디스크립터에 응용 프로그램이 설정하는 값에 따라 여전히 서로 다른 CCSID를 사용할 수 있습니다.

관련 정보

[메시지 및 큐](#)

메시지 우선순위

메시지의 우선순위를 숫자 값으로 설정하거나 메시지에서 큐의 기본 우선순위를 사용하도록 할 수 있습니다.

메시지를 큐에 넣는 경우 메시지의 우선순위를 설정합니다(MQMD 구조의 *Priority* 필드에서). 우선순위에 대해 숫자 값을 설정하거나 메시지가 큐의 우선순위를 사용하도록 할 수 있습니다.

큐의 **MsgDeliverySequence** 속성은 큐의 메시지가 FIFO(선입선출) 순서 또는 우선순위 순서 내에 FIFO로 저장될지 판별합니다. 이 속성이 MQMDS_PRIORITY로 설정되는 경우 메시지 디스크립터의 *Priority* 필드에 지정된 우선순위대로 메시지를 큐에 넣습니다. 동일한 우선순위를 갖는 메시지는 도착하는 순서대로 큐에 저장됩니다.

큐의 **DefPriority** 속성은 해당 큐에 들어지는 메시지에 대한 기본 우선순위 값을 설정합니다. 큐가 작성될 때 이 값이 설정되지만 그 후에 변경할 수 있습니다. 리모트 큐의 로컬 정의 및 알리어스 큐에는 해석되는 기본 큐와 다른 기본 우선순위를 가질 수 있습니다. 해석 경로에 하나 이상의 큐 정의가 있는 경우(765 페이지의 [『이름 해석』](#) 참조) 기본 우선순위는 열린 명령에 지정된 큐의 **DefPriority** 속성 값(put 조작 시)을 사용합니다.

큐 관리자의 **MaxPriority** 속성 값은 큐 관리자에서 처리하는 메시지에 지정할 수 있는 최대 우선순위입니다. 이 속성의 값은 변경할 수 없습니다. IBM MQ에서 속성은 9 값을 갖습니다. 가장 낮은 0과 가장 높은 9 사이의 우선순위를 갖는 메시지를 작성할 수 있습니다.

메시지 특성

메시지 특성을 사용하여 애플리케이션이 처리할 메시지를 선택하거나 MQMD 또는 MQRFH2 헤더에 액세스하지 않고 메시지에 대한 정보를 검색할 수 있습니다. 또한 IBM MQ 애플리케이션과 JMS 애플리케이션 사이의 통신을 용이하게 합니다.

메시지 특성은 메시지와 연관된 데이터로 텍스트 형식 이름과 특정 유형의 값으로 구성됩니다. 메시지 특성은 메시지 선택자가 발행물을 주제로 필터링하거나, 큐에서 선택적으로 메시지를 가져오는 데 사용합니다. 애플리케이션 데이터에 저장하지 않고 비즈니스 데이터 또는 상태 정보를 포함시키는 데 메시지 특성을 사용할 수 있습니다. 애플리케이션은 MQ 메시지 디스크립터(MQMD) 또는 MQRFH2 헤더에 있는 데이터에 액세스할 필요가 없습니다. 메시지 큐 인터페이스(MQI) 기능 호출을 사용하여 메시지 특성처럼 이러한 데이터 구조의 필드에 액세스할 수 있기 때문입니다.

IBM MQ에서 메시지 특성의 사용은 JMS에서 특성의 사용과 비슷합니다. 이는 JMS 애플리케이션에서 특성을 설정하고 프로시저 IBM MQ 애플리케이션에서 이를 검색하거나 반대의 경우도 가능하다는 것을 의미합니다. JMS 애플리케이션에서 특성을 사용할 수 있게 하려면 "usr" 접두부를 지정하십시오. 그러면 JMS 메시지 사용자 특성처럼 (접두부 없이) 사용할 수 있습니다. 예를 들어, IBM MQ 특성 *usr.myproperty* (문자열)는 JMS 호출 `message.getStringProperty('myproperty')`을 사용하여 JMS 애플리케이션에 액세스할 수 있습니다. JMS 애플리케이션은 두 개 이상의 U+002E (".") 문자를 포함하는 경우 "usr" 접두부를 사용하여 특성에 액세스할 수 없습니다. 접두부가 없고 U+002E (".") 문자가 없는 특성은 접두부 "usr"이 있는 것처럼 처리됩니다. 반대로, JMS 애플리케이션에 설정된 사용자 특성은 "usr"을 추가하여 IBM MQ 애플리케이션에서 액세스할 수 있습니다. MQINQMP 호출에서 조회된 특성 이름에 대한 접두부입니다.

메시지 특성 및 메시지 길이

큐 관리자 속성 **MaxPropertiesLength**를 사용하여 IBM MQ 큐 관리자의 메시지와 이동할 수 있는 특성의 크기를 제어합니다.

일반적으로 MQSETMP를 사용하여 특성을 설정하는 경우 특성의 크기는 특성 이름의 길이(바이트)에 MQSETMP 호출로 전달되는 특성 값의 길이(바이트)를 더한 값입니다. 메시지를 대상으로 전송하는 동안 유니코드로 변환할

수 있기 때문에 특성 이름의 문자 세트와 특성 값을 변경할 수 있습니다. 이 경우 특성의 크기를 변경할 수 있습니다.

MQPUT 또는 MQPUT1 호출에서, 메시지의 특성은 큐 및 큐 관리자에 대한 메시지 길이에 포함되지 않지만, (메시지 특성 MQI 호출을 사용하여 특성이 설정되었는지 여부와 상관없이) 큐 관리자가 인지한 특성의 길이에 포함됩니다.

특성의 크기가 최대 특성 크기를 초과하는 경우 MQRC_PROPERTIES_TOO_BIG으로 메시지가 거부됩니다. 특성의 크기는 특성의 표현에 따라 달라지기 때문에 총 레벨에서 최대 특성 길이를 설정해야 합니다.

버퍼에 특성이 포함되는 경우 애플리케이션은 *MaxMsgLength* 값보다 큰 버퍼를 사용하여 성공적으로 메시지를 넣을 수 있습니다. 그 이유는 MQRFH2 요소로 표현되는 경우에도 메시지 특성이 메시지의 길이에 포함되지 않기 때문입니다. 하나 이상의 폴더가 포함되어 있고 헤더의 모드 폴더에 특성이 있는 경우에만 MQRFH2 헤더 필드가 특성 길이에 추가됩니다. MQRFH2 헤더에 하나 이상의 폴더가 포함되어 있고 폴더에 특성이 없는 경우에는 MQRFH2 헤더 필드는 대신 메시지 길이에 포함됩니다.

MQGET 호출에서, 메시지 특성은 큐 및 큐 관리자가 관계되지 않는 한 메시지 길이에 포함되지 않습니다. 그러나 특성이 별도로 계산되기 때문에 MQGET 호출에서 리턴하는 버퍼가 *MaxMsgLength* 속성의 값보다 큼니다.

애플리케이션에서 *MaxMsgLength* 값을 조회하지 않도록 하고 MQGET 호출 전에 이 크기의 버퍼를 할당하십시오. 대신 충분히 크다고 생각되는 만큼의 버퍼를 할당하십시오. MQGET에 실패하는 경우 *DataLength* 매개변수의 크기대로 버퍼를 할당하십시오.

MQGET 호출의 *DataLength* 매개변수는 메시지 핸들이 MQGMO 구조에 지정되지 않은 경우 사용자가 제공한 버퍼에 리턴된 모든 특성과 애플리케이션 데이터의 길이(바이트)를 리턴합니다.

MQPUT 호출의 *Buffer* 매개변수에는 송신될 애플리케이션 메시지 데이터와 메시지 데이터에 표시된 특성이 포함되어 있습니다.

메시지 디스크립터 또는 각 메시지에 대한 확장을 제외하고 메시지 특성에 대해 100MB의 길이 제한이 있습니다.

내부 표현에서 특성의 크기는 이름의 길이, 해당 값의 크기, 특성에 대한 일부 제어 데이터를 모두 합한 값입니다. 하나의 특성이 메시지에 추가되고 나면 특성 세트에 대한 일부 제어 데이터도 있습니다.

특성 이름

특성 이름은 문자열입니다. 특정 제한사항이 해당 길이 및 사용될 수 있는 문자 세트에 적용됩니다.

속성 이름은 대소문자를 구분하는 문자열로, 문맥에 의해 별도로 제한되지 않는 한 4095자로 제한됩니다. 이 한계는 MQ_MAX_PROPERTY_NAME_LENGTH 상수에 포함됩니다.

메시지 특성 MQI 호출을 사용할 때 이 최대 길이를 초과하는 경우 MQRC_PROPERTY_NAME_LENGTH_ERR의 이유 코드로 호출에 실패합니다.

에서 속성 이름의 최대 길이 JMS 제한이 없기 때문에, 애플리케이션이 JMS 유효한 JMS 속성 이름을 설정할 수 있지만, 해당 이름이 MQRFH2 구조체에 저장될 때는 유효한 IBM MQ 속성 이름이 아닐 수 있습니다.

이 경우, 구문 분석될 때 특성 이름의 처음 4095자만 사용되고 다음 문자는 잘립니다. 하나 이상의 특성이 동일한 이름으로 잘릴 수 있기 때문에 선택자를 사용하는 애플리케이션에서 선택 문자열과 일치시키거나 예상하지 않은 문자열과 일치시키는 데 실패할 수 있습니다. 속성 이름이 잘릴 경우, 오류 로그 IBM MQ 메시지를 출력합니다.

유니코드 문자 U+002E (.)가 이름의 시작이 아닌 일부로 허용되는 경우를 제외하고 모든 특성 이름은 Java ID용 Java Language Specification에서 정의한 규칙을 따라야 합니다. Java ID에 대한 규칙은 특성 이름에 대해 JMS 스펙에 포함된 규칙과 동등합니다.

공백 문자 및 비교 연산자는 금지됩니다. 임베드된 널은 특성 이름에 허용되지만 권장하지는 않습니다. 임베드된 널을 사용하는 경우 변수 길이 문자열을 지정하기 위해 MQCHARV 구조를 함께 사용하면 MQVS_NULL_TERMINATED 상수를 사용하지 못하게 됩니다.

애플리케이션은 특성 이름을 기반으로 메시지를 선택할 수 있으며 이름과 선택자 문자 세트 간의 변환이 예상치 못하게 선택에 실패하게 할 수 있기 때문에 특성 이름을 단순하게 유지하십시오.

IBM MQ 특성 이름은 특성의 논리 그룹화에 U+002E (.) 문자를 사용합니다. 이 문자는 특성에 대한 네임스페이스를 나눕니다. 다음 접두부가 포함되고 대소문자가 혼합되어 있는 특성은 제품에서 사용하도록 예약됩니다.

- mcd

- jms
- usr
- mq
- sib
- wmq
- Root
- Body
- Properties

이름 충돌을 피하기 위한 좋은 방법은 모든 애플리케이션이 해당 인터넷 도메인 이름으로 메시지 특성의 접두부를 지정하는 것입니다. 예를 들어, 도메인 이름 `ourcompany.com`을 사용하는 애플리케이션을 개발 중인 경우 접두부로 `com.ourcompany`를 사용하여 모든 특성의 이름을 지정할 수 있습니다. 또한 이 이름 지정 규칙을 통해 특성을 쉽게 선택할 수 있습니다. 예를 들어, 애플리케이션은 `com.ourcompany.%`로 시작하는 모든 메시지 특성에 대해 조회할 수 있습니다.

특성 이름 사용에 대한 자세한 정보는 [특성 이름 제한사항](#)을 참조하십시오.

특성 이름 제한사항

특성의 이름을 지정하는 경우 특정 규칙을 준수해야 합니다.

다음 제한사항이 특성 이름에 적용됩니다.

1. 특성은 다음 문자열로 시작되지 않아야 합니다.

- "JMS" - IBM MQ classes for JMS에서 사용하도록 예약됨.
- "usr.JMS" - 올바르지 않음.

유일한 예외는 JMS 특성에 대해 동의어를 제공하는 다음 특성입니다.

특성	동의어
JMSCorrelationID	Root .MQMD.CorrelId 또는 jms.Cid
JMSDeliveryMode	Root .MQMD.Persistence 또는 jms.Dlv
JMSDestination	jms.Dst
JMSExpiration	Root .MQMD.Expiry 또는 jms.Exp
JMSMessageID	Root .MQMD.MsgId
JMSPriority	Root .MQMD.Priority 또는 jms.Pri
JMSRedelivered	Root .MQMD.BackoutCount
JMSReplyTo (URI로 인코딩된 문자열)	Root .MQMD.ReplyToQ 또는 Root .MQMD.ReplyToQMGr 또는 jms.Rto
JMSTimestamp	Root .MQMD.PutDate 또는 Root .MQMD.PutTime 또는 jms.Tms
JMSType	mcd.Type 또는 mcd.Set 또는 mcd.Fmt
JMSXAppID	Root .MQMD.PutApplName
JMSXDeliveryCount	Root .MQMD.BackoutCount
JMSXGroupID	Root .MQMD.GroupId 또는 jms.Gid
JMSXGroupSeq	Root .MQMD.MsgSeqNumber 또는 jms.Seq
JMSXUserID	Root .MQMD.UserIdentifier

이러한 동의어를 사용하면 MQI 애플리케이션이 클라이언트 애플리케이션과 IBM MQ classes for JMS 유사한 방식으로 속성에 접근할 JMS 수 있습니다. 이러한 특성 중에서 JMSCorrelationID, JMSReplyTo, JMSType, JMSXGroupID 및 JMSXGroupSeq만 MQI를 사용하여 설정할 수 있습니다.

IBM MQ classes for JMS 내에서 사용 가능한 JMS_IBM_* 특성은 MQI를 통해서만 사용 가능하지 않습니다. JMS_IBM_* 특성이 참조하는 필드는 MQI 애플리케이션에서 다른 방법으로 액세스할 수 있습니다.

2. 특성은 "NULL", "TRUE", "FALSE", "NOT", "AND", "OR", "BETWEEN", "LIKE", "IN", "IS" 및 "ESCAPE"에 대해 대소문자를 혼합하여 호출할 수 없습니다. 이는 선택 문자열에 사용된 SQL 키워드의 이름들입니다.
3. "로 시작하는 특성 이름 "mq" 소문자 또는 대문자가 혼합되고 "mq_usr" 로 시작하지 않는 경우 하나의 "." 만 포함할 수 있습니다. 문자(U+002E)만 포함할 수 있습니다. 여러 개의 "." 문자는 이러한 접두부의 특성에서 허용되지 않습니다.
4. 두 개의 "." 문자는 사이에 다른 문자를 포함해야 합니다. 계층에 빈 지점을 보유할 수 없습니다. 마찬가지로 특성 이름은 "." 으로 끝날 수 없습니다. 사용할 수 없습니다.
5. 애플리케이션이 특성 "a.b"를 설정한 후 특성 "a.b.c"를 설정하는 경우 계층 "b"에 값 또는 다른 논리 그룹화가 포함되어 있는지 명확하지 않습니다. 이런 계층은 "혼합 콘텐츠"이며 지원되지 않습니다. 콘텐츠가 혼합되어 있는 특성의 설정은 허용되지 않습니다.

이러한 제한사항은 다음과 같이 유효성 검증 메커니즘에 의해 적용됩니다.

- 메시지 핸들이 작성되었을 때 유효성 검증이 요청되면, MQSETMP - 메시지 속성 설정 호출을 사용하여 특성을 설정할 때 특성 이름의 유효성이 검증됩니다. 특성의 유효성 검증을 시도하고 특성 이름의 스펙에 오류가 발생하여 이 시도에 실패하는 경우, 완료 코드는 다음 이유와 함께 MQCC_FAILED입니다.
 - 이유 1-4의 경우 MQRC_PROPERTY_NAME_ERROR입니다.
 - 이유 5의 경우 MQRC_MIXED_CONTENT_NOT_ALLOWED입니다.
- MQRFH2 요소로 직접 지정된 특성의 이름은 MQPUT 호출에 의해 유효성이 검증되지 않을 수 있습니다.

특성으로 메시지 디스크립터 필드

대부분의 메시지 디스크립터 필드는 특성으로 처리될 수 있습니다. 특성 이름은 메시지 디스크립터 필드의 이름에 접두부를 추가하여 구성됩니다.

MQI 애플리케이션에서 예를 들어, 선택자 문자열 또는 메시지 특성 API를 사용하여 메시지 디스크립터 필드에 포함된 메시지 특성을 식별하려는 경우 다음 구문을 사용하십시오.

특성 이름	메시지 디스크립터 필드
Root.MQMD.Field	필드

Field를 C 언어 선언의 MQMD 구조와 동일한 경우로 지정하십시오. 예를 들어, 특성 이름 Root.MQMD.AccountingToken은 메시지 디스크립터의 AccountingToken 필드에 액세스합니다.

메시지 디스크립터의 StrucId 및 Version 필드는 표시된 구문을 사용하여 액세스할 수 없습니다.

메시지 디스크립터 필드는 다른 특성에 대해 MQRFH2 헤더에 표시되지 않습니다.

메시지 데이터가 큐 관리자에서 인식되는 MQMDE로 시작하는 경우 설명된 Root.MQMD.Field 표기법을 사용하여 MQMDE 필드에 액세스할 수 있습니다. 이 경우 MQMDE 필드는 특성 관점에서 MQMD의 논리적 부분으로 처리됩니다. MQMDE 개요를 참조하십시오.

특성 데이터 유형 및 값

특성은 부울, 바이트 문자열, 문자열, 부동 소수점 또는 정수가 될 수 있습니다. 특성은 컨텍스트에서 제한하지 않는 한 데이터 유형 범위 내의 올바른 값을 저장할 수 있습니다.

특성 값의 데이터 유형은 다음 값 중 하나여야 합니다.

- MQBOOL
- MQBYTE[]
- MQCHAR[]
- MQFLOAT32
- MQFLOAT64

- MQINT8
- MQINT16
- MQINT32
- MQINT64

특성이 존재할 수 있지만 정의된 값이 없으므로 널 특성입니다. 널 특성은 정의되어 있지만 값이 비어 있는 즉, 0 길이 값을 가지고 있다는 점에서 바이트 특성(MQBYTE[]) 또는 문자열 특성(MQCHAR[])과 다릅니다.

바이트 문자열은 JMS 또는 XMS에서 올바른 특성 데이터 유형이 아닙니다. *usr* 폴더에서는 바이트 문자열 특성을 사용하지 않는 것이 좋습니다.

큐에서 메시지 선택

MQGET 호출의 MsgId 및 CorrelId 필드를 사용하거나 MQOPEN 또는 MQSUB 호출에서 SelectionString을 사용하여 큐에서 메시지를 선택할 수 있습니다.

선택자

메시지 선택자는 선택 문자열이 표시하는 SQL(Structured Query Language) 조회를 충족하는 특성이 있는 이러한 메시지에만 관심을 등록하기 위해 애플리케이션에서 사용하는 변수 길이 문자열입니다.

MQSUB 및 MQOPEN 함수 호출을 사용하여 선택

MQCHARV 유형의 구조인 *SelectionString*을 사용하여 MQSUB 및 MQOPEN 호출을 통해 선택합니다.

SelectionString 구조는 변수 길이 선택 문자열을 큐 관리자에 전달하는 데 사용됩니다.

선택자 문자열과 연관된 CCSID는 MQCHARV 구조의 VSCCSID 필드를 통해 설정됩니다. 사용된 값은 선택자 문자열에 지원되는 CCSID여야 합니다. 지원되는 코드 페이지 목록은 [코드 페이지 변환](#)을 참조하십시오.

IBM MQ 지원 유니코드 변환이 없는 CCSID를 지정하면 MQRC_SOURCE_CCSID_ERROR 오류가 발생합니다. 이 오류는 선택자가 큐 관리자에 표시될 때 즉, MQSUB, MQOPEN 또는 MQPUT1 호출 시 리턴됩니다.

VSCCSID 필드의 기본값은 MQCCSI_APPL이며, 이는 선택 문자열의 CCSID가 큐 관리자 CCSID 또는 클라이언트를 통해 연결된 경우 클라이언트 CCSID와 동일하다는 것을 나타냅니다. 그러나 MQCCSI_APPL 상수는 컴파일 전에 애플리케이션 재정의에 의해 대체될 수 있습니다.

MQCHARV 선택자가 NULL 문자열을 표시하는 경우 이 메시지 이용자에 대해서는 아무 선택도 발생하지 않으며 메시지는 마치 선택자가 사용되지 않은 것처럼 전달됩니다.

선택 문자열의 최대 길이는 MQCHARV 필드 *VSLength*가 나타낼 수 있는 값에 의해서만 제한됩니다.

버퍼가 제공되고 VSBufSize에 양수 버퍼 길이가 있는 경우 MQSO_RESUME 구독 옵션을 사용하는 MQSUB 호출의 출력에 *SelectionString*이 리턴됩니다. 버퍼를 제공하지 않으면 선택 문자열의 길이만 MQCHARV의 *VSLength* 필드에 리턴됩니다. 제공된 버퍼가 필드를 리턴하는 데 필요한 공간 보다 작은 경우, 제공된 버퍼에는 VSBufSize 바이트만 리턴됩니다.

애플리케이션은 먼저 큐에 대한 핸들(MQOPEN의 경우) 또는 구독(MQSUB의 경우)을 닫은 후에 선택 문자열을 대체할 수 있습니다. 그런 다음 후속 MQOPEN 또는 MQSUB 호출에서 새로운 선택 문자열을 지정할 수 있습니다.

MQOPEN

MQCLOSE를 사용하여 열려 있는 핸들을 닫으십시오. 그런 다음 후속 MQOPEN 호출에 새로운 선택 문자열을 지정하십시오.

MQSUB

MQCLOSE를 사용하여 리턴된 구독 핸들(hSub)을 닫으십시오. 그런 다음 후속 MQSUB 호출에 새 선택 문자열을 지정하십시오.

[37 페이지의 그림 3](#)에서는 MQSUB 호출을 사용하는 선택 프로세스를 보여줍니다.

MQOPEN

(APP 1)

ObjectName = "MyDestQ"
hObj



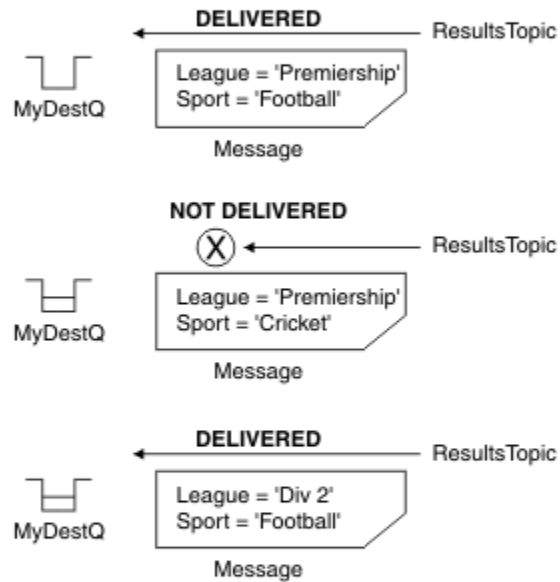
MQSUB

(APP 1)

SelectionString = "Sport = 'Football'"
hObj
TopicString = "ResultsTopic"



ResultsTopic



MQGET

(APP 1) hObj

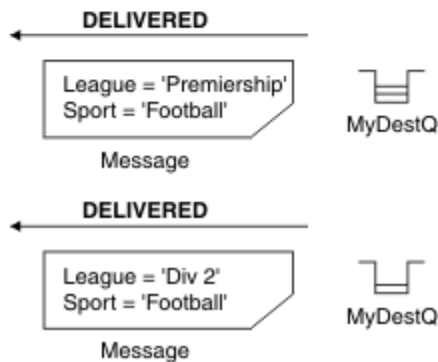


그림 3. MQSUB 호출을 사용한 선택

선택자는 MQSD 구조의 *SelectionString* 필드를 사용하여 MQSUB에 대한 호출에서 전달될 수 있습니다. MQSUB의 선택자로 전달되는 경우 구독 중인 토픽으로 발행되고 제공된 선택 문자열과 일치하는 메시지만 목적지 큐에서 사용 가능해집니다.

38 페이지의 [그림 4](#)에서는 MQOPEN 호출을 사용하는 선택 프로세스를 보여줍니다.

MQOPEN

(APP 1)

SelectorString = "League = 'Premiership'"
ObjectName = "SportQ"
hObj



MQPUT Application 2

League = 'Div 2'
Sport = 'Football'

Message

MQPUT Application 2

League = 'Premiership'
Sport = 'Football'

Message

MQGET

(APP 1) hObj

NOT DELIVERED



League = 'Div 2'
Sport = 'Football'

Message



DELIVERED

League = 'Premiership'
Sport = 'Football'

Message



MQRC_NO_MSG_AVAILABLE



그림 4. MQOPEN 호출을 사용한 선택

선택자는 MQSD 구조의 *SelectionString* 필드를 사용하여 MQOPEN에 대한 호출에서 전달될 수 있습니다. MQOPEN 호출의 선택자로 전달된 경우 열린 큐에 있고 선택자와 일치하는 메시지만 메시지 이용자에게 전달됩니다.

MQOPEN 호출의 선택자는 애플리케이션이 큐에서 선택자와 일치하는 메시지만 수신하도록 선택할 수 있는 포인트-투-포인트 경우에 기본적으로 사용됩니다. 이전 예에서는 두 개의 메시지를 MQOPEN에 의해 열린 큐에 넣지만 한 개의 메시지만 선택자와 일치하므로 이 메시지만 애플리케이션이 수신하여 가져오는 단순한 시나리오를 표시합니다.

제공된 선택자와 일치하는 추가적인 메시지가 큐에 없으므로 후속 MQGET 호출의 결과가 MQRC_NO_MSG_AVAILABLE로 나타나는 점을 참고하십시오.

관련 개념

45 페이지의 『선택 문자열 규칙 및 제한사항』

선택 문자열이 해석되는 방법에 대한 규칙 및 선택자를 사용할 때 잠재적인 문제점을 피하기 위한 문자 제한사항을 숙지하십시오.

선택 동작

IBM MQ 선택 동작에 대한 개요입니다.

MQMD가 다음과 같은 경우, MQMDE 구조의 필드가 해당하는 메시지 디스크립터 특성에 대한 메시지 특성이 된다고 간주합니다.

- MQFMT_MD_EXTENSION 형식을 포함하는 경우
- 바로 뒤에 올바른 MQMDE 구조가 오는 경우
- 버전 1이거나 기본 버전 두 개 필드만 포함하는 경우

선택 문자열이 메시지 특성에 대한 일치 발생하기 전에 TRUE 또는 FALSE로 해석될 수 있습니다. 예를 들어, 선택 문자열이 "TRUE <> FALSE"로 설정된 경우일 수 있습니다. 이러한 초기 평가는 선택 문자열에 메시지 특성 참조가 없는 경우에만 발생된다고 보장합니다.

메시지 특성이 고려되기 전에 선택 문자열이 TRUE로 해석되는 경우 이용자가 구독한 주제로 발행된 모든 메시지가 전달됩니다. 메시지 특성이 고려되기 전에 선택 문자열이 FALSE로 해석되는 경우

MQRC_SELECTOR_ALWAYS_FALSE 이유 코드 및 MQCC_FAILED 완료 코드가 선택자를 표시한 함수 호출에 리턴됩니다.

메시지에 (헤더 특성 이외에) 메시지 특성이 포함되어 있지 않은 경우라도 여전히 선택할 자격이 있을 수 있습니다. 선택 문자열이 존재하지 않는 메시지 특성을 참조하는 경우 이 특성은 NULL 또는 '알 수 없음'의 값을 가지고 있다고 가정합니다.

예를 들어, 메시지는 'Color IS NULL'과 같은 선택 문자열을 계속 만족시킬 수 있으며 여기서 'Color'는 메시지에서 메시지 특성으로 존재하지 않습니다.

확장 메시지 선택 제공자가 사용 가능하지 않는 한, 메시지 자체가 아닌 메시지와 연관된 특성에서만 선택을 수행할 수 있습니다. 확장 메시지 선택 제공자가 사용 가능한 경우에만 메시지 페이로드에서 선택을 수행할 수 있습니다.

각 메시지 특성에는 특성과 연관된 유형이 있습니다. 선택을 수행하는 경우 메시지 특성을 테스트하기 위해 표현식에 사용된 값이 올바른 유형의 값인지 확인해야 합니다. 유형 불일치가 발생하면 질문의 표현식은 FALSE로 해석됩니다.

사용자는 선택 문자열과 메시지 특성이 호환 가능한 유형을 사용해야 합니다.

원래 제공된 선택 문자열과 일치하는 메시지만 유지되도록 지속 가능한 비활성 구독자를 대신하여 선택 기준이 계속 적용됩니다.

지속 가능한 구독이 대체 (MQSO_ALTER)로 재개되는 경우 선택 문자열은 변경 가능하지 않습니다. 지속 가능한 구독자가 활동을 재개할 때 다른 선택 문자열이 표시되는 경우에는 MQRC_SELECTOR_NOT_ALTERABLE이 애플리케이션에 리턴됩니다.

애플리케이션은 선택 기준을 충족하는 큐에 메시지가 없는 경우 MQRC_NO_MSG_AVAILABLE의 리턴 코드를 수신합니다.

애플리케이션에 특성 값을 포함하는 선택 문자열이 지정된 경우에는 일치하는 특성을 포함하는 해당 메시지만 선택할 수 있습니다. 예를 들어, 구독자가 선택 문자열 "a = 3"을 지정하고 특성을 포함하지 않은 메시지 또는 'a'가 3이 아니거나 존재하지 않는 특성을 포함한 메시지가 발행됩니다. 구독자는 목적지 큐에 대한 해당 메시지를 수신하지 않습니다.

메시징 성능

큐에서 메시지를 선택하려면 IBM MQ에서 순차적으로 큐의 각 메시지를 조사해야 합니다. 선택 기준과 일치하는 메시지가 발견되거나 검사할 메시지가 더 이상 없을 때까지 메시지를 계속 조사합니다. 따라서 메시지 선택이 딥 큐에 사용되는 경우 메시징 성능이 저하됩니다.

JMSCorrelationID 또는 JMSMessageID에 기반하여 선택하는 딥 큐에서 메시지 선택을 최적화하려면 양식의 선택 문자열을 사용하십시오.

- JMSCorrelationID='ID:correlation_id'
- JMSMessageID='ID:message_id'

설명:

- `correlation_id`는 표준 IBM MQ 상관 ID를 포함하는 문자열입니다.
- `message_id`는 표준 IBM MQ 메시지 ID를 포함하는 문자열입니다.

참고: 선택자는 특성 중 하나만 참조해야 합니다. 이러한 형식 중 하나를 보유한 선택자를 사용하면 JMSCorrelationID에서 선택 시 성능이 크게 향상되며 JMSMessageID에 대해서는 약간의 성능 향상을 제공합니다. 자세한 정보는 140 페이지의 『JMS의 메시지 선택자』의 내용을 참조하십시오.

복잡한 선택자 사용

선택자는 여러 컴포넌트를 포함할 수 있습니다. 예를 들면, 다음과 같습니다.

a 및 b 또는 c 및 d 또는 e 및 f 또는 g 및 h 또는 i 및 j... 또는 y 및 z

이와 같은 복잡한 선택자를 사용하면 성능에 심각한 영향을 줄 수 있으며 과도한 자원이 요구될 수 있습니다. 이에 따라 IBM MQ는 시스템 자원 부족이 발생할 수 있는 지나치게 복잡한 선택자 처리에 실패하여 시스템을 보호하게 됩니다. 100개 이상의 테스트가 포함되어 있는 선택 문자열에서 또는 IBM MQ에서 운영 체제의 크기 한계에 도달하고 있는지 감지하는 경우 보호가 발생할 수 있습니다. 적절한 플랫폼에서 보호 한계에 도달하지 않았는지 확인하기 위해 여러 컴포넌트가 포함된 선택 문자열의 사용을 철저히 테스트해야 합니다.

컴포넌트를 결합하기 위해 추가 괄호를 사용하여 간소화함으로써 선택자의 성능과 복잡성을 향상시킬 수 있습니다. 예를 들면, 다음과 같습니다.

(a 및 b 또는 c 및 d) 또는 (e 및 f 또는 g 및 h) 또는 (i 및 j) ...

관련 개념

45 페이지의 『선택 문자열 규칙 및 제한사항』

선택 문자열이 해석되는 방법에 대한 규칙 및 선택자를 사용할 때 잠재적인 문제점을 피하기 위한 문자 제한사항을 숙지하십시오.

메시지 선택자 구문

IBM MQ 메시지 선택자는 SQL92 조건식 구문의 서브세트를 기반으로 하는 구문을 가진 문자열입니다.

메시지 선택자가 평가되는 순서는 우선순위 레벨 내의 왼쪽에서 오른쪽으로 평가됩니다. 괄호를 사용하여 이 순서를 변경할 수 있습니다. 사전 정의된 선택자 리터럴 및 연산자 이름은 여기에 대문자로 작성되지만 대소문자를 구분하지는 않습니다.

선택자가 API를 통해 제공된 경우 IBM MQ는 메시지 선택자가 표시될 때 구문의 정확성을 확인합니다. 선택 문자열의 구문이 올바르지 않거나 특성 이름이 올바르지 않고 확장 메시지 선택 제공자를 사용할 수 없는 경우, `MQRC_SELECTION_NOT_AVAILABLE`이 애플리케이션으로 리턴됩니다. 선택 문자열의 구문이 올바르지 않거나 특성 이름이 올바르지 않은 경우 구독이 재개되면 `MQRC_SELECTOR_SYNTAX_ERROR`가 애플리케이션에 리턴됩니다. 특성 이름 유효성 검증이 사용 불가능한 경우 (`MQCMHO_VALIDATE` 대신 `MQCMHO_NONE`을 설정하여) 특성이 설정되고 애플리케이션에서 그 후에 올바르지 않은 특성 이름으로 메시지를 넣으면 이 메시지는 선택되지 않습니다.

선택자가 제시될 당시, 매개변수 **SELTYPEDISPLAY SUB**의 값이 `EXTENDED`인 경우와 같이 관리적으로 정의된 구독 선택자가 확장 메시지 구문을 사용하고 있다고 판단되면 IBM MQ 오류가 반환되지 않습니다. 이 경우, 선택 문자열의 구문 검사는 공개 시간까지 지연됩니다(`MQRC_SELECTION_NOT_AVAILABLE` 참조).

선택자는 다음을 포함할 수 있습니다.

- 리터럴:
 - 문자열 리터럴은 작은따옴표로 묶여 있습니다. 두 개의 연속적인 작은따옴표는 작은따옴표를 나타냅니다. 예를 들어 'literal' 및 'literal's'입니다. Java 문자열 리터럴처럼 유니코드 문자 인코딩을 사용합니다. 문자열 리터럴을 묶는 데 큰따옴표를 사용할 수 없습니다. 바이트 시퀀스는 작은따옴표 사이에 사용될 수 있습니다.
 - 바이트 문자열은 하나 이상의 16진 문자 쌍이 큰따옴표로 묶여 있으며 0x가 접두부로 지정됩니다. 예를 들어, "0x2F1C" 또는 "0XD43A"입니다. 바이트 문자열의 길이는 1바이트 이상이어야 합니다. 선택자 바이트 문자열이 `MQTYPE_BYTE_STRING` 유형의 메시지 특성과 일치하는 경우 선두 문자 또는 후미 문자 0에 특별한 조치를 수행하지 않습니다. 바이트는 다른 문자로 처리됩니다. 엔디언도 고려되지 않습니다. 선택자와 특성 바이트 문자열 둘 다의 길이가 동일해야 하며 바이트 시퀀스가 동일해야 합니다.

다음과 일치하는 바이트 문자열 선택의 예(`myBytes = 0AFC23`으로 가정):

- "myBytes = "0x0AFC23"" = TRUE

다음 문자열 선택은 일치하지 않습니다.

- "myBytes = "0xAFC23"" = MQRC_SELECTOR_SYNTAX_ERROR (바이트 수가 2배수가 아니기 때문에)

- "myBytes = "0x0AFC2300"" = FALSE (후미 문자 0이 비교에 중요하기 때문에)

- "myBytes = "0x000AFC23"" = FALSE (선행 문자 0이 비교에 중요하기 때문에)

- "myBytes = "0x23FC0A"" = FALSE (엔디언이 고려되지 않기 때문에)

- 16진수는 0으로 시작하고, 뒤에 대문자 또는 소문자 x가 나옵니다. 리터럴의 나머지는 하나 이상의 유효한 16진수 문자를 포함합니다. 예는 0xA, 0xAF, 0X2020입니다.

- 선행 문자 0 뒤에 0-7 사이의 하나 이상의 숫자가 오면 항상 8진수의 시작으로 해석됩니다. 이와 같이 0을 접두부로 지정하는 10진수는 표시할 수 있습니다. 예를 들어, 09는 9가 올바른 8진 숫자가 아니기 때문에 구문 오류를 리턴합니다. 8진수의 예는 0177, 0713입니다.

- 정확한 숫자 상수는 57, -957 및 +62와 같이 소수점이 없는 숫자 값입니다. 정확한 숫자 상수는 뒤에 대문자 또는 소문자 L이 올 수 있으며 수가 저장되거나 해석되는 방법에 영향을 미치지 않습니다. IBM MQ는 -9,223,372,036,854,775,808에서 9,223,372,036,854,775,807 사이의 정확한 숫자를 지원합니다.

- 근사 숫자 리터럴은 7E3 또는 -57.9E2와 같은 과학적 표기법의 숫자 값이나 7., -95.7 또는 +6.2와 같은 소수점을 포함한 숫자 값입니다. IBM MQ는 -1.797693134862315E+308에서 1.797693134862315E+308 사이의 수를 지원합니다.

가수부는 선택적 부호 문자(+ 또는 -) 뒤에 와야 합니다. 가수부는 정수 또는 분수여야 합니다. 가수부의 분수 부분은 선행 숫자가 없어도 됩니다.

대문자 또는 소문자 E는 선택적 지수의 시작을 표시합니다. 지수는 10진 기수를 가지며 지수의 숫자 부분은 선택적 부호 문자가 접두부로 지정될 수 있습니다.

근사 숫자 리터럴은 F 또는 D 문자(대소문자를 구분하지 않음)로 끝날 수 있습니다. 이 구문은 단일 또는 배정밀도 숫자에 태그를 지정하는 교차 언어 방법을 지원하기 위해 사용됩니다. 이러한 문자는 선택사항이며 근사 숫자 리터럴이 저장되거나 처리되는 방법에는 영향을 주지 않습니다. 이러한 수는 항상 배정밀도를 사용하여 저장되고 처리됩니다.

- 부울 리터럴 TRUE 및 FALSE.

참고: NaN, +Infinity, -Infinity와 같은 비정형 IEEE-754 표현은 선택 문자열에서 지원되지 않습니다. 따라서 이러한 값을 표현식의 피연산자로 사용할 수 없습니다. 음수 0는 수학적 연산의 경우 양수 0과 동일하게 처리됩니다.

• ID:

ID는 올바른 ID 시작 문자로 시작해야 하며 뒤에 0 이상의 올바른 ID 부분 문자가 오는 변수 길이 문자 시퀀스입니다. ID 이름에 대한 규칙은 메시지 특성 이름의 규칙과 동일합니다. 자세한 정보는 [33 페이지의 『특성 이름』](#) 및 [34 페이지의 『특성 이름 제한사항』](#)의 내용을 참조하십시오.

참고: 확장 메시지 선택 제공자가 사용 가능한 경우에만 메시지 페이로드에서 선택을 수행할 수 있습니다.

ID는 헤더 필드 참조 또는 특성 참조입니다. 가능한 위치에서 숫자 승격이 수행되어도 메시지 선택자에서 특성 값의 유형은 특성을 설정하는 데 사용된 유형에 해당해야 합니다. 유형 불일치가 발생하는 경우 표현식의 결과는 FALSE입니다. 메시지에 존재하지 않는 특성이 참조되는 경우 해당 값은 NULL입니다.

특성이 메시지 선택자 표현식에 사용되는 경우 특성에 대한 Get 메소드에 적용되는 유형 변환은 적용되지 않습니다. 예를 들어, 문자열 값으로 특성을 설정한 다음 선택자를 사용하여 숫자 값으로 특성을 조회하는 경우 표현식은 FALSE를 리턴합니다.

JMS 필드와 특성 이름 또는 MQMD 필드 이름에 맵핑되는 특성 이름도 선택 문자열에서 올바른 ID입니다. IBM MQ는 인식된 JMS 필드와 특성 이름을 메시지 특성 값에 맵핑합니다. 자세한 정보는 [140 페이지의 『JMS의 메시지 선택자』](#)의 내용을 참조하십시오. 예를 들어, "JMSPriority >=" 선택 문자열은 현재 메시지의 jms 폴더에서 찾은 Pri 특성에서 선택합니다.

• 오버플로우/언더플로우:

십진 및 근사 숫자 값 모두의 경우 다음 조건은 정의되지 않습니다.

- 정의된 범위 밖의 수 지정
- 오버플로우 또는 언더플로우를 발생시키는 산술 연산식 지정

이러한 조건에 대해서는 검사가 수행되지 않습니다.

• 공백:

공백, 용지 넘김, 줄 바꾸기, 캐리지 리턴, 가로 탭 또는 세로 탭으로 정의됩니다. 다음 유니코드 문자는 공백으로 인식됩니다.

- \u0009 to \u000D
- \u0020
- \u001C
- \u001D
- \u001E
- \u001F
- \u1680
- \u180E
- \u2000 - \u200A
- \u2028
- \u2029
- \u202F
- \u205F
- \u3000

• 표현식:

- 선택자는 조건식입니다. true로 평가되는 선택자는 일치하며, false나 알 수 없음으로 평가되는 선택자는 일치하지 않습니다.
- 산술 연산식은 연산식 자체, 산술 연산, ID(ID 값은 숫자 리터럴로 처리됨) 및 숫자 리터럴로 구성됩니다.
- 조건식은 조건식, 비교 연산 및 논리 연산으로 구성됩니다.

• 표현식을 평가하는 순서를 설정하는 표준 대괄호()가 지원됩니다.

• 선행 순서의 논리 연산자: NOT, AND, OR.

• 비로 연산자: =, >, >=, <, <=, <>(같지 않음).

- 문자열의 길이가 같고 바이트 시퀀스가 동일한 경우에만 2바이트 문자열이 동일합니다.
- 동일한 유형의 값만 비교할 수 있습니다. 한 가지 예외는 정확한 숫자 값과 근사 숫자 값의 비교가 올바른 경우입니다(필수 유형 변환은 Java 숫자 승격의 규칙에서 정의함). 다른 유형을 비교하려고 시도할 경우 선택자는 항상 false입니다.
- 문자열 및 부울 비교는 = 및 <> 연산자로 제한됩니다. 두 문자열은 동일한 순서의 문자를 포함하는 경우에만 동일합니다.

• 우선순위 순서의 산술 연산자:

- +, - 단항.
- * 곱하기 및 / 나누기.
- + 더하기 및 - 빼기.
- NULL 값의 산술 연산은 지원되지 않습니다. 이 연산을 시도하는 경우 전체 선택자는 항상 false입니다.
- 산술 연산에서는 Java 숫자 승격을 사용해야 합니다.

• arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 및 arithmetic-expr3 비교 연산자:

- Age BETWEEN 15 and 19는 age >= 15 AND age <= 19와 같습니다.
- Age NOT BETWEEN 15 and 19는 age < 15 OR age > 19와 같습니다.
- BETWEEN 연산의 표현식 중에 NULL이 있는 경우 연산의 값은 false입니다. NOT BETWEEN 연산의 표현식 중에 NULL이 있는 경우 연산 값의 true입니다.
- identifier [NOT] IN (string-literal1, string-literal2,...) 비교 연산자입니다. 여기서 identifier에는 String 또는 NULL 값이 있습니다.
 - Country IN ('UK', 'US', 'France')는 'UK'에 대해 true이고 'Peru'는 false입니다. (Country = 'UK') OR (Country = 'US') OR (Country = 'France') 표현식과 동일합니다.
 - Country NOT IN ('UK', 'US', 'France')는 'UK'에 대해 false이고 'Peru'는 true입니다. NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France')) 표현식과 동일합니다.
 - IN 또는 NOT IN 연산의 ID가 NULL인 경우 연산의 값은 알 수 없습니다.
- identifier [NOT] LIKE *pattern-value* [ESCAPE *escape-character*] 비교 연산자, 여기서 identifier에 문자열 값이 있습니다. *pattern-value*는 문자열 리터럴이며 여기서 _는 단일 문자를 의미하며 %는 문자의 시퀀스(빈 시퀀스 포함)를 나타냅니다. 기타 모든 문자는 해당 문자 자체를 나타냅니다. 선택적 *escape-character*는 *pattern-value*에서 _ 및 %의 특수 의미를 이스케이프하는 데 사용되는 단일 문자 문자열 리터럴입니다. LIKE 연산자는 두 개 문자열 값을 비교할 때에만 사용해야 합니다.
 - phone LIKE '12%3'은 123 및 12993에 대해 true이고 1234에 대해서는 false입니다.
 - word LIKE 'l_se'는 lose에 대해 true이고 loose의 경우는 false입니다.
 - underscored LIKE '_%' ESCAPE '\ '는 _foo에 대해 true이고 bar는 false입니다.
 - phone NOT LIKE '12%3'은 123 및 12993에 대해 false이며 1234의 경우 true입니다.
 - LIKE 또는 NOT LIKE 연산의 ID가 NULL인 경우 연산의 값은 알 수 없음입니다.

참고: LIKE 연산자는 두 개 문자열 값을 비교하는 데 사용해야 합니다. Root.MQMD.CorrelId의 값은 문자열이 아닌 24바이트 바이트 배열입니다. 선택자 문자열 Root.MQMD.CorrelId LIKE 'ABC%'는 구문 분석기에서 올바른 구문으로 허용하지만 false로 평가됩니다. 바이트 배열과 문자열을 비교하는 경우 따라서 LIKE는 사용할 수 없습니다.

- identifier IS NULL 비교 연산자는 NULL 헤더 필드 값 또는 누락된 특성 값에 대해 테스트합니다.
- identifier IS NOT NULL 비교 연산자는 널이 아닌 헤더 필드 값 또는 특성 값이 있는지 테스트합니다.
- 널값

NULL 값을 포함하는 선택자 표현식의 평가는 SQL 92 NULL 시맨틱에서 정의합니다. 요약하자면 다음과 같습니다.

- SQL은 NULL 값을 알 수 없음으로 처리합니다.
- 값이 알 수 없음인 비교 또는 산술의 결과 값은 항상 알 수 없음입니다.
- IS NULL 및 IS NOT NULL 연산자는 알 수 없는 값을 TRUE 및 FALSE 값으로 변환합니다.

부울 연산자는 3가 논리(T=TRUE, F=FALSE, U=UNKNOWN)를 사용합니다.

표 1. 논리가 A AND B인 부울 연산자 결과의 값		
연산자 A	연산자 B	결과(A AND B)
T	F	F
T	U	U
T	T	T
F	T	F

표 1. 논리가 A AND B인 부울 연산자 결과의 값 (계속)		
연산자 A	연산자 B	결과(A AND B)
F	U	F
F	F	F
U	T	U
U	U	U
U	F	F

표 2. 논리가 A OR B인 부울 연산자 결과의 값		
연산자 A	연산자 B	결과(A OR B)
T	F	T
T	U	T
T	T	T
F	T	T
F	U	U
F	F	F
U	T	T
U	U	U
U	F	U

표 3. 논리가 NOT A인 부울 연산자 결과의 값	
연산자 A	결과(NOT A)
T	F
F	T
U	U

다음 메시지 선택자는 메시지 유형이 자동차이고 색상이 파란색이며 무게가 2500lbs를 초과하는 메시지를 선택합니다.

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

SQL에서 고정 소수점 비교와 산술을 지원하지만 메시지 선택자는 지원하지 않습니다. 정확한 숫자 리터럴이 소수점이 없는 리터럴로 제한되기 때문입니다. 또한 근사 숫자 값의 대체 표시로 10진수가 있는 숫자를 사용하는 이유이기도 합니다.

SQL 주석은 지원되지 않습니다.

관련 개념

32 페이지의 『메시지 특성』

메시지 특성을 사용하여 애플리케이션이 처리할 메시지를 선택하거나 MQMD 또는 MQRFH2 헤더에 액세스하지 않고 메시지에 대한 정보를 검색할 수 있습니다. 또한 IBM MQ 애플리케이션과 JMS 애플리케이션 사이의 통신을 용이하게 합니다.

관련 참조

[MsgHandle](#)

[MQBUFMMH - 버퍼를 메시지 핸들로 변환](#)

선택 문자열 규칙 및 제한사항

선택 문자열이 해석되는 방법에 대한 규칙 및 선택자를 사용할 때 잠재적인 문제점을 피하기 위한 문자 제한사항을 숙지하십시오.

- 발행/구독 메시지를 위한 메시지 선택은 발행자에 의해 송신될 때 메시지에서 발생합니다. 선택 문자열을 참조하십시오.
- 단일 등호 문자를 사용하여 등가가 테스트됩니다. 예를 들어, `a = b`가 올바른 반면 `a == b`는 올바르지 않습니다.
- '같지 않음(not equal to)'을 표현하기 위해 여러 프로그래밍 언어에서 사용하는 연산자는 `!=`입니다. 이 표현은 `<>`에 대해 올바른 동의어가 아닙니다. 예를 들어, `a <> b`는 올바르지만 `a != b`는 올바르지 않습니다.
- ' (U+0027) 문자가 사용되는 경우에만 작은따옴표가 인식됩니다. 마찬가지로 바이트 문자열을 묶는 데 사용될 때에만 올바른 큰따옴표는 " (U+0022) 문자를 사용해야 합니다.
- `&`, `&&`, `|` 및 `||` 기호는 논리적 결합/분리의 동의어가 아닙니다. 예를 들어, `a && b`를 `a AND b`로 지정해야 합니다.
- 와일드카드 문자 `*` 및 `?`는 `%` 및 `_`에 대한 동의어가 아닙니다.
- `20 < b < 30`과 같은 복합 표현식을 포함하는 선택자는 유효하지 않습니다. 구문 분석기는 왼쪽에서 오른쪽의 동일한 선행 레벨을 갖는 연산자를 평가합니다. 따라서 이 예는 `(20 < b) < 30`이 되며, 이는 아무런 의미가 없습니다. 대신에 표현식을 `(b > 20) AND (b < 30)`으로 써야 합니다.
- 바이트 문자열은 큰따옴표로 묶어야 합니다. 작은따옴표가 사용되는 경우 바이트 문자열은 문자열 리터럴이 됩니다. `0x` 뒤에 오는 문자 수(문자가 표시하는 숫자가 아님)는 2배수여야 합니다.
- 키워드는 등호 문자의 IS 동의어가 아닙니다. 따라서 선택 문자열 `a IS 3` 및 `b IS 'red'`는 올바르지 않습니다. IS 키워드는 IS NULL 및 IS NOT NULL 경우를 지원하기 위해서만 존재합니다.

관련 개념

39 페이지의 『선택 동작』

IBM MQ 선택 동작에 대한 개요입니다.

관련 참조

선택 문자열

메시지 선택자를 사용하는 경우 UTF-8 및 유니코드 고려사항

작은따옴표로 묶이지 않은 선택 문자열의 예약 키워드를 구성하는 문자는 Basic Latin 유니코드(U+0000에서 U+0007F 사이)로 입력되어야 합니다. 영숫자 문자의 다른 코드 포인트 표현을 사용하는 것은 올바르지 않습니다. 예를 들어, 숫자 1은 유니코드 U+0031로 표현되어야 하며 Fullwidth Digit 등가 U+FF11 또는 아라비아 숫자 등가 U+0661을 사용하는 것은 올바르지 않습니다.

메시지 특성 이름은 올바른 유니코드 문자 시퀀스를 사용하여 지정할 수 있습니다. UTF-8로 인코딩된 선택 문자열 내의 메시지 특성 이름은 멀티바이트 문자를 포함하더라도 유효성이 검증됩니다. 멀티바이트 UTF-8의 유효성 검증은 엄격하며 메시지 특성 이름에 올바른 UTF-8 시퀀스가 사용되었는지 확인해야 합니다. UTF-16에서 대리 코드 포인트(X'D800'에서 X'DFFF'까지)로 표시되는 유니코드 BMP(Basic Multilingual Plane)(U+FFFF 초과) 또는 UTF-8의 4바이트는 메시지 특성 이름에서 지원되지 않습니다.

동등한지 비교하는 경우 특성 이름 또는 값에 대해 추가적인 처리는 수행되지 않습니다. 이는 사전 구성/분해가 발생하지 않으며 연결 문자는 특별한 의미를 제공하지 않음을 의미합니다. 예를 들어, 사전 구성된 움라우트 문자 U+00FC는 U+0075 + U+0308와 동일한 것으로 간주되지 않으며 문자 시퀀스 `ff`는 유니코드 U+FB00(LATIN SMALL LIGATURE FF)와 동일한 것으로 간주되지 않습니다.

작은따옴표로 묶인 특성 데이터는 바이트 시퀀스로 나타낼 수 있으며 유효성이 검증되지 않습니다.

메시지의 콘텐츠 선택

메시지 페이로드 콘텐츠(콘텐츠 필터링으로 알려진)의 선택에 따라 구독할 수 있지만, 구독과 같이 어떤 메시지가 전달되어야 하는지에 대한 의사결정은 IBM MQ에서 직접 수행할 수 없습니다. 대신 확장 메시지 선택 제공자(예: IBM App Connect Enterprise)는 메시지를 처리해야 합니다.

애플리케이션에서 하나 이상의 구독자가 메시지의 콘텐츠에 대해 선택하는 선택 문자열을 가지고 있는 주제 문자열에 대해 발행하는 경우 IBM MQ는 확장 메시지 선택 제공자가 발행물의 구문을 분석하고, 발행물이 콘텐츠 필터를 사용하여 각 구독자가 지정한 선택 기준과 일치하는지 IBM MQ에 알려주도록 요청합니다.

확장 메시지 선택 제공자가 발행물이 구독자의 선택 문자열과 일치한다고 판별하는 경우 구독자에게 계속 메시지가 전달됩니다.

확장 메시지 선택 제공자가 발행물이 일치하지 않는다고 판별하는 경우 메시지는 구독자에게 전달되지 않습니다. 이는 MQPUT 또는 MQPUT1 호출이 MQRC_PUBLICATION_FAILURE 이유 코드와 함께 실패하는 원인이 될 수 있습니다. 확장 메시지 선택 제공자가 발행물의 구문을 분석할 수 없는 경우 이유 코드 MQRC_CONTENT_ERROR가 리턴되고 MQPUT 또는 MQPUT1 호출에 실패합니다.

확장 메시지 선택 제공자를 사용할 수 없거나 구독자가 발행물을 수신해야 하는지 판별할 수 없는 경우 이유 코드 MQRC_SELECTION_NOT_AVAILABLE이 리턴되고 MQPUT 또는 MQPUT1 호출에 실패합니다.

구독이 콘텐츠 필터를 사용하여 작성 중이며 확장 메시지 선택 제공자가 사용 가능하지 않은 경우 MQSUB 호출은 MQRC_SELECTION_NOT_AVAILABLE 이유 코드와 함께 실패합니다. 콘텐츠 필터를 사용하는 구독을 재개 중이며 확장 메시지 선택 제공자가 사용 가능하지 않은 경우 MQSUB 호출은 MQRC_SELECTION_NOT_AVAILABLE의 경고를 리턴하지만 구독은 재개되도록 허용됩니다.

관련 참조

[선택 문자열](#)

IBM MQ 메시지의 비동기 이용

비동기 이용은 MQI(Message Queue Interface) 확장 세트를 사용하며 MQI는 애플리케이션이 MQI 큐 세트의 메시지를 이용하기 위해 작성될 수 있는 MQCB 및 MQCTL을 호출합니다. 메시지는 '코드 단위'를 호출하여 애플리케이션에 전달되며 메시지를 전달하는 애플리케이션 또는 메시지를 표시하는 토큰에 의해 식별됩니다.

대부분의 애플리케이션 직접 환경에서 코드 단위는 함수 포인터로 정의되지만, 다른 환경에서 코드 단위는 프로그램 또는 모듈 이름으로 정의할 수 있습니다.

메시지의 비동기 이용에서 다음 용어가 사용됩니다.

메시지 이용자

애플리케이션 요구사항과 일치하는 프로그램 또는 함수가 사용 가능하게 될 때 메시지와 함께 호출되도록 프로그램, 또는 함수를 정의할 수 있는 프로그래밍 구성입니다.

이벤트 핸들러

큐 관리자 정지와 같은 비동기 이벤트가 발생하는 경우 호출하도록 프로그램 또는 함수를 정의할 수 있는 프로그래밍 구성입니다.

콜백

메시지 이용자 또는 이벤트 핸들러 루틴 중 하나를 참조하는 데 사용되는 일반 용어입니다.

비동기 이용은 새 애플리케이션 특히, 여러 입력 큐 또는 구독을 처리하는 애플리케이션의 경우 설계와 구현을 간소화할 수 있습니다. 그러나 하나 이상의 입력 큐를 사용하고 우선순위 순서대로 메시지를 처리 중인 경우 우선순위 시퀀스는 각 큐 내에서 독립적으로 준수됩니다. 한 큐에서 낮은 우선순위의 메시지를 다른 큐의 우선순위가 높은 메시지보다 먼저 받을 수 있습니다. 여러 큐에서 메시지 순서는 보장되지 않습니다. 또한 API 엑시트를 사용하는 경우 MQCB 및 MQCTL 호출을 포함하도록 엑시트를 변경해야 할 수 있습니다.

다음 그림은 이 기능을 어떻게 사용할 수 있는지에 대한 예를 제공합니다.

47 페이지의 그림 5에서는 두 개의 큐에서 메시지를 이용하는 멀티스레드 애플리케이션을 보여줍니다. 예에서는 단일 함수에 전달되는 모든 메시지를 표시합니다.

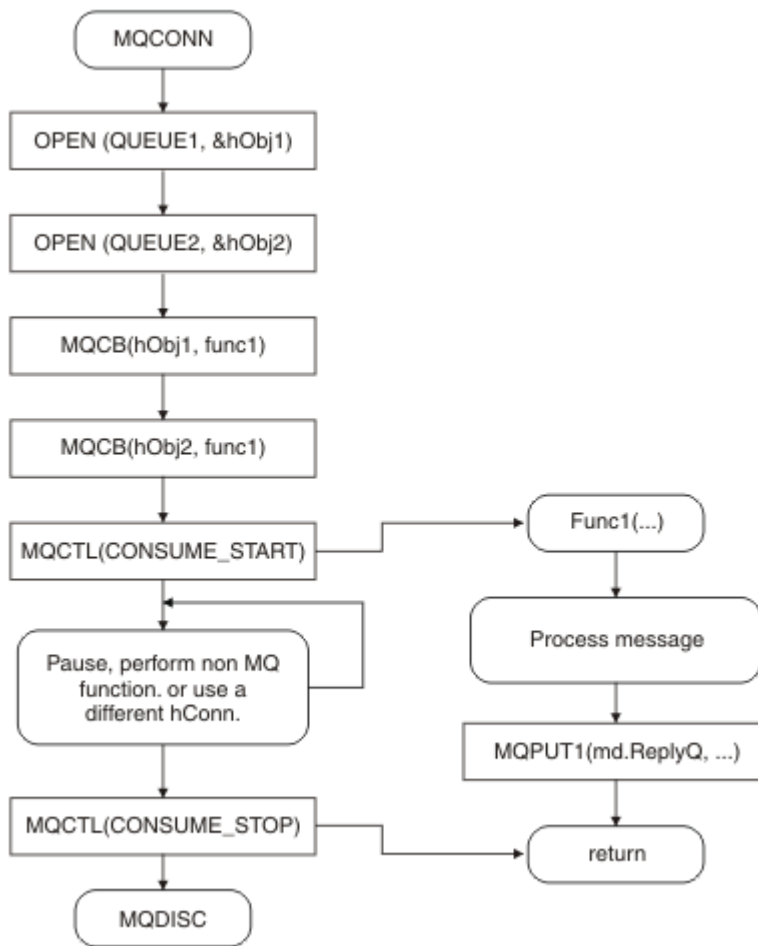


그림 5. 두 개 큐에서 이용하는 표준 메시지 구동 애플리케이션

z/OS z/OS에서 기본 제어 스레드는 종료 전에 MQDISC 호출을 발행해야 합니다. 이를 통해 콜백 스레드가 시스템 자원을 종료하거나 해제할 수 있습니다.

48 페이지의 그림 6 이 샘플 플로우는 두 개 큐에서 메시지를 이용하는 단일 스레드 애플리케이션을 표시합니다. 예에서는 단일 함수에 전달되는 모든 메시지를 표시합니다.

비동기 케이스와의 차이점은 모든 이용자가 스스로 비활성화할 때까지 MQCTL의 발행자에게 제어가 리턴되지 않는다는 점입니다. 즉, 한 이용자가 MQCTL STOP 요청 또는 큐 관리자 정지를 발행합니다.

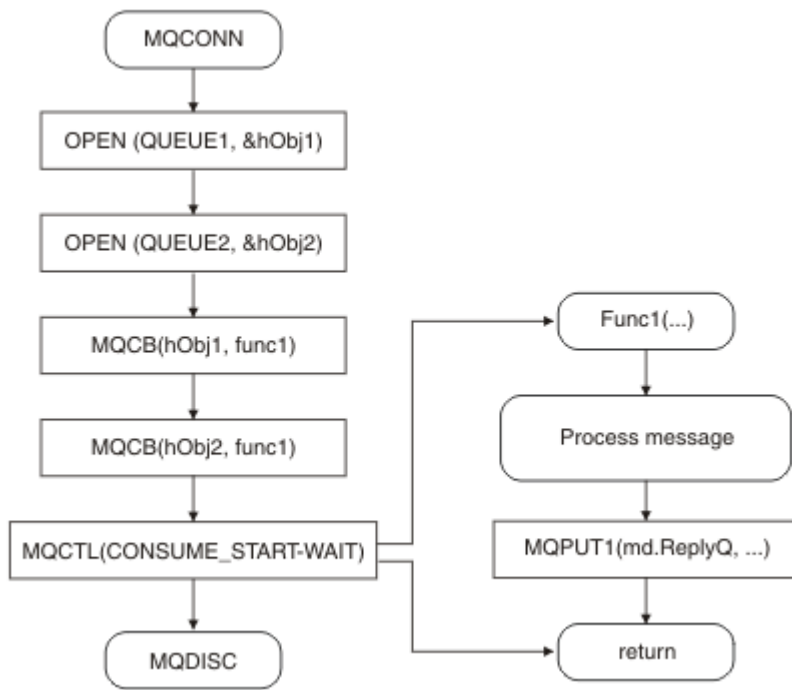


그림 6. 두 개 큐에서 이용하는 단일 스레드 메시지 구동 애플리케이션

메시지 그룹

메시지는 그룹 내에서 메시지 순서 지정을 허용할 수 있습니다.

메시지 그룹을 통해 여러 메시지를 서로 관련된 것으로 표시할 수 있으며 논리 순서를 그룹에 적용할 수 있습니다 (791 페이지의 『논리적 및 물리적 정렬』 참조). 다중 플랫폼에서는 메시지 세그먼트화를 사용하여 대형 메시지를 더 작은 세그먼트로 나눌 수 있습니다. 토픽에 넣을 때는 그룹화되거나 세그먼트화된 메시지를 사용할 수 없습니다.

그룹 내의 계층은 다음과 같습니다.

그룹

계층에서 최상위 레벨이며 *GroupId*로 식별됩니다. 동일한 *GroupId*를 포함하는 하나 이상의 메시지로 구성됩니다. 이러한 메시지는 큐 어디에나 저장할 수 있습니다.

참고: 여기에서는 메시지가 큐에 있는 하나의 항목을 표시하는 데 사용되며 MQGMO_COMPLETE_MSG를 지정하지 않는 단일 MQGET에 의해 리턴됩니다.

48 페이지의 그림 7에서는 논리 메시지 그룹을 표시합니다.

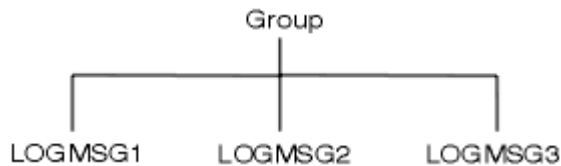


그림 7. 논리 메시지 그룹

큐를 열고 MQOO_BIND_ON_GROUP을 지정하면, 이 큐로 보내지는 한 그룹의 모든 메시지가 큐의 동일한 인스턴스로 보내집니다. BIND_ON_GROUP 옵션에 대한 자세한 정보는 [메시지 연관관계 핸들링](#)을 참조하십시오.

논리 메시지

그룹 내의 논리 메시지는 *GroupId* 및 *MsgSeqNumber* 필드에 의해 식별됩니다. *MsgSeqNumber*는 그룹 내의 첫 번째 메시지에 대해 1로 시작하고, 메시지가 그룹에 있지 않은 경우 해당 필드의 값은 1입니다.

그룹 내의 논리 메시지를 다음에 사용하십시오.

- 순서 지정을 확인합니다(메시지가 전송되는 환경에서 보장되지 않는 경우).
- 애플리케이션이 유사한 메시지를 그룹화합니다(예를 들어, 동일한 서버 인스턴스에서 모두 처리되어야 하는 메시지).

그룹 내의 각 메시지는 세그먼트로 분할되지 않는 한 하나의 물리적 메시지로 구성됩니다. 각 메시지는 논리적으로 별도의 메시지이며 MQMD의 *GroupId* 및 *MsgSeqNumber* 필드만 그룹의 다른 메시지와의 관계를 제공해야 합니다. MQMD의 다른 필드는 독립적입니다. 일부 필드는 그룹의 모든 메시지에 대해 동일할 수 있는 반면, 다른 일부는 다를 수 있습니다. 예를 들어, 그룹의 메시지는 다른 형식 이름, CCSID 및 인코딩을 가질 수 있습니다.

세그먼트

세그먼트는 넣기 또는 가져오기 애플리케이션 또는 큐 관리자(메시지가 통과하는 중간 큐 관리자 포함)에 대해 너무 큰 메시지를 핸들링하는 데 사용됩니다. 자세한 정보는 807 페이지의 『메시지 세그먼트화』의 내용을 참조하십시오.

개별 메시지는 세그먼트라는 더 작은 메시지로 나누어집니다. 메시지의 세그먼트는 *GroupId*, *MsgSeqNumber* 및 *Offset* 필드로 식별됩니다. *Offset* 필드는 메시지 내의 첫 번째 세그먼트에 대해 0으로 시작합니다.

각 세그먼트는 그룹에 속할 수 있는 하나의 물리적 메시지로 구성됩니다(49 페이지의 그림 8에서는 그룹 내의 메시지 예를 표시함). 세그먼트는 단일 메시지의 논리적 부분이므로 MQMD의 *MsgId*, *Offset* 및 *MsgFlags* 필드는 동일한 메시지의 개별 세그먼트 간에 달라야 합니다. 세그먼트가 도착하지 못하면 이유 코드 *MQRC_INCOMPLETE_GROUP* 또는 *MQRC_INCOMPLETE_MSG*가 적절하게 리턴됩니다.

49 페이지의 그림 8에서는 논리 메시지의 그룹을 표시하며 일부는 다음과 같이 세그먼트되었습니다.

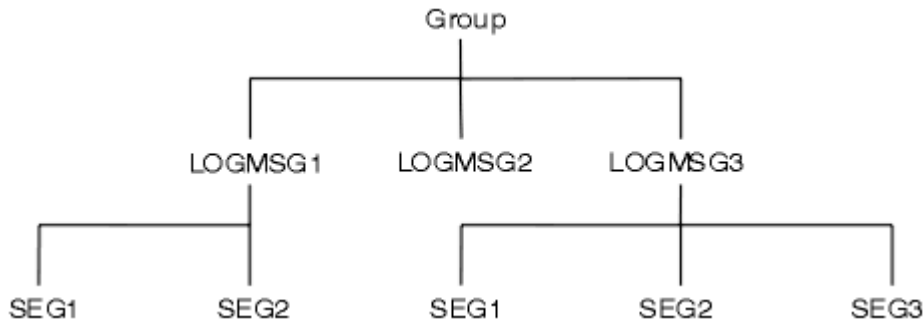


그림 8. 세그먼트된 메시지

z/OS 세그먼트화는 IBM MQ for z/OS에서 지원되지 않습니다.

발행/구독으로 세그먼트되거나 그룹화된 메시지를 사용할 수 없습니다.

관련 개념

807 페이지의 『메시지 세그먼트화』

특정 상황에서는 대용량 메시지(최대 100MB)를 여러 개의 작은 조각으로 분할하여 전송할 경우 성공률이 높아질 수 있습니다. 메시지는 애플리케이션 또는 큐 관리자에 의해 세분화될 수 있습니다. 이 기능을 IBM MQ classes for JMS 사용하는 애플리케이션이나 IBM MQmessaging REST API 에서 ~에 IBM MQ for z/OS, 또는 지원되지 않습니다.

관련 참조

791 페이지의 『논리적 및 물리적 정렬』

각 우선순위 수준 내에서, 큐에 있는 메시지는 물리적 순서 또는 논리적 순서로 발생할 수 있습니다.

MQMD - 메시지 디스크립터

메시지 지속성

지속 메시지가 로그 및 큐 데이터 파일에 작성됩니다. 큐 관리자가 실패 후에 재시작하는 경우 로그된 데이터에서 필요에 따라 이러한 지속 메시지를 복구합니다. 지속이 아닌 메시지는 큐 관리자가 중지될 때 제거되며 이러한 중지는 연산자 명령의 결과일 수도 있고 시스템 일부의 장애 때문일 수 있습니다.

▶ **z/OS** z/OS의 커플링 기능(CF)에 저장되는 비지속 메시지는 예외입니다. CF가 사용 가능하면 계속 지속됩니다.

메시지를 작성할 때 기본값을 사용하여 메시지 디스크립터(MQMD)를 초기화하는 경우 메시지에 대한 지속성은 MQOPEN 명령에 지정된 큐의 **DefPersistence** 속성에서 가져온 것입니다. 또는 메시지를 지속 또는 비지속으로 정의하는 데 MQMD 구조의 *Persistence* 필드를 사용하여 메시지의 지속성을 설정할 수 있습니다.

애플리케이션의 성능은 지속 메시지를 사용하는 경우 영향을 받으며 영향의 범위는 시스템의 I/O 서브시스템에 대한 성능 특성 및 각 플랫폼에서 동기점 옵션을 사용하는 방법에 따라 달라집니다.

- 현재 작업 단위를 벗어나는 지속 메시지는 모든 Put 및 Get 조작 시에 디스크에 작성됩니다. [863 페이지의 『작업 단위 커밋 및 백아웃』](#)의 내용을 참조하십시오.

- ▶ **z/OS** **ALW** IBM i를 제외하는 모든 플랫폼에서 현재 작업 단위 내의 지속 메시지는 작업 단위가 커밋되고 작업 단위에 많은 큐 조작이 포함될 수 있는 경우에만 로그됩니다.

비지속 메시지를 빠른 메시징에 사용할 수 있습니다. 빠른 메시징에 대한 자세한 정보는 [메시지의 안전](#)을 참조하십시오.

참고: 작업 단위 내의 지속 메시지 작성과 작업 단위 외부의 지속 메시지 작성을 결합하면 잠재적으로 애플리케이션에 심각한 성능 문제가 발생할 수 있습니다. 이는 특히 두 조작에 동일한 대상 큐를 사용하는 경우에 적용됩니다.

전달되지 못하는 메시지

큐 관리자가 큐에 메시지를 넣을 수 없는 경우 다양한 옵션이 있습니다.

다음은 수행할 수 있습니다.

- 큐에 메시지를 넣으려고 다시 시도합니다.
- 메시지가 송신자에 리턴되도록 요청합니다.
- 데드-레터 큐에 메시지를 넣습니다.

자세한 정보는 [1133 페이지의 『절차에 따른 프로그램 오류 핸들링』](#)의 내용을 참조하십시오.

백아웃된 메시지

작업 단위의 제어 아래 큐에서 메시지를 처리하는 경우 작업 단위는 하나 이상의 메시지로 구성됩니다. 백아웃이 발생하는 경우 큐에서 검색된 메시지가 큐에 복원되고 다른 작업 단위에서 다시 처리될 수 있습니다. 특정 메시지의 처리에서 문제점이 발생하면 작업 단위가 다시 백아웃됩니다. 이것은 처리 루프를 일으킬 수 있습니다. 큐에 넣어졌던 메시지는 큐에서 제거됩니다.

애플리케이션은 MQMD의 *BackoutCount* 필드를 테스트하여 이러한 루프에 갇혀 있는 메시지를 감지할 수 있습니다. 애플리케이션은 상황을 수정하거나 운영자에게 경고를 발행할 수 있습니다.

▶ **Multi** 백아웃 수는 항상 큐 관리자를 재시작한 이후에도 유지됩니다. **HardenGetBackout** 속성에 대한 모든 변경은 무시됩니다.

▶ **z/OS** 공유 큐의 경우 백아웃 수는 항상 큐 관리자를 재시작한 이후에도 유지됩니다. z/OS에서의 기타 모든 구성의 경우 개인 큐에 대한 백아웃 수가 큐 관리자 재시작 후에도 유지되도록 하려면, **HardenGetBackout** 속성을 MQQA_BACKOUT_HARDENED로 설정하십시오. 그렇지 않으면 큐 관리자를 다시 시작해야 하는 경우 각 메시지에 대한 정확한 백아웃 수가 유지되지 않습니다. 이러한 방식으로 속성을 설정하면 추가 처리의 비용이 추가됩니다.

메시지 커밋 및 백아웃에 대한 자세한 정보는 [863 페이지의 『작업 단위 커밋 및 백아웃』](#)의 내용을 참조하십시오.

응답 대상 큐 및 큐 관리자

보낸 메시지에 대한 응답으로 다음과 같은 메시지를 수신하는 경우가 있습니다.

- 요청 메시지에 대한 응답으로 오는 응답 메시지

- 예상치 못한 이벤트 또는 만기에 대한 보고 메시지
- 도착 확인(COA) 또는 전달 확인(COD) 이벤트에 대한 보고 메시지
- 긍정적인 조치 알림(PAN) 또는 부정적인 조치 알림(NAN) 이벤트에 대한 보고 메시지

MQMD 구조를 사용하여 *ReplyToQ* 필드에 응답 및 보고 메시지를 보내려는 큐의 이름을 지정하십시오.
ReplyToQMGr 필드에 응답 큐를 소유하는 큐 관리자 이름을 지정하십시오.

ReplyToQMGr 필드를 공백으로 두면 큐 관리자가 큐의 메시지 디스크립터에서 다음 필드의 콘텐츠를 설정합니다.

ReplyToQ

*ReplyToQ*가 리모트 큐의 로컬 정의인 경우 *ReplyToQ* 필드가 리모트 큐의 이름으로 설정됩니다. 그렇지 않으면 이 필드는 변경되지 않습니다.

ReplyToQMGr

*ReplyToQ*가 리모트 큐의 로컬 정의인 경우 *ReplyToQMGr* 필드가 리모트 큐를 소유하는 큐 관리자의 이름으로 설정됩니다. 그렇지 않으면 *ReplyToQMGr* 필드가 애플리케이션이 연결된 큐 관리자의 이름으로 설정됩니다.

참고: 큐 관리자가 메시지 전달을 두 번 이상 시도하도록 요청할 수 있으며, 실패하는 경우 메시지를 제거하도록 요청할 수 있습니다. 전달에 실패하는 경우 메시지가 제거되지 않는다면 리모트 큐 관리자가 해당 데드 레터(전달되지 않은 메시지) 큐에 메시지를 넣습니다([1136 페이지의 『데드 레터\(미전달 메시지\) 큐 사용』](#) 참조).

메시지 컨텍스트

메시지 컨텍스트 정보를 통해 메시지를 검색하는 애플리케이션에서 메시지의 진원지를 찾아낼 수 있습니다.

검색 애플리케이션에서 다음을 수행하려고 할 수 있습니다.

- 송신 애플리케이션이 올바른 권한 레벨을 가지고 있는지 확인합니다.
- 수행해야 하는 작업에 대해 송신 애플리케이션에 청구할 수 있도록 일부 회계 기능을 수행합니다.
- 작업한 모든 메시지의 감사 추적을 보관합니다.

MQPUT 또는 MQPUT1 호출을 사용하여 메시지를 큐에 넣을 때 큐 관리자가 일부 기본 컨텍스트 정보를 메시지 디스크립터에 추가하도록 지정할 수 있습니다. 적절한 권한 레벨이 있는 애플리케이션이 추가 컨텍스트 정보를 추가할 수 있습니다. 컨텍스트 정보를 지정하는 방법에 대한 자세한 정보는 [778 페이지의 『메시지 컨텍스트 정보 제어』](#)의 내용을 참조하십시오.

사용자 컨텍스트는 다음 유형의 보고 메시지를 생성할 때 큐 관리자가 사용합니다.

- COD(Confirm on delivery)
- 만기

이러한 보고 메시지가 생성되는 경우, 보고서의 대상에 대한 +put 및 +passid 권한에 대해 사용자 컨텍스트가 확인됩니다. 사용자 컨텍스트의 권한이 충분하지 않은 경우 보고 메시지는 정의된 데드-레터 큐가 있으면 이 큐에 넣어집니다. 데드-레터 큐가 없는 경우 보고 메시지는 제거됩니다.

모든 컨텍스트 정보는 메시지 디스크립터의 컨텍스트 필드에 저장됩니다. 정보 유형은 ID, 원본 및 사용자 컨텍스트 정보로 나뉘어집니다.

ID 컨텍스트

ID 컨텍스트 정보는 처음으로 메시지를 큐에 넣는 애플리케이션의 사용자를 식별합니다. 적절하게 권한이 부여된 애플리케이션은 다음 필드를 설정할 수 있습니다.

- 큐 관리자는 사용자를 식별하는 이름을 *UserIdentifier* 필드에 입력합니다. 큐 관리자가 이를 수행하는 방법은 애플리케이션이 실행 중인 환경에 따라 다릅니다.
- 큐 관리자는 메시지를 넣은 애플리케이션에서 판별한 토큰 또는 번호를 *AccountingToken* 필드에 입력합니다.
- 애플리케이션은 사용자 정보에 포함시킬 추가 정보(예: 암호화된 비밀번호)를 위해 *ApplIdentityData* 필드를 사용할 수 있습니다.

시스템 보안 식별자 Windows (SID)는 메시지가 아래에서 생성될 때 필드에 *AccountingToken* IBM MQ for Windows 저장됩니다. SID는 *UserIdentifier* 필드를 보충하고 사용자의 신임 정보를 설정하는 데 사용될 수 있습니다.

큐 관리자가 *UserIdentifier* 및 *AccountingToken* 필드를 입력하는 방법에 대한 정보는 *UserIdentifier* 및 *AccountingToken*에서 해당 필드의 설명을 참조하십시오. V.10.0.0 참고: 긴 사용자 식별자가 사용 중이고 '실제' 사용자 이름이 12자를 초과할 경우, 메시지 컨텍스트 내에서 첫 11자로 잘린 후 '+' 기호가 접미사로 추가됩니다.

한 큐 관리자에서 다른 큐 관리자로 메시지를 전달하는 애플리케이션은 ID 컨텍스트 정보도 전달하여 다른 애플리케이션이 메시지 진원지의 ID를 알 수 있도록 해야 합니다.

원본 컨텍스트

원본 컨텍스트 정보는 메시지가 현재 저장되어 있는 큐에 메시지를 넣는 애플리케이션을 설명합니다. 메시지 디스크립터에는 원본 컨텍스트 정보에 대한 다음 필드가 포함되어 있습니다.

- *PutApplType*은 메시지를 넣는 애플리케이션의 유형을 정의합니다(예: CICS 트랜잭션).
- *PutApplName*은 메시지를 넣는 애플리케이션의 이름을 정의합니다(예: 작업 또는 트랜잭션의 이름).
- *PutDate*는 메시지를 큐에 넣은 날짜를 정의합니다.
- *PutTime*은 메시지를 큐에 넣은 시간을 정의합니다.
- *ApplOriginData*는 애플리케이션이 메시지 원본에 대해 포함시키려는 추가 정보를 정의합니다. 예를 들어, ID 데이터를 신뢰할 수 있는지 표시하기 위해 적절한 권한이 부여된 애플리케이션에서 설정할 수 있습니다.

원본 컨텍스트 정보는 일반적으로 큐 관리자가 제공합니다. *PutDate* 및 *PutTime* 필드는 그리니치 표준시 (GMT)를 사용합니다. *PutDate* 및 *PutTime*에서 이러한 필드에 대한 설명을 참조하십시오.

충분한 권한을 가진 애플리케이션은 자체 컨텍스트를 제공할 수 있습니다. 이는 생성된 메시지를 처리하는 각 시스템에서 단일 사용자가 다른 사용자 ID를 가질 때 계정 정보가 보존될 수 있도록 합니다.

IBM MQ 오브젝트

이 정보는 큐 관리자, 큐 공유 그룹, 큐, 관리 주제 오브젝트, 이름 목록, 프로세스 정의, 인증 정보 오브젝트, 채널, 스토리지 클래스, 리스너 및 서비스가 포함된 IBM MQ 오브젝트에 대한 세부사항을 제공합니다.

큐 관리자는 이러한 오브젝트의 특성(속성이라고 함)을 정의합니다. 이러한 속성 값은 IBM MQ에서 이러한 오브젝트를 처리하는 방법에 영향을 미칩니다. 애플리케이션에서 MQI(Message Queue Interface)를 사용하여 이러한 오브젝트를 제어합니다. 오브젝트는 프로그램에서 주소가 지정될 때 오브젝트 디스크립터(MQOD)로 식별됩니다.

예를 들어, 오브젝트를 정의, 변경 또는 삭제하기 위해 IBM MQ 명령을 사용하는 경우 큐 관리자는 이러한 조작을 수행할 필수 권한 레벨이 있는지 확인합니다. 마찬가지로, 애플리케이션이 MQOPEN 호출을 사용하여 오브젝트를 열 때 큐 관리자는 애플리케이션이 필요한 권한 레벨을 가지고 있는지 확인한 후 해당 오브젝트에 대한 액세스를 허용합니다. 열려 있는 오브젝트의 이름에 대해 확인합니다.

관련 개념

778 페이지의 『메시지 컨텍스트 정보 제어』

MQPUT 또는 MQPUT1 호출을 사용하여 메시지를 큐에 넣을 때 큐 관리자가 일부 기본 컨텍스트 정보를 메시지 디스크립터에 추가하도록 지정할 수 있습니다. 적절한 권한 레벨이 있는 애플리케이션이 추가 컨텍스트 정보를 추가할 수 있습니다. MQPMO 구조의 옵션 필드를 사용하여 컨텍스트 정보를 제어할 수 있습니다.

관련 정보

770 페이지의 『메시지 컨텍스트와 관련된 MQOPEN 옵션』

Windows Microsoft Transaction Server 애플리케이션 준비 및 실행

MTS 애플리케이션을 IBM MQ MQI client 애플리케이션으로 실행하기 위해 준비하려면 환경에 적합하도록 다음 지시사항을 따르십시오.

IBM MQ 리소스에 액세스하는 Microsoft 트랜잭션 서버(MTS) 애플리케이션을 개발하는 방법에 대한 일반적인 정보는 IBM MQ 헬프 센터의 MTS 섹션을 참조하십시오.

MTS 애플리케이션을 IBM MQ MQI client 애플리케이션으로 실행하기 위해 준비하려면, 애플리케이션의 각 컴포넌트에 대해 다음 중 하나를 수행하십시오.

- 컴포넌트에서 MQI용 C 언어 바인딩을 사용하는 경우 952 페이지의 『Windows에서 C 프로그램 준비』의 지시사항을 따르십시오. 단, 컴포넌트를 mqic.lib 대신 라이브러리 mqicxa.lib에 링크하십시오.
- 컴포넌트에서 IBM MQ C++ 클래스를 사용하는 경우 516 페이지의 『Windows에 C++ 프로그램 빌드』의 지시사항을 따르십시오. 단, 컴포넌트를 imqc23vn.lib 대신 라이브러리 imqx23vn.lib에 링크하십시오.
- 컴포넌트가 'MQI' 용 Visual Basic 언어 바인딩을 사용하는 경우, Visual Basic 프로젝트를 정의할 때 '조건부 컴파일 인수' 필드에 를 입력하십시오 MqType=3 .

IBM MQ 애플리케이션에 대한 설계 고려사항

애플리케이션에서 사용 가능한 플랫폼과 환경을 이용할 수 있는 방법을 결정한 경우 IBM MQ에서 제공한 기능의 사용 방법을 결정해야 합니다.

참고: **Deprecated** 부터 IBM MQ 9.3.0, 모든 플랫폼에서 네트워크 및 로컬 바인딩 연결 모두에 대해 32비트 IBM MQ 애플리케이션 라이브러리가 더 이상 사용되지 않습니다. 32비트 애플리케이션은 향후 IBM MQ 애플리케이션 라이브러리로의 IBM MQ 마이그레이션을 보장하기 위해 64비트로 재컴파일해야 합니다.

Removed 버전부터 IBM MQ 9.4.0, 지원되는 일부 운영 체제에서 지원이 중단됨에 따라 에 IBM MQ 대한 31비트 Linux on IBM Z® 라이브러리 지원이 제거되었습니다.

IBM MQ 애플리케이션을 설계하는 경우 다음 질문과 옵션을 고려하십시오.

애플리케이션 유형

애플리케이션의 목적은 무엇입니까? 개발 가능한 다양한 애플리케이션 유형에 대한 정보는 관련 링크를 참조하십시오:

- 서버
- 클라이언트
- 발행/구독
- 웹 서비스
- 사용자 엑시트, API 엑시트 및 설치 가능 서비스

추가적으로 IBM MQ의 관리를 자동화하는 사용자 자체 애플리케이션을 작성할 수 있습니다. 자세한 정보는 MQAI(IBM MQ Administration Interface) 및 관리 태스크 자동화를 참조하십시오.

프로그래밍 언어

IBM MQ는 애플리케이션 작성을 위해 다양한 프로그래밍 언어를 지원합니다. 자세한 정보는 5 페이지의 『IBM MQ용 애플리케이션 개발』의 내용을 참조하십시오.

두 개 이상의 플랫폼을 위한 애플리케이션

애플리케이션이 두 개 이상의 플랫폼에서 실행됩니까? 오늘 사용하는 플랫폼에서 다른 플랫폼으로 이동시킬 전략이 있습니까? 위의 질문 중 하나에 대한 대답이 예라면 플랫폼의 독립성을 위해 프로그램을 코딩해야 합니다.

예를 들어, C를 사용하는 경우 ANSI 표준 C로 코딩하십시오. 플랫폼별 기능이 더 빠르거나 효율적인 경우에도 동등한 플랫폼별 기능이 아닌 표준 C 라이브러리 함수를 사용하십시오. 코드의 효율성이 가장 중요한 경우, #ifdef를 사용하는 두 상황 모두에 대해 코딩해야 하는 경우는 예외입니다. 예를 들면, 다음과 같습니다.

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

큐 유형

필요할 때마다 큐를 작성하시겠습니까? 또는 이미 설정되어 있는 큐를 사용하시겠습니까? 사용이 끝나면 큐를 삭제하시겠습니까? 또는 다시 사용하시겠습니까? 애플리케이션 독립성을 위해 알리어스 큐를 사용하시겠습니까? 지원되는 큐의 유형을 보려면 큐를 참조하십시오.

z/OS 공유 큐, 큐 공유 그룹 및 큐 공유 그룹 클러스터 사용(IBM MQ for z/OS 전용)

큐 공유 그룹으로 공유 큐를 사용할 때 가능해지는 증가된 가용성, 확장성 및 워크로드 밸런싱의 장점을 활용할 수 있습니다. 자세한 정보는 [공유 큐 및 큐 공유 그룹](#)을 참조하십시오.

또한 평균 및 최대 메시지 플로우를 추정하고 워크로드 분산을 위해 큐 공유 그룹 클러스터의 사용을 고려할 수 있습니다. 자세한 정보는 [공유 큐 및 큐 공유 그룹](#)을 참조하십시오.

큐 관리자 클러스터 사용

간소화된 시스템 관리 및 클러스터를 사용하는 경우 가능해지는 향상된 가용성, 확장성 및 워크로드 밸런싱을 활용할 수 있습니다.

메시지 유형

단순 메시지에 데이터그램을 사용하고 기타 상황에 대해서는 요청 메시지(응답을 기대하는)를 사용할 수 있습니다. 일부 메시지에 다른 우선순위를 지정할 수 있습니다. 메시지 설계에 대한 자세한 정보는 [61 페이지](#)의 [『메시지를 위한 설계 기술』](#)의 내용을 참조하십시오.

발행/구독 또는 포인트-투-포인트 메시징 사용

발행/구독 메시징을 사용하여 송신 애플리케이션은 IBM MQ 메시지에서 공유하려는 정보를 IBM MQ 발행/구독에서 관리하는 표준 대상으로 보내며 IBM MQ에서 해당 정보의 분배를 처리하게 합니다. 대상 애플리케이션은 수신하는 정보의 소스에 대해 알 필요가 없으며 단지 하나 이상의 주제에 관심을 표명하여 정보가 사용 가능할 때 해당 정보를 수신합니다. 발행/구독 메시징에 대한 자세한 정보는 [발행/구독 메시징](#)을 참조하십시오.

포인트-투-포인트 메시징을 사용하여 송신 애플리케이션은 수신 애플리케이션이 메시지를 검색할 것으로 알려진 특정 큐에 메시지를 송신합니다. 수신 애플리케이션은 특정 큐에서 메시지를 가져오고 해당 콘텐츠에 대해 작업을 수행합니다. 애플리케이션은 종종 다른 애플리케이션에 조회를 송신하거나 응답을 수신하는 송신자 및 수신자 기능을 합니다.

IBM MQ 프로그램 제어

일부 프로그램을 자동으로 시작하거나 특정 메시지가 큐에 도착할 때까지 프로그램을 대기하도록 할 수 있습니다(IBM MQ 트리거링 기능 사용, [873 페이지](#)의 [『트리거를 사용한 IBM MQ 애플리케이션 시작』](#) 참조). 대안으로, 큐의 메시지가 충분히 빠르게 처리되지 않을 때 애플리케이션의 다른 인스턴스를 시작할 수도 있습니다(인스트루멘테이션 IBM MQ 이벤트 기능 사용, 자세한 내용은 [인스트루멘테이션 이벤트](#) 참조).

IBM MQ 클라이언트에서 애플리케이션 실행

전체 MQI는 클라이언트 환경에서 지원되며 프로시저 언어로 작성된 거의 모든 IBM MQ 애플리케이션을 IBM MQ MQI client에서 실행하기 위해 다시 링크할 수 있습니다. IBM MQ MQI client의 애플리케이션을

MQI 라이브러리가 아닌 MQIC 라이브러리에 연결하십시오.  **z/OS** z/OS에서 Get(신호)은 지원되지 않습니다.

참고: IBM MQ 클라이언트에서 실행 중인 애플리케이션은 동시에 둘 이상의 큐 관리자에 연결하거나, MQCONN 또는 MQCONNX 호출에 별표(*)가 있는 큐 관리자 이름을 사용할 수 있습니다. 이 기능이 사용 가능하지 않을 때 클라이언트 라이브러리 대신에 큐 관리자 라이브러리에 링크하려면 애플리케이션을 변경하십시오.

자세한 정보는 [923 페이지](#)의 [『IBM MQ MQI client 환경에서의 애플리케이션 실행』](#)의 내용을 참조하십시오.

애플리케이션 성능

설계 의사결정은 애플리케이션 성능에 영향을 줄 수 있습니다. IBM MQ 애플리케이션의 성능을 향상시키기 위한 제안은 [62 페이지](#)의 [『애플리케이션 설계 및 성능 고려사항』](#)  **IBM i** 및 [65 페이지](#)의 [『IBM i 애플리케이션에 대한 설계 및 성능 고려사항』](#)의 내용을 참조하십시오.

고급 IBM MQ 기술

고급 애플리케이션의 경우에는 응답 상관, IBM MQ 컨텍스트 정보의 생성 및 전송과 같은 일부 고급 IBM MQ 기술을 사용하려고 할 수 있습니다. 자세한 정보는 [64 페이지](#)의 [『고급 애플리케이션을 위한 기술 설계』](#)의 내용을 참조하십시오.

데이터 보안 및 데이터 무결성 유지보수

메시지와 함께 전달된 컨텍스트 정보를 사용하여 해당 메시지가 승인 가능한 소스에서 전송되었는지 테스트할 수 있습니다. IBM MQ 또는 운영 체제에서 제공하는 동기점 조정 기능을 사용하여 데이터가 다른 자원과 일치하는 상태로 남아 있는지 확인할 수 있습니다(자세한 내용은 [863 페이지](#)의 [『작업 단위 커밋 및 백아웃』](#) 참조). 메시지의 IBM MQ 지속성 기능을 사용하여 중요한 메시지의 전달을 보장할 수 있습니다.

IBM MQ 애플리케이션 테스트

IBM MQ 프로그램에 대한 애플리케이션 개발 환경이 다른 어떤 애플리케이션에 대한 환경과 다르지 않으므로 동일한 개발 도구 및 IBM MQ 추적 기능을 사용할 수 있습니다.

z/OS 애플리케이션을 CICS 테스트할 때 실행 진단 기능(IBM MQ for z/OS CICS CEDF)을 사용할 수 있습니다. CEDF는 모든 CICS 서비스에 대한 호출뿐만 아니라 모든 MQI 호출의 엔트리 및 엑시트를 추적합니다. 또한 CICS 환경에서 모든 MQI 호출 전후에 진단 정보를 제공하기 위한 API 교환 엑시트 프로그램을 작성할 수 있습니다. 이를 수행하는 방법에 대한 정보는 895 페이지의 『IBM MQ for z/OS의 애플리케이션 사용 및 작성』의 내용을 참조하십시오.

IBM i IBM i 애플리케이션을 테스트할 때 표준 디버거를 사용할 수 있습니다. 이를 시작하려면 STRDBG 명령을 사용하십시오.

예외 및 오류 핸들링

전달할 수 없는 메시지를 처리하는 방법 및 큐 관리자가 보고한 오류 상황을 해결하는 방법에 대해 고려해야 합니다. 일부 보고서의 경우 MQPUT에 보고서 옵션을 설정해야 합니다.

관련 개념

IBM MQ 기술 개요

67 페이지의 『z/OS 애플리케이션에 대한 설계 및 성능 고려사항』
애플리케이션 디자인은 성능에 영향을 미치는 가장 중요한 요소 중 하나입니다. 이 주제를 사용하여 성능에 관련된 일부 설계 요소에 대해 이해할 수 있습니다.

5 페이지의 『IBM MQ용 애플리케이션 개발』

메시지를 송신하고 수신하며, 큐 관리자와 관련 자원을 관리하기 위한 애플리케이션을 개발할 수 있습니다. IBM MQ는 많은 다양한 언어와 프레임워크로 작성된 애플리케이션을 지원합니다.

6 페이지의 『애플리케이션 개발 개념』

사용자는 원하는 절차적 또는 객체 지향 언어를 사용하여 IBM MQ 애플리케이션을 작성할 수 있습니다. IBM MQ 애플리케이션을 디자인하고 작성하기 전에 기본 IBM MQ 개념을 숙지하십시오.

743 페이지의 『큐잉을 위한 프로시저 애플리케이션 작성』

이 정보를 사용하여 큐잉 애플리케이션 작성, 큐 관리자에 연결 및 연결 끊기, 발행/구독 및 오브젝트 열기 및 닫기에 대해 알아보십시오.

917 페이지의 『클라이언트 프로시저 애플리케이션 작성』

프로시저 언어를 사용하여 IBM MQ에서 클라이언트 애플리케이션을 작성할 때 알아야 할 사항입니다.

494 페이지의 『C++ 애플리케이션 개발』

IBM MQ에서는 IBM MQ 오브젝트에 해당하는 C++ 클래스와 배열 데이터 유형에 해당하는 추가 클래스를 제공합니다. MQI를 통해 사용할 수 없는 여러 기능을 제공합니다.

83 페이지의 『사용 IBM MQ classes for JMS/Jakarta Messaging』

IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 는 와 함께 제공되는 IBM MQ메시징 Java 공급자입니다. 이 메시징 공급자들은 및 Jakarta Messaging JMS 사양에서 정의된 인터페이스를 구현하는 것 외에도, 메시징 Java API에 두 가지 확장 기능을 추가합니다.

320 페이지의 『IBM MQ classes for Java 사용』

사용 IBM MQJava 환경에서 사용하십시오. IBM MQ classes for Java를 사용하면 Java 애플리케이션을 IBM MQ 클라이언트로 IBM MQ에 연결하거나 IBM MQ 큐 관리자에 직접 연결할 수 있습니다.

관련 태스크

520 페이지의 『.NET 애플리케이션 개발』

IBM MQ classes for .NET 애플리케이션이 로 IBM MQ MQI clientIBM MQ 연결하거나 .NETIBM MQ 서버에 직접 연결할 수 있도록 허용합니다.

지원되는 프로그래밍 언어로 애플리케이션 이름 지정

IBM MQ 9.2.0 이전에 이미 Java 또는 JMS 클라이언트 애플리케이션에 애플리케이션 이름을 지정할 수 있었습니다. IBM MQ 9.2.0부터 이 기능은 IBM MQ for Multiplatforms의 다른 프로그래밍 언어로 확장되었습니다.

애플리케이션 이름 사용 방법

애플리케이션 이름은 다음 명령에서 제공하는 출력입니다.

- runmqsc DISPLAY CONN APPLTAG
- runmqsc DISPLAY QSTATUS TYPE(HANDLE) APPLTAG
- runmqsc DISPLAY CHSTATUS RAPPLTAG
- MQMD.PutApplName
- 애플리케이션 활동 추적

애플리케이션 이름은 애플리케이션 활동 추적을 구성할 때도 사용됩니다. 기본 애플리케이션 이름은 실행Java 파일의 잘린 이름입니다. 단, IBM i 및 Windows에서는 예외입니다.

Windows Windows의 경우, 기본 이름은 완전한 실행 파일 이름(왼쪽에서 28자까지 잘림)입니다.

IBM i IBM i의 경우 기본 이름이 작업 이름입니다.

Java 애플리케이션의 경우 패키지 이름이 접두부로 붙는 클래스 이름(왼쪽에서 28자까지 잘림)입니다.

자세한 정보는 [PutApplName](#)을 참조하십시오.

애플리케이션은 관리적으로 또는 다양한 프로그래밍 방법을 IBM MQ for Multiplatforms 사용하여 애플리케이션 이름을 설정할 수 있습니다. 따라서 애플리케이션 활동 추적을 구성할 때나 다양한 **runmqsc** 명령에서 출력할 때 더 의미 있는 플랫폼 독립적인 이름을 제공할 수 있습니다.

응용 프로그램을 균일한 클러스터 전반에 걸쳐 재조정할 수 있습니다. 이 기능을 얻는 데 의미 있는 애플리케이션 이름이 사용됩니다.

지원되는 문자

애플리케이션 이름 지정 방법에 대한 자세한 정보는 [57 페이지의 『추천된 애플리케이션 이름 문자』](#)의 내용을 참조하십시오.

프로그래밍 언어

C 및 기타 프로그래밍 언어에서 IBM MQ 라이브러리를 참조하는 애플리케이션이 애플리케이션 이름을 제공하는 방법에 대한 자세한 내용은 [58 페이지의 『프로그래밍 언어 연결』](#) 참조하십시오.

관리 .NET 애플리케이션

관리되는 .NET 애플리케이션이 애플리케이션 이름을 제공하는 방법에 대한 정보는 [59 페이지의 『관리 .NET 애플리케이션』](#) 참조하십시오.

XMS 애플리케이션

XMS 애플리케이션이 애플리케이션 이름을 제공할 수 있는 방법에 대한 정보는 [60 페이지의 『XMS 애플리케이션』](#)의 내용을 참조하십시오.

Java 및 JMS 바인딩 애플리케이션

ALW

애플리케이션이 JMS Java 애플리케이션 이름을 제공하는 방법에 대한 정보는 [60 페이지의 『Java 및 JMS 바인딩 애플리케이션』](#) 참조하십시오.

관련 개념

[애플리케이션 활동 추적](#)

[균등 클러스터 정보](#)

지원되는 프로그래밍 언어로 애플리케이션 이름 사용

애플리케이션 이름이 IBM MQ가 지원하는 다양한 언어에서 선택되는 방법을 알아보려면 이 정보를 사용하십시오.

추천된 애플리케이션 이름 문자

응용 프로그램 이름은 큐 관리자 필드의 **CodedCharSetId** 속성에서 지정된 문자 집합으로 지정되어야 합니다. 이 속성에 대한 자세한 내용은 [큐 관리자 속성](#)을 참조하십시오.

그러나 애플리케이션이 IBM MQ MQI client로 실행 중인 경우, 애플리케이션 이름은 클라이언트의 문자 세트 및 인코딩으로 되어 있어야 합니다.

큐 관리자 간 애플리케이션 이름의 원활한 전환을 보장하고, 리소스 모니터링 토픽을 통한 애플리케이션 리소스 모니터링을 허용하기 위해 애플리케이션 이름에는 단일 바이트 인쇄 가능 문자만 포함되어야 합니다.

참고:

- 또한 애플리케이션 이름에 슬래시 및 앰퍼샌드 문자를 사용해서는 안 됩니다.
- 애플리케이션 이름에 앰퍼샌드 문자를 사용하지 않아야 합니다. 앰퍼샌드가 포함된 애플리케이션 이름에 대한 시스템 토픽 STATAPP 메트릭은 생성되지 않습니다.

이 이름은 다음으로 제한됩니다.

- 영숫자 문자: A-Z, a-z 및 0-9

참고: EBCDIC Katakana를 사용하는 시스템의 애플리케이션 이름에 소문자 a-z 문자를 사용하지 마십시오.

- 공백 문자
- EBCDIC에서 불변인 인쇄 가능 문자: + < = > % * ' () , _ - . : ; ?
- / 문자. 이름에 슬래시가 포함된 애플리케이션의 활동 추적 또는 STATAPP 시스템 토픽 메트릭을 구독할 때에는 모든 슬래시 문자를 앰퍼샌드 문자로 바꿔야 합니다. 예를 들어, "DEPT1/APPS/STOCKQUOTE"란 애플리케이션에 대한 STATAPP 메트릭을 수신하려면 "\$SYS/MQ/INFO/QMGR/QMBASIC/Monitor/STATAPP/DEPT1&APPS&STOCKQUOTE/INSTANCE" 토픽 문자열을 구독해야 합니다. amqsact 및 amqsrua 샘플 애플리케이션은 구독을 작성할 때 자동으로 슬래시 문자를 앰퍼샌드로 변환합니다.

문자 설정 방법

다음 표에는 IBM MQ가 지원하는 다양한 언어로 애플리케이션 이름을 선택하는 방법이 요약되어 있습니다. 이름을 선택하는 방법은 가장 높은 우선 순위의 순서로 선택합니다.

	C 바인딩 및 클라이언트	Java 바인딩 및 클라이언트	JMS 바인딩 및 클라이언트	관리 .NET 클라이언트	비관리 .NET 바인딩 및 클라이언트	관리 XMS 클라이언트	비관리 .XMS 바인딩 및 클라이언트
연결 특성 대체		Java 연결 특성 대체		.NET 연결 특성 대체	.NET 연결 특성 대체		
대체된 특성		Java 대체된 특성		.NET 대체된 특성	.NET 대체된 특성		

	C 바인딩 및 클라이언트	Java 바인딩 및 클라이언트	JMS 바인딩 및 클라이언트	관리 .NET 클라이언트	비관리 .NET 바인딩 및 클라이언트	관리 XMS 클라이언트	비관리 XMS 바인딩 및 클라이언트
MQEnvironment		Java MQEnvironment		.NET MQEnvironment	.NET MQEnvironment		
연결 팩토리 특성			연결 팩토리 특성			연결 팩토리 특성	연결 팩토리 특성
JMSAdmin			JMSAdmin			JMSAdmin	JMSAdmin
MQCNO	연결 옵션						
환경 변수	환경 변수				환경 변수		환경 변수
mqclient.ini (클라이언트 연결에만 적용 가능)	클라이언트 연결				클라이언트 연결		클라이언트 연결
Java 클래스 이름		Java 클래스 이름	Java 클래스 이름				
기본 이름	기본 이름			.NET 기본 이름	.NET 기본 이름	.NET 기본 이름	.NET 기본 이름

참고: C 바인딩 및 클라이언트 열은 다음 프로그래밍 언어에도 적용됩니다.

- COBOL
- 어셈블러
- Visual Basic
- RPG

프로그래밍 언어 연결

C에서 IBM MQ 라이브러리로 해석되는 애플리케이션 및 기타 프로그래밍 언어는 다음과 같은 방법으로 애플리케이션 이름을 제공할 수 있습니다.

연결 방법은 가장 높은 순서부터 우선 순위의 순서대로 나열됩니다.

Multi 연결 옵션

- **ALW** MQCNO

참고: **z/OS** 큐 IBM MQ for z/OS 매니저에 연결할 때는 클라이언트 모드 연결을 사용하거나 IBM MQ classes for JMS 또는 IBM MQ classes for Java 애플리케이션을 통해서만 애플리케이션 이름을 설정할 수 있습니다.

- **IBM i** IBM i의 MQCNO

ALW 환경 변수

아직 애플리케이션 이름을 선택하지 않았다면, 환경 **MQAPPLNAME** 변수를 사용하여 큐 관리자에 대한 연결을 식별할 수 있습니다. 예를 들면, 다음과 같습니다.

```
export MQAPPLNAME=ExampleAppName
```

처음 28자만 사용되며 이 문자는 모두 공백이거나 널일 수 없다는 것을 참고하십시오.

참고: 이 속성은 지원되는 프로그래밍 언어, 비관리형 .NET 및 비관리형 XMS 연결에만 적용됩니다.

클라이언트 구성 파일

아직 애플리케이션 이름을 선택하지 않았고 연결이 클라이언트 연결인 경우, 큐 매니저에 대한 연결을 식별하기 위해 클라이언트 구성 파일(예: mqclient.ini)에 다음 정보를 지정할 수 있습니다.

```
Connection:
  AppName=ExampleAppName
```

참고:

1. 처음 28자만 사용되며 이 문자는 모두 공백 또는 널이 아니어야 합니다.
2. 이 속성은 지원되는 프로그래밍 언어의 클라이언트 연결, .NET 비관리형 연결 및 XMS 비관리형 연결에만 적용됩니다.

자세한 내용은 을 참조하십시오 [IBM MQ MQI client 구성 파일, mqclient.ini](#).

기본 이름

애플리케이션 이름을 아직 선택하지 않은 경우 운영 체제가 표시하는 경로와 실행 파일 이름을 포함하는 기본 이름이 계속 사용됩니다. 자세한 정보는 [PutAppName](#)을 참조하십시오.

관리 .NET 애플리케이션

관리 .NET 애플리케이션은 다음의 방법으로 애플리케이션 이름을 제공할 수 있습니다.

연결 방법은 가장 높은 순서부터 우선 순위의 순서대로 나열됩니다.

연결 특성 대체

다음의 방법으로 연결 세부사항 대체 파일을 애플리케이션에 제공할 수 있습니다.

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

overrideConnectionDetailsFile에 의해 지정된 파일은 mqj가 접두부인 특성의 목록을 포함합니다. 애플리케이션은 **mqj.APPNAME** 특성을 정의해야 합니다. 여기서 **mqj.APPNAME** 특성의 값은 큐 관리자에 대한 연결을 식별하는 데 사용되는 이름을 지정합니다.

이름의 처음 28자만 사용됩니다. 예를 들면, 다음과 같습니다.

```
mqj.APPNAME=ExampleAppName
```

대체된 특성

MQC.APPNAME_PROPERTY 상수는 **APPNAME** 값으로 정의되었습니다. 이제 이름의 처음 28자만을 사용하여 **MQQueueManager** 생성자에 이 특성을 전달할 수 있습니다. 예를 들면, 다음과 같습니다.

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

자세한 정보는 599 페이지의 『[.NET의 관리 조작 및 비관리 조작](#)』의 내용을 참조하십시오.

MQEnvironment

속성이 *AppName* 클래스에 **MQEnvironment** 추가되며, 처음 28자만 사용됩니다. 예를 들면, 다음과 같습니다.

```
MQEnvironment.AppName = "ExampleApp1Name";
```

기본 이름

앞서 설명한 방법 중 어느 것으로도 응용 프로그램 이름을 지정하지 않은 경우, 응용 프로그램 이름은 자동으로 실행 파일 이름(및 포함될 수 있는 경로 부분)으로 설정됩니다.

XMS 애플리케이션

연결 방법은 가장 높은 순서부터 우선 순위의 순서대로 나열됩니다.

연결 팩토리 특성

XMS 애플리케이션은 JMS와 유사한 방식으로 연결 팩토리에서 "*XMSC_WMQ_APPNAME*" 속성을 **XMSC.WMQ_APPLICATIONNAME** 사용하여 애플리케이션 이름을 제공할 수 있습니다. 최대 28개의 문자를 지정할 수 있습니다.

자세한 정보는 624 페이지의 『[XMS .NET에서 관리 대상 오브젝트를 작성함](#)』 및 634 페이지의 『[XMS 메시지의 특성](#)』의 내용을 참조하십시오.

JMSAdmin

관리 도구에서 해당 속성은 "**APPLICATIONNAME**" 또는 줄여서 "**APPNAME**"로 알려져 있습니다.

Java 및 JMS 바인딩 애플리케이션

연결 방법은 가장 높은 순서부터 우선 순위의 순서대로 나열됩니다.

 Java 클라이언트 애플리케이션은 이미 애플리케이션 JMS 이름을 지정할 수 있으며, 이는 MQCNO **App1Name** 필드를 활용하여 바인딩 애플리케이션으로 IBM MQ for Multiplatforms 확장되었습니다.

연결 특성 대체

해당 **Application name** 속성이 재정의 가능한 연결 속성 목록에 추가되었습니다. 자세한 내용은 [연결 속성 IBM MQ 재정의의 사용](#)을 참조하십시오.



주의: 연결 특성 대체 파일을 사용하는 방법 및 연결 특성은 IBM MQ classes for Java 및 .NET에서 동일합니다.

대체된 특성

MQC.APPNAME_PROPERTY 상수는 *APPNAME* 값으로 정의되었습니다. 이제 이름의 처음 28자만을 사용하여 **MQQueueManager** 생성자에 이 특성을 전달할 수 있습니다. 자세한 내용은 [연결 속성 재정의의 사용을 IBM MQ classes for Java](#) 참조하십시오.

MQEnvironment

속성이 *AppName* 클래스에 **MQEnvironment** 추가되며, 처음 28자만 사용됩니다.

자세한 정보는 345 페이지의 『[IBM MQ classes for Java 에 대한 IBM MQ 환경 설정](#)』의 내용을 참조하십시오.

Java 클래스 이름

위 텍스트의 방법으로 애플리케이션 이름을 제공하지 않은 경우 애플리케이션 이름은 기본 클래스 이름에서 도출됩니다.

자세한 정보는 345 페이지의 『IBM MQ classes for Java 에 대한 IBM MQ 환경 설정』의 내용을 참조하십시오.



주의: IBM i IBM i에서 기본 클래스 이름을 조회하는 것은 불가능하므로 IBM MQ client for Java가 대신 사용됩니다.

관련 개념

345 페이지의 『IBM MQ classes for Java 에 대한 IBM MQ 환경 설정』

클라이언트 모드로 큐 관리자에 애플리케이션을 연결하려는 경우 애플리케이션은 채널 이름, 호스트 이름 및 포트 번호를 지정해야 합니다.

관련 참조

MQCNO

IBM i용 MQCNO

메시지를 위한 설계 기술

선택자 및 메시지 특성에 대한 고려사항을 포함하여 메시지를 디자인하는 데 유용한 고려사항입니다.

디자인 단계에서 고려할 사항

MQI 호출을 사용하여 큐에 메시지를 넣을 때 메시지를 작성합니다. 호출에 대한 입력으로 메시지 디스크립터(MQMD)의 일부 제어 정보와 다른 프로그램에 보낼 데이터를 제공합니다. 그러나 사용자의 메시지를 작성하는 방법에 영향을 미치기 때문에 설계 단계에서 다음을 고려해야 합니다.

사용할 메시지 유형

메시지를 보낸 후 추가 조치를 수행하지 않는 간단한 애플리케이션을 설계 중입니까? 또는 질문에 대한 응답을 요청하고 있습니까? 질문을 하고 있는 경우 메시지 디스크립터에 응답을 수신할 큐의 이름을 포함시킬 수 있습니다.

요청 및 응답 메시지의 동기화가 필요합니까? 이를 위해서는 요청에 답변하는 응답에 대한 제한시간 기간을 설정해야 하며, 해당 기간 내에 응답이 수신되지 않으면 오류로 처리됩니다.

또는 프로세스가 공통 타이밍 신호와 같은 특정 이벤트 발생에 영향을 받지 않도록 비동기적으로 작업하시겠습니까?

또한 모든 메시지를 하나의 작업 단위 내에 유지할지도 고려해야 합니다.

메시지에 다른 우선순위 지정

우선순위 값을 각 메시지에 지정할 수 있으며 해당 우선순위의 순서대로 메시지를 유지보수하도록 큐를 정의할 수 있습니다. 이렇게 하면 다른 프로그램에서 큐의 메시지를 검색할 때 항상 우선순위가 가장 높은 메시지가 검색됩니다. 큐에서 우선순위 순서대로 해당 메시지를 유지보수하지 않는 경우 큐에서 메시지를 검색하는 프로그램은 큐에 메시지가 추가된 순서대로 메시지를 검색합니다.

또한 프로그램에서는 메시지를 큐에 넣을 때 큐 관리자가 할당한 ID를 사용하여 메시지를 선택할 수 있습니다. 또는 각 메시지에 대해 자체 ID를 생성할 수 있습니다.

메시지에서 큐 관리자 재시작의 효과

큐 관리자는 모든 지속 메시지를 보존하며 큐 관리자가 재시작될 때 IBM MQ 로그 파일에서 필요한 경우 메시지를 복구합니다. 비지속 메시지 및 임시 동적 큐는 보존되지 않습니다. 제거하지 않을 메시지는 메시지가 작성될 때 지속 메시지로 정의되어야 합니다. AIX and Linux 시스템에서 IBM MQ for Windows 또는 IBM MQ 용 애플리케이션을 작성할 때 로그 파일 한계까지 실행할 애플리케이션을 디자인하는 위험을 줄이기 위해 로그 파일 할당과 관련하여 시스템이 설정된 방법을 알고 있어야 합니다.



공유 큐의 메시지(IBM MQ for z/OS에서만 사용 가능)가 커플링 기능(CF)에 보유되기 때문에 CF를 사용할 수 있는 동안은 큐 관리자의 재시작에서 비지속 메시지는 보존됩니다. CF에 실패하는 경우 비지속 메시지는 유실됩니다.

메시지 수신인에게 자신에 대한 정보 제공

일반적으로 큐 관리자가 사용자 ID를 설정하지만 적절한 권한이 부여된 애플리케이션에서도 이 필드를 설정하여 수신 프로그램에서 회계 또는 보안 목적으로 사용할 수 있도록 사용자 ID와 기타 정보를 포함시킬 수 있습니다.

수신 큐의 양

Multi 메시지를 여러 큐에 넣어야 할 수도 있는 경우 주제 또는 분배 목록에 발행할 수 있습니다.

z/OS 메시지를 여러 큐에 넣어야 할 수도 있는 경우 주제에 발행할 수 있습니다.

선택자 및 메시지 특성

메시지는 기본 메시지 페이로드와 함께 메시지와 연관된 메타데이터를 가질 수 있습니다. 이러한 메시지 특성은 추가적인 데이터를 제공하는 데 유용할 수 있습니다.

다음에 대해 알아야 하는 이 추가 데이터에 대한 두 가지 양상이 있습니다.

- 특성은 Advanced Message Security(AMS) 보호를 따르지 않습니다. 데이터를 보호하기 위해 AMS를 사용하려는 경우, 이를 메시지 특성이 아닌 페이로드에 넣으십시오.
- 특성은 메시지의 선택을 수행하는 데 사용될 수 있습니다.

선택자를 사용하는 것이 FIFO의 표준 메시지 규약을 깬다는 점에 주목하는 것이 중요합니다. 큐 관리자가 이 워크로드에 대해 최적화되어 있으므로 복잡한 선택자를 제공하는 것은 성능상의 이유로 권장되지 않습니다. 큐 관리자는 메시지 특성의 인덱스를 저장하지 않으므로 메시지 검색은 선형 검색이어야 합니다. 큐가 깊을수록 선택자가 복잡해지고 메시지와 일치하는 선택자가 성능에 부정적인 영향을 미칠 확률이 낮아집니다.

복잡한 선택사항이 필요한 경우, IBM Integration Bus 같은 처리 엔진이나 애플리케이션을 사용하여 메시지를 다른 목적지로 필터링하도록 제안됩니다. 또는 토픽 계층의 사용이 유용할 수 있습니다.

참고: IBM MQ classes for Java 는 선택기 사용을 지원하지 않습니다. 선택기를 사용하려면 JMS API를 통해 이를 수행해야 합니다.

애플리케이션 설계 및 성능 고려사항

프로그램 설계가 잘못되면 여러 면에서 성능에 영향을 줄 수 있습니다. 프로그램이 잘 수행되는 것으로 표시될 수 있으므로 감지하기 어려울 수 있지만 다른 작업에 영향을 줄 수 있습니다. IBM MQ 호출을 작성하는 프로그램에 특정한 여러 문제점에 대해 이 주제에 설명되어 있습니다.

효과적인 애플리케이션을 설계하는 데 도움이 되는 몇 가지 아이디어가 있습니다.

- 사용자가 생각하는 대로 바로 처리될 수 있도록 애플리케이션을 설계하십시오.
 - 패널을 표시하고 애플리케이션이 초기화되는 동안 사용자가 입력을 시작할 수 있도록 허용하십시오.
 - 필요한 데이터를 서로 다른 서버에서 동시에 가져오십시오.
- 반복적으로 큐를 열고, 닫고, 연결을 설정하고 다시 끊는 대신 이를 재사용하려는 경우 연결되어 있으며 큐가 열린 상태를 유지하십시오.
- 그러나 하나의 메시지만 넣는 서버 애플리케이션은 MQPUT1을 사용해야 합니다.
- 큐 관리자는 4KB와 100KB 사이의 크기를 갖는 메시지에 최적화되어 있습니다. 매우 큰 메시지의 경우는 비효율적입니다. 100MB의 단일 메시지보다 100개의 1MB짜리 메시지를 보내는 것이 나을 수 있습니다. 매우 작은 메시지 역시 효율적이지 않습니다. 큐 관리자는 단일 바이트 메시지에 대해 4KB 메시지와 동일한 양의 작업을 수행합니다.
- 메시지를 커밋하거나 동시에 백아웃할 수 있도록 작업 단위 내에서 메시지를 유지하십시오.
- 복구할 필요가 없는 메시지에 대해서는 비지속 옵션을 사용하십시오.
- 메시지를 여러 대상 큐에 송신해야 하는 경우 분배 목록의 사용에 대해 고려하십시오.

메시지 길이의 효과

메시지의 데이터 양은 메시지를 처리하는 애플리케이션의 성능에 영향을 줄 수 있습니다. 애플리케이션이 최고의 성능을 발휘하게 하려면, 메시지의 필수 데이터만 송신하십시오. 예를 들어, 은행 계좌 출금 요청의 경우 계좌 번호와 출금 금액만 클라이언트에서 서버 애플리케이션으로 전달되면 됩니다.

메시지 지속성의 효과

일반적으로 지속 메시지는 로깅됩니다. 메시지 로그를 기록하면 애플리케이션의 성능을 저하시키므로 필수 데이터에만 지속 메시지를 사용하십시오. 메시지의 데이터를 제거할 수 있는 경우 큐 관리자가 중지하거나 실패하면 비지속 메시지를 사용하십시오.

z/OS 조작을 기록하기 위한 복구 로그 공간이 충분하지 않으면 지속 메시지에 대한 MQPUT 및 MQGET 조작이 차단됩니다. 이러한 조건은 메시지 CSQJ110E 및 CSQJ111A에 의해 큐 관리자 작업 로그에 표시됩니다.. 이러한 조건을 관리하고 방지할 수 있도록 모니터링 프로세스가 작성되어 있는지 확인하십시오.

특정 메시지 검색

MQGET 호출은 대개 큐에서 첫번째 메시지를 검색합니다. 특정 메시지를 지정하기 위해 메시지 디스크립터에서 메시지 및 상관 ID를 사용하는 경우(MsgId 및 CorrelId) 큐 관리자는 해당 메시지를 찾을 때까지 큐를 검색해야 합니다. 이러한 방법으로 MQGET 호출을 사용하면 애플리케이션의 성능에 영향을 주게 됩니다.

다른 길이의 메시지를 포함하는 큐

애플리케이션에서 고정된 길이의 메시지를 사용할 수 없는 경우 일반 메시지 크기에 맞도록 버퍼를 동적으로 늘리거나 줄이십시오. 애플리케이션에서 실패하는 MQGET 호출을 발행하면 버퍼가 너무 작기 때문에 메시지 데이터의 크기가 리턴됩니다. 버퍼의 크기가 적절하게 조정되고 MQGET 호출이 다시 발행되도록 애플리케이션에 코드를 추가하십시오.

참고: MaxMsgLength 속성을 명시적으로 설정하지 않는 경우 4MB로 기본값이 설정되며 이 값이 애플리케이션 버퍼 크기에 영향을 미치는 경우 매우 비효율적일 수 있습니다.

동기점 빈도

동기점 내에 커밋하지 않고 대량의 MQPUT 또는 MQGET 호출을 발행하는 프로그램은 성능 문제점이 발생할 수 있습니다. 다른 태스크가 이러한 메시지를 가져오기 위해 대기하는 동안 영향을 받은 큐는 현재 액세스할 수 없는 메시지로 채워질 수 있습니다. 이것은 메시지를 가져오려는 태스크와 연결된 스토리지 및 스레드와 관련이 있습니다.

MQPUT1 호출의 사용

큐에 단일 메시지를 넣는 경우 MQPUT1 호출만 사용하십시오. 하나 이상의 메시지를 넣으려는 경우 MQOPEN 호출을 사용한 후 일련의 MQPUT 호출과 단일 MQCLOSE 호출을 사용하십시오.

사용 중인 스레드 수

Windows IBM MQ for Windows의 경우 애플리케이션은 다수의 스레드가 필요할 수 있습니다. 각 큐 관리자 프로세스에 허용 가능한 최대 애플리케이션 스레드가 할당됩니다.

애플리케이션은 아주 많은 스레드를 사용할 수 있습니다. 애플리케이션에서 이러한 가능성을 고려하고 있는지, 이러한 유형의 발생을 방지 또는 보고하기 위해 조치를 취하는지 고려하십시오.

동기점 아래 지속 메시지 넣기

지속 메시지는 동기점 아래 넣고 가져와야 합니다. 동기점 외부에서 지속 메시지를 가져올 때 가져오기에 실패하는 경우 애플리케이션에서 메시지가 큐로부터 가져오기되었는지 여부와 메시지가 가져오기 되었다면 유실되었는지 여부도 알 수 있는 방법이 없기 때문입니다. 동기점 아래에서 지속 메시지를 가져올 때 실패하는 경우 트랜잭션은 롤백되고 지속 메시지는 유실되지 않습니다. 메시지가 여전히 큐에 있기 때문입니다.

마찬가지로 지속 메시지를 넣을 때 동기점 아래에 두십시오. 동기점 아래에 메시지를 넣고 가져와야 하는 또다른 이유는 IBM MQ의 지속 메시지 코드가 동기점에 상당히 최적화되어 있기 때문입니다. 따라서 지속 메시지를 동기점 아래 넣고 가져오는 것이 동기점 외부에서 메시지를 넣고 가져오는 것보다 훨씬 빠릅니다.

애플리케이션이 동기점 외부에 지속 메시지를 넣는 경우, 큐 관리자는 애플리케이션을 대신하여 암시적인 동기점을 작성할 수 있는지 확인합니다. 큐 관리자가 작성할 수 있으면 여기에는 해당 동기점 내부로의 넣기가 포함되

고 이는 자동으로 커밋됩니다. 자세한 설명은 870 페이지의 『멀티플랫폼의 암시적 동기점』의 내용을 참조하십시오.

그러나 비지속 메시지는 동기점 외부에서 넣고 가져오는 것이 더 빠릅니다. 그 이유는 IBM MQ의 비지속 코드가 동기점 외부에 있는 것에 대해 최적화되어 있기 때문입니다. 지속 메시지는 디스크에 지속되기 때문에 디스크 속도로 지속 메시지를 넣고 가져옵니다. 그러나 동기점을 사용할 때가 아니라도 관련된 디스크 쓰기가 없기 때문에 비지속 메시지 넣기 및 가져오기는 CPU 속도에서 진행됩니다.

애플리케이션에서 메시지를 가져오고 미리 메시지가 지속 또는 비지속인지를 알 수 없는 경우 GMO 옵션 MQGMO_SYNCPOINT_IF_PERSISTENT를 사용할 수 있습니다.

고급 애플리케이션을 위한 기술 설계

고급 애플리케이션을 설계하는 경우 메시지 대기, 응답 상관, 컨텍스트 정보 설정 및 사용, 애플리케이션 자동 시작, 보고서 생성 및 메시지 연관관계 제거와 같이 클러스터링을 사용할 때 고려할 수 있는 몇 가지 기술이 있습니다.

단순 IBM MQ 애플리케이션의 경우 애플리케이션에 어떤 IBM MQ 오브젝트를 사용할지 및 사용하려는 메시지 유형을 결정해야 할 수 있습니다. 더 고급 애플리케이션의 경우 다음 섹션에서 소개되는 일부 기술을 사용하려고 할 수 있습니다.

메시지 대기

큐를 제공하는 프로그램은 다음을 기준으로 메시지를 기다릴 수 있습니다.

- 메시지가 도착하거나 지정된 시간 간격이 만료될 때까지 대기(811 페이지의 『메시지 대기』 참조).
-  IBM MQ for z/OS에서만 메시지가 도착할 때 프로그램에 알리도록 신호 설정. 자세한 정보는 812 페이지의 『신호보내기』의 내용을 참조하십시오.
- 메시지가 도착하는 경우 구동될 콜백 엑시트 설정. 46 페이지의 『IBM MQ 메시지의 비동기 이용』의 내용을 참조하십시오.
- 메시지가 도착했는지(폴링) 확인하기 위해 큐에서 주기적으로 호출 작성. 일반적으로는 성능에 영향을 미칠 수 있기 때문에 권장하지 않습니다.

응답 상관

IBM MQ 애플리케이션에서 프로그램이 일부 작업을 수행하도록 요청하는 메시지를 수신하는 경우 일반적으로 프로그램은 하나 이상의 응답 메시지를 요청자에게 송신합니다.

요청자가 이러한 응답을 원래 요청과 연관시키도록 도와주기 위해 애플리케이션은 각 메시지의 디스크립터에서 상관 ID 필드를 설정할 수 있습니다. 그런 다음 프로그램은 요청 메시지의 메시지 ID를 해당 응답 메시지의 상관 ID로 복사합니다.

컨텍스트 정보 설정 및 사용

컨텍스트 정보는 메시지를 생성한 사용자와 메시지를 연관시키고 메시지를 생성한 애플리케이션을 식별하는 데 사용됩니다. 이러한 정보는 보안, 회계 및 문제점 판별에 유용합니다.

메시지를 작성할 때 해당 큐 관리자가 기본 컨텍스트 정보를 메시지와 연관시키도록 요청하는 옵션을 지정할 수 있습니다.

컨텍스트 정보 사용 및 설정에 대한 자세한 정보는 51 페이지의 『메시지 컨텍스트』의 내용을 참조하십시오.

자동으로 IBM MQ 프로그램 시작

IBM MQ 트리거링을 사용하여 메시지가 큐에 도착하면 자동으로 프로그램을 시작하십시오.

프로그램이 해당 큐의 처리를 시작하도록 트리거 조건을 큐에 설정할 수 있습니다.

- 큐에 메시지가 도착할 때마다
- 첫 번째 메시지가 큐에 도착할 때

- 큐의 메시지 수가 사전 정의된 수에 도달하는 경우

트리거링에 대한 자세한 정보는 873 페이지의 [『트리거를 사용한 IBM MQ 애플리케이션 시작』](#)의 내용을 참조하십시오. 트리거링은 프로그램을 자동으로 시작하는 한가지 방법입니다. 예를 들어, 비IBM MQ 기능을 사용하여 타이머에서 자동으로 프로그램을 시작할 수 있습니다.

Multi 다중 플랫폼에서 IBM MQ는 큐 관리자가 시작될 때 IBM MQ 프로그램을 시작하는 서비스 오브젝트를 정의할 수 있습니다. [서비스 오브젝트](#)를 참조하십시오.

IBM MQ 보고서 생성

애플리케이션 내에서 다음 보고서를 요청할 수 있습니다.

- 예외 보고서
- 만기 보고서
- 도착 시 확인(COA) 보고서
- 전달 시 확인(COD) 보고서
- 긍정적인 조치 알림(PAN) 보고서
- 부정적인 조치 알림(NAN) 보고서

20 페이지의 [『보고 메시지』](#)에서 설명됩니다.

클러스터 및 메시지 연관관계

동일한 큐에 대해 여러 정의를 사용하여 클러스터 사용을 시작하기 전에 관련된 메시지를 교환해야 하는지 확인하기 위해 애플리케이션을 조사하십시오.

클러스터 내에서 메시지를 적절한 큐의 인스턴스를 호스팅하는 큐 관리자로 라우팅할 수 있습니다. 따라서 메시지 연관관계가 포함된 애플리케이션의 로직에 문제가 생길 수 있습니다.

예를 들어, 질문서와 답변의 양식으로 애플리케이션 간에 이동하는 일련의 메시지에 의존하는 두 개의 애플리케이션이 있을 수 있습니다. 모든 질문이 동일한 큐 관리자로 송신되고 모든 응답이 다른 큐 관리자에 다시 송신되는 것이 중요합니다. 이 경우 워크로드 관리 루틴에서는 적절한 큐의 인스턴스를 호스팅하게 되는 큐 관리자에 메시지를 송신하지 않는다는 것이 중요합니다.

가능한 경우 연관관계를 제거하십시오. 메시지 연관관계를 제거하면 애플리케이션의 가용성 및 확장성이 향상됩니다.

자세한 정보는 [메시지 연관관계 처리](#)의 내용을 참조하십시오.

IBM i 애플리케이션에 대한 설계 및 성능 고려사항

이 정보를 사용하여 애플리케이션 디자인, 스레드 및 스토리지가 성능에 어떠한 영향을 미칠 수 있는지 이해할 수 있습니다.

이 정보는 두 가지 섹션으로 나뉩니다.

- 65 페이지의 [『애플리케이션 설계 고려사항』](#)
- 66 페이지의 [『특정 성능 문제』](#)

애플리케이션 설계 고려사항

프로그램 설계가 잘못되면 여러 면에서 성능에 영향을 줄 수 있습니다. 프로그램이 잘 수행되는 것으로 표시될 수 있으므로 이러한 문제점을 감지하기 어려울 수 있지만 다른 태스크의 성능에 영향을 줄 수 있습니다. IBM MQ for IBM i 호출을 작성하는 프로그램에 특정한 여러 문제점에 대해 다음 섹션에서 설명됩니다.

애플리케이션 설계에 대한 자세한 정보는 53 페이지의 [『IBM MQ 애플리케이션에 대한 설계 고려사항』](#)의 내용을 참조하십시오.

메시지 길이의 효과

IBM MQ for IBM i를 통해 메시지를 최대 100MB까지 유지할 수 있더라도 메시지의 데이터 양은 메시지를 처리하는 애플리케이션의 성능에 영향을 줍니다. 애플리케이션이 최고의 성능을 발휘하려면, 메시지의 필수 데이터만 송신하십시오. 예를 들어, 은행 계좌 출금 요청의 경우 계좌 번호와 출금 금액만 클라이언트에서 서버 애플리케이션으로 전달되면 됩니다.

메시지 지속성의 효과

지속 메시지는 저널링됩니다. 메시지를 저널링하면 애플리케이션의 성능이 저하되므로 필수 데이터에만 지속 메시지를 사용하십시오. 메시지의 데이터를 제거할 수 있는 경우 큐 관리자가 중지하거나 실패하면 비지속 메시지를 사용하십시오.

특정 메시지 검색

MQGET 호출은 대개 큐에서 첫번째 메시지를 검색합니다. 메시지 디스크립터에서 메시지 및 상관 ID(*MsgId* 및 *CorrelId*)를 사용하여 특정 메시지를 지정하는 경우, 큐 관리자는 해당 메시지를 찾을 때까지 큐를 검색해야 합니다. 이 방법으로 MQGET 호출을 사용하면 애플리케이션의 성능에 영향을 주게 됩니다.

다른 길이의 메시지를 포함하는 큐

큐에 있는 메시지의 길이가 서로 다르면 메시지의 크기를 판별하기 위해 애플리케이션에서 *BufferLength* 필드가 0으로 설정된 MQGET 호출을 사용할 수 있습니다. 그러면 호출에 실패하더라도 메시지 데이터의 크기를 리턴할 수 있습니다. 그런 다음 애플리케이션은 첫 번째 호출에서 측정된 메시지 ID 및 올바른 크기의 버퍼를 지정하여 호출을 반복할 수 있습니다. 그러나 동일한 큐에 서비스를 제공하는 다른 애플리케이션이 있는 경우, 두 호출 사이 시간 동안 다른 애플리케이션이 검색한 메시지를 두 번째 MQGET 호출에서 검색하는데 시간을 소모하기 때문에 애플리케이션의 성능이 저하됩니다.

애플리케이션에서 고정 길이의 메시지를 사용할 수 없다면 이 문제에 대한 다른 해결 방법은 큐가 허용할 수 있는 최대 크기의 메시지를 찾기 위해 MQINQ 호출을 사용하는 것입니다. 그런 다음 MQGET 호출에 이 값을 사용하십시오. 큐의 최대 메시지 크기는 큐의 **MaxMsgLen** 속성에 저장됩니다. 이 방법은 많은 양의 스토리지를 사용할 수 있지만, 이 큐 속성 값이 IBM MQ for IBM i에서 허용된 최대일 수 있으며 2GB보다 클 수 있습니다.

동기점 빈도

동기점 내에 커밋하지 않고 대량의 MQPUT 호출을 발행하는 프로그램은 성능 문제점이 발생할 수 있습니다. 다른 태스크가 이러한 메시지를 가져오기 위해 대기하는 동안 영향을 받은 큐는 현재 사용할 수 없는 메시지로 채워질 수 있습니다. 이 문제점은 메시지를 가져오려는 태스크와 연결된 스토리지 및 스레드와 관련이 있습니다.

MQPUT1 호출의 사용

큐에 단일 메시지를 넣는 경우 MQPUT1 호출만 사용하십시오. 하나 이상의 메시지를 넣으려는 경우 MQOPEN 호출을 사용한 후 일련의 MQPUT 호출과 단일 MQCLOSE 호출을 사용하십시오.

사용 중인 스레드 수

애플리케이션에 많은 스레드가 필요할 수 있습니다. 각 큐 관리자 프로세스에 허용 가능한 최대 스레드가 할당됩니다. 일부 애플리케이션이 문제가 있는 경우 너무 많은 스레드를 사용하는 디자인 때문일 수 있습니다. 애플리케이션에서 이러한 가능성을 고려하고 있는지, 이러한 유형의 발생을 정지 또는 보고하기 위해 조치를 취하는지 고려하십시오. IBM i에서 허용하는 최대 스레드 수는 4,095입니다. 그러나 기본값은 64입니다. IBM MQ는 해당 프로세스에 최대 63개까지 스레드를 사용할 수 있습니다.

특정 성능 문제

이 섹션에서는 스토리지 및 성능 저하 문제점에 대해 설명합니다.

스토리지 문제점

시스템 메시지 CPF0907. Serious storage condition may exist 를 수신하는 경우 IBM MQ for IBM i 큐 관리자와 연관된 공간을 채울 수 있습니다.

애플리케이션 또는 IBM MQ for IBM i가 느리게 실행됩니까?

애플리케이션이 느리게 실행되는 경우 루프에 있거나 사용 불가능한 자원을 대기 중임을 나타낼 수 있습니다. 이 느린 실행은 성능 문제를 야기할 수도 있습니다. 아마도 시스템이 용량 한계에 다다른 상태에서 조작되고 있기 때문일 수 있습니다. 이러한 유형의 문제점은 일반적으로 최대 시스템 로드 시간인 오전 중간 및 오후 중간에 가장 심합니다. (네트워크가 둘 이상의 시간대에 걸쳐 있는 경우, 시스템 최대 로드는 다른 시간대에 발생할 수 있습니다.)

시스템 로드 따라 성능이 저하되지는 않지만 시스템에 로드가 적을 때 가끔 이런 현상이 발생하는 경우, 애플리케이션의 설계가 잘못되었기 때문입니다. 이 문제점은 특정 큐에 액세스할 때에만 발생하는 문제점일 수 있습니다.

QTOTJOB 및 QADLTOTJ는 조사할 가치가 있는 시스템 값입니다.

다음 증상은 IBM MQ for IBM i가 느리게 실행되고 있다는 것을 나타낼 수 있습니다.

- 시스템에서 MQSC 명령에 응답이 늦는 경우
- 큐 깊이에 대해 반복되는 표시가 많은 양의 큐 활동을 예상할 수 있는 애플리케이션에 대해 큐가 느리게 처리되고 있음을 나타내는 경우.
- IBM MQ 추적의 실행 여부

Linux Linux on Power Systems - Little Endian 애플리케이션에 대한 설계 고려 사항

Linux on Power Systems - Little Endian은 64비트 애플리케이션만 지원하므로 32비트 애플리케이션에 대해서는 IBM MQ에 제공되는 지원이 없습니다.

관련 개념

53 페이지의 [『IBM MQ 애플리케이션에 대한 설계 고려사항』](#)

애플리케이션에서 사용 가능한 플랫폼과 환경을 이용할 수 있는 방법을 결정한 경우 IBM MQ에서 제공한 기능의 사용 방법을 결정해야 합니다.

z/OS z/OS 애플리케이션에 대한 설계 및 성능 고려사항

애플리케이션 디자인은 성능에 영향을 미치는 가장 중요한 요소 중 하나입니다. 이 주제를 사용하여 성능에 관련된 일부 설계 요소에 대해 이해할 수 있습니다.

프로그램 설계가 잘못되면 여러 면에서 성능에 영향을 줄 수 있습니다. 프로그램이 잘 수행되는 것으로 표시될 수 있으므로 이러한 문제점을 감지하기 어려울 수 있지만 다른 태스크의 성능에 영향을 줄 수 있습니다. MQI 호출을 작성하는 프로그램에 특정한 문제점은 다음 섹션에서 표시됩니다.

애플리케이션 설계에 대한 자세한 정보는 53 페이지의 [『IBM MQ 애플리케이션에 대한 설계 고려사항』](#)의 내용을 참조하십시오.

메시지 길이의 효과

IBM MQ for z/OS를 통해 메시지를 최대 100MB까지 유지할 수 있더라도 메시지의 데이터 양은 메시지를 처리하는 애플리케이션의 성능에 영향을 줍니다. 애플리케이션이 최고의 성능을 발휘하게 하려면, 메시지의 필수 데이터만 송신하십시오. 예를 들어, 은행 계좌 출금 요청의 경우 계좌 번호와 출금 금액만 클라이언트에서 서버 애플리케이션으로 전달되면 됩니다.

메시지 지속성의 효과

지속 메시지는 로깅됩니다. 메시지 로그를 기록하면 애플리케이션의 성능을 저하시키므로 필수 데이터에만 지속 메시지를 사용하십시오. 메시지의 데이터를 제거할 수 있는 경우 큐 관리자가 중지하거나 실패하면 비지속 메시지를 사용하십시오.

지속 메시지에 대한 데이터가 버퍼를 로깅하기 위해 작성됩니다. 이러한 버퍼는 다음 경우에 로그 데이터 세트에 작성됩니다.

- 커미트가 발생하는 경우
- 메시지를 동기점 외부에서 가져오거나 넣는 경우
- WRTHRSH 버퍼가 채워진 경우

하나의 작업 단위에서 많은 메시지를 처리하는 것은 메시지가 각 작업 단위에 또는 동기점 외부에서 하나씩 처리되는 경우보다 입력/출력이 적을 수 있습니다.

특정 메시지 검색

MQGET 호출은 일반적으로 큐에서 첫 번째 메시지를 검색합니다. 메시지 디스크립터에서 메시지 및 상관 ID(**MsgId** 및 **CorrelId**)를 사용하여 특정 메시지를 지정하면 큐 관리자는 해당 메시지를 찾을 때까지 큐를 검색합니다. 이러한 방법으로 MQGET을 사용하면 애플리케이션의 성능에 영향을 미칩니다. 특정 메시지를 찾기 위해 IBM MQ에서 전체 큐를 스캔해야 할 수 있기 때문입니다.

IndexType 큐 속성을 사용하여 큐 관리자가 큐에서 MQGET 조작의 속도를 늘리는 데 사용할 수 있는 인덱스를 유지보수하도록 지정할 수 있습니다. 그러나 색인을 유지보수하게 되면 약간의 성능 저하가 발생하므로 색인을 사용해야 하는 경우에만 생성하도록 하십시오. 메시지 ID 또는 상관 ID의 색인을 빌드하도록 선택하거나 메시지가 연속으로 검색되는 큐에 대해서는 색인을 빌드하지 않도록 선택할 수 있습니다. 동일한 값을 여러 번 사용하는 것이 아니라 여러 가지 다양한 키 값을 사용해보십시오. 예를 들어, **Balance**를 세 번 사용하지 말고 **Balance1**, **Balance2** 및 **Balance3**를 사용해보십시오. 공유 큐의 경우 올바른 **IndexType**이(가) 있어야 합니다.

IndexType 큐 속성에 대한 자세한 내용은 [IndexType](#)의 내용을 참조하십시오.

색인화된 큐 사용으로 큐 관리자 재시작 시간에 영향을 미치지 않으려면, CSQ6SYSP 매크로에 QINDXBLD(NOWAIT) 매개변수를 사용하십시오. 이를 통해 큐 색인 빌드가 완료될 때까지 기다리지 않고 큐 관리자 재시작을 완료할 수 있습니다.

IndexType 속성 및 기타 오브젝트 속성에 대한 전체 설명은 [오브젝트의 속성](#)을 참조하십시오.

다른 길이의 메시지를 포함하는 큐

메시지의 예상 크기와 일치하는 버퍼 크기를 사용하여 메시지를 가져오십시오. 메시지가 너무 길다는 리턴 코드를 수신하는 경우 더 큰 버퍼를 가져오십시오. 이 방법으로 가져오기가 실패하면 리턴된 데이터 길이는 변환되지 않은 메시지 데이터의 크기입니다. MQGET 호출에 MQGMO_CONVERT를 지정하고 변환 중에 데이터를 확장하는 경우 여전히 버퍼에 맞지 않을 수 있으며 이러한 경우에는 버퍼의 크기를 더 늘려야 합니다.

버퍼 길이를 0으로 설정하여 MQGET을 발행하면 메시지 크기를 리턴한 후 애플리케이션에서 이 크기의 버퍼를 가져온 다음 Get 명령을 다시 발행할 수 있습니다. 큐를 처리하는 여러 애플리케이션이If 있는 경우 원래 애플리케이션이 가져오기를 재발행했을 때 다른 애플리케이션은 이미 메시지를 처리했을 수 있습니다. 때때로 대형 메시지가 있는 경우 이러한 메시지를 위해 대형 버퍼를 가져오고 메시지를 처리한 후에 다시 릴리스해야 할 수 있습니다. 모든 애플리케이션에 대형 버퍼가 있는 경우 가상 스토리지 문제점을 줄이도록 지원해야 합니다.

애플리케이션에서 고정 길이의 메시지를 사용할 수 없다면 이 문제에 대한 다른 해결 방법은 큐가 허용할 수 있는 최대 크기의 메시지를 찾기 위해 MQINQ 호출을 사용하는 것입니다. 그런 다음 MQGET 호출에 이 값을 사용하십시오. 큐의 최대 메시지 크기는 큐의 **MaxMsgL** 속성에 저장됩니다. 그러나 **MaxMsgL**의 값이 IBM MQ for z/OS에서 허용되는 최대값인 100MB만큼 높을 수 있기 때문에 이 메소드에서는 대량의 스토리지를 사용할 수 있습니다.

참고: 큰 메시지를 큐에 넣은 후 **MaxMsgL** 매개변수를 낮출 수 있습니다. 예를 들어, 100MB 메시지를 넣은 다음 **MaxMsgL**을(를) 50바이트로 설정할 수 있습니다. 이것은 애플리케이션에서 예상하는 것보다 더 큰 메시지를 계속 가져올 수 있다는 것을 의미합니다.

동기점 빈도

동기점 내에 커밋하지 않고 많은 MQPUT 호출을 발행하는 프로그램은 성능 문제점이 발생할 수 있습니다. 다른 태스크가 이러한 메시지를 가져오기 위해 대기하는 동안 영향을 받은 큐는 현재 사용할 수 없는 메시지로 채워질 수 있습니다. 이는 메시지를 가져오려는 태스크와 연결된 스토리지 및 스레드와 관련이 있습니다.

결과적으로 큐를 처리하는 애플리케이션이 여러 개인 경우 일반적으로 동기점에 대해

- 100개의 짧은 메시지(1KB 미만)가 있거나
- 대형 메시지(100KB)를 위한 하나의 메시지

가 있는 경우 최고의 성능이 발휘됩니다. 큐를 처리하는 애플리케이션이 단 하나인 경우 각 작업 단위에 더 많은 메시지가 있어야 합니다.

태스크가 **MAXUMSGS** 큐 관리자 속성을 사용하여 단일 복구 단위 내에서 가져오거나 넣을 수 있는 메시지 수를 제한할 수 있습니다. 이 속성에 대한 자세한 내용은 [MQSC 명령어의 ALTER QMGR](#) 명령어를 참조하십시오.

MQPUT1 호출의 장점

큐에 넣을 메시지가 하나인 경우에만 MQPUT1 호출을 사용하십시오. 하나 이상의 메시지를 넣으려는 경우 MQOPEN 호출을 사용하고 다음에 일련의 MQPUT 호출과 단일 MQCLOSE 호출을 사용하십시오.

큐 관리자가 포함할 수 있는 메시지 수

로컬 큐

큐 관리자가 보유할 수 있는 로컬 메시지의 수는 기본적으로 페이지 세트의 크기입니다. 최대 100개의 페이지 세트를 가질 수 있습니다(시스템 관련 오브젝트 및 큐에 대해 페이지 세트 0과 페이지 세트 1이 권장되는 경우라도). 확장된 형식이 포함된 페이지 세트를 사용하고 페이지 세트의 용량을 늘릴 수 있습니다.

공유 큐

공유 큐의 용량은 커플링 키금(CF)의 크기에 따라 달라집니다. IBM MQ는 기본적인 스토리지 단위가 입력 항목 및 요소인 CF 목록 구조를 사용합니다. 각 메시지는 1개 입력 항목 및 연관된 MQMD와 다른 메시지 데이터를 포함하는 여러 요소로 저장됩니다. 단일 메시지에서 이용하는 요소의 수는 메시지의 크기 및 CFLEVEL(5)의 경우 MQPUT 시간에 요력이 발생하는 오프로드 규칙에 따라 달라집니다. 메시지 데이터가 Db2 또는 SMDS에 대해 오프로드되는 경우 더 적은 요소가 필요하게 됩니다. 메시지가 오프로드된 경우 메시지 데이터 액세스가 더 느려집니다. 메시지 오프로드와 연관된 성능 및 CPU 오버헤드의 추가적인 비교는 성능 Supportpac MP1H를 참조하십시오.

성능에 영향을 미치는 항목

성능은 얼마나 빠르게 메시지를 처리할 수 있는지를 의미할 수 있으며 메시지당 필요한 CPU의 양을 의미할 수도 있습니다.

메시지를 얼마나 빠르게 처리할 수 있는지에 영향을 미치는 항목

지속 메시지의 경우 가장 큰 영향을 주는 것은 로그 데이터 세트의 속도입니다. 로그 데이터 세트의 속도는 데이터 세트가 있는 DASD에 따라 달라집니다. 따라서 경합을 줄이기 위해 적게 사용되는 볼륨에 로그 데이터 세트를 넣을 때는 주의해야 합니다. MQ 로그를 스트라이핑하면 입/출력당 작성된 페이지가 여러 개인 경우 로그 성능이 개선됩니다. zHPF(Z High Performance Fibre) 연결은 또한 입/출력 서브시스템이 사용 중일 때 입/출력 응답 시간에 상당한 성능을 제공합니다.

메시지를 가져오거나 넣는 요청이 있는 경우 큐의 무결성을 지속하는 요청 중에는 큐에 대한 액세스가 잠깁니다. 계획의 목적으로 전체 요청에 대해 잠긴 큐를 고려해보십시오. 넣기 위한 시간이 100 마이크로초이며 초당 10,000개 이상의 요청이 있는 경우 지연이 발생할 수 있습니다. 실제로는 이보다 더 많이 달성할 수 있지만 좋은 일반 규칙입니다. 성능을 향상시키기 위해 다른 큐를 사용할 수 있습니다.

가능한 이유는 다음이 될 수 있습니다.

- 모든 CICS 트랜잭션에서 사용하는 공통 응답 큐를 사용합니다.
- 각 CICS 트랜잭션이 큐에 고유 응답을 지정합니다.
- CICS 영역을 위한 큐에 대한 응답 및 CICS 영역의 모든 트랜잭션은 이 큐를 사용합니다.

응답은 초당 요청 수 및 요청의 응답 시간에 따라 달라집니다.

메시지가 페이지 세트에서 읽혀져야 하는 경우 메시지가 버퍼 풀에 있는 경우와 비교하면 더 느려집니다. 버퍼 풀에 적절한 것보다 더 많은 메시지가 있는 경우에는 디스크로 메시지가 유출됩니다. 따라서 버퍼 풀이 일시적인 메시지에 대해 충분히 크지 확인해야 합니다. 상당한 시간이 흐른 후에 처리하는 메시지가 있는 경우 이러한 메시지는 디스크로 유출되기 쉬우므로 버퍼 풀에 있는 경우보다 메시지에 대한 가져오기가 더 느릴 것으로 예상해야 합니다.

공유 큐의 경우 메시지의 속도는 커플링 기능(CF)의 속도에 따라 달라집니다. 실제 프로세서 내의 CF는 외부 CF보다 더 빠를 수 있습니다. CF 응답 시간은 CF 기능이 얼마나 많이 사용되는지에 따라 달라집니다. 예를 들어, Hursley 시스템에서 CF를 17% 사용 중이었을 때 응답 시간은 14마이크로초였습니다. CF가 95%까지 사용되었을 때 응답 시간은 45마이크로초였습니다.

MQ 요청에서 많은 CPU를 사용하는 경우 메시지가 얼마나 빨리 처리되는지에 영향을 줄 수 있습니다. 논리 파티션(LPAR)이 CPU에 대해 제한되면 애플리케이션이 CPU를 대기하면서 지연이 발생하기 때문입니다.

메시지당 CPU 양

일반적으로 더 큰 메시지는 더 많은 CPU를 사용하므로 가능하다면 대형(x MB) 메시지는 피하십시오.

큐에서 특정 메시지를 가져오는 경우 큐 관리자가 직접 메시지로 이동할 수 있도록 큐를 색인화해야 합니다 (그러면 큐의 전체 스캔을 잠재적으로 방지합니다.). 큐가 색인화되지 않는 경우라면 큐는 처음부터 스캔되며 메시지를 찾습니다. 큐에 1,000개의 메시지가 있는 경우 1,000개 메시지 모두 스캔해야 할 수 있습니다. 결과는 불필요하게 많은 CPU가 사용됩니다.

TLS를 사용하는 채널은 메시지의 암호화로 인해 추가적인 비용이 발생합니다.

MQ V7에서 **CORRELID** 또는 **MSGID** 외에 선택기 문자열로 메시지를 선택할 수 있습니다. 모든 메시지 내용을 살펴봐야 하므로 큐에 메시지가 많은 경우 비용이 많이 발생합니다.

PUT1 PUT1보다 OPEN PUT PUT CLOSE를 수행하는 애플리케이션에 더 효율적입니다.

CICS에서 트리거

트리거된 큐에 대한 메시지의 메시지 도착 비율이 낮은 경우 트리거를 먼저 사용하는 것이 효율적입니다. 메시지 도착 비율이 초당 10개 메시지보다 많은 경우 첫 번째 트랜잭션을 트리거한 후 트랜잭션에서 메시지를 처리하고 다음 메시지를 가져오는 등을 수행하게 하는 것이 더 효율적입니다. 메시지가 단기간에 도착하지 않는다면(0.1과 1초 사이인 경우) 트랜잭션은 종료됩니다. 높은 처리량에서 메시지를 처리하고 메시지 빌드를 방지하기 위해 실행 중인 여러 트랜잭션이 필요할 수 있습니다. 생성된 모든 트리거 메시지의 경우 트리거 메시지의 넣기 및 가져오기가 필요하며 실제로는 메시지 비용이 두 배가 됩니다.

지원되는 연결 또는 동시 사용자 수

각 연결은 큐 관리자 내에서 가상 스토리지를 사용하므로 동시 사용자가 많을수록 더 많은 스토리지를 사용합니다. 매우 큰 버퍼 풀과 많은 사용자가 필요한 경우에는 가상 스토리지를 제한할 수 있으며 버퍼 풀의 크기를 줄여야 할 수 있습니다.

보안을 사용 중인 경우 큐 관리자는 긴 기간 동안 큐 관리자 내에서 정보를 캐싱합니다. 큐 관리자 내에서 사용되는 가상 스토리지의 양에 영향을 줍니다.

CHINIT에서는 최대 약 10,000개의 연결을 지원할 수 있습니다. 가상 스토리지에서 제한합니다. 연결에서 추가 스토리지를 사용하는 경우(예: TLS 사용) 연결당 스토리지가 증가하므로 **CHINIT**에서 더 적은 수의 연결을 지원할 수 있습니다. 큰 메시지를 처리하는 경우에는 **CHINIT**의 버퍼에 대한 추가 스토리지가 필요하므로 **CHINIT**에서 더 적은 수의 메시지를 지원할 수 있습니다.

리모트 큐 관리자에 대한 연결이 클라이언트 연결보다 더 효율적입니다. 예를 들어, 모든 MQ 클라이언트 요청은 두 개의 네트워크 플로우가 필요합니다(요청에 하나 응답에 하나). 리모트 큐 관리자에 대한 채널을 사용하면 응답이 돌아오기 전에 네트워크에서 30개의 송신이 처리될 수 있습니다. 대형 클라이언트 네트워크를 고려 중이라면 분배 상자에서 집중기 큐 관리자를 사용하는 것이 보다 효율적일 수 있으며 집중기에 하나의 채널이 들어오고 나가게 됩니다.

성능에 영향을 주는 기타 항목

로그 데이터 세트 크기가 적어도 1,000개 실린더여야 합니다. 로그가 이보다 작은 경우 체크포인트 활동이 너무 자주 발생할 수 있습니다. 사용 중인 시스템에서 체크포인트는 일반적으로 15분마다 또는 더 길어야 하며 처리량이 매우 높은 시점에는 이보다 자주일 수 있습니다. 체크포인트가 발생하는 경우 버퍼 풀이 스캔되고 '이전' 메시지 및 변경된 페이지는 디스크에 작성됩니다. 체크포인트가 너무 자주 발생하면 성능에 영향을 줄 수 있습니다. LOGLOAD의 값도 체크포인트 빈도에 영향을 줄 수 있습니다. 큐 관리자가 비정상적으로 종료된 후에는 재시작 시 이전 3개 체크포인트까지 읽어야 할 수 있습니다. 가장 적절한 체크포인트 간격은 체크포인트가 사용될 때의 활동과 큐 관리자가 재시작될 때 읽어야 하는 로그 데이터의 양 사이에 균형을 이루는 것입니다.

채널을 시작하면 상당한 오버헤드가 발생합니다. 일반적으로 채널을 자주 시작하고 중지하는 것보다 채널을 시작하고 연결된 상태로 두는 것이 좋습니다.

관련 정보

[MP1K: IBM MQ for z/OS 9.0 성능 보고서](#)

▶ z/OS IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션

이 정보는 IBM MQ을(를) 사용하여 IMS 애플리케이션을 작성하는 데 도움이 됩니다.

- IMS 애플리케이션에서 동기점 및 MQI 호출을 사용하려면, [71 페이지의 『사용하여 애플리케이션 IMS 작성 IBM MQ』](#)의 내용을 참조하십시오.
- IBM MQ - IMS 브릿지를 사용하는 애플리케이션을 쓰려면, [75 페이지의 『IMS 브릿지 애플리케이션 작성』](#)의 내용을 참조하십시오.

다음 링크를 사용하여 IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션에 대해 자세히 알아보십시오.

- [71 페이지의 『사용하여 애플리케이션 IMS 작성 IBM MQ』](#)
- [75 페이지의 『IMS 브릿지 애플리케이션 작성』](#)

관련 개념

[744 페이지의 『MQI\(Message Queue Interface\) 개요』](#)

MQI(Message Queue Interface) 컴포넌트에 대해 학습합니다.

[756 페이지의 『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』](#)

IBM MQ 프로그래밍 서비스를 사용하려면 프로그램을 큐 관리자에 연결해야 합니다. 이 정보를 사용하여 큐 관리자에 연결하고 큐 관리자에서 연결을 끊는 방법에 대해 알아보십시오.

[763 페이지의 『오브젝트 열기 및 닫기』](#)

이 정보는 IBM MQ 오브젝트의 열기 및 닫기 조작에 대한 통찰력을 제공합니다.

[772 페이지의 『큐에 메시지 넣기』](#)

이 정보를 사용하여 메시지를 큐에 넣는 방법에 대해 알아보십시오.

[786 페이지의 『큐에서 메시지 가져오기』](#)

이 정보를 사용하여 큐에서 메시지를 가져오는 방법에 대해 알아보십시오.

[861 페이지의 『오브젝트 속성 조회 및 설정』](#)

속성은 IBM MQ 오브젝트의 특성을 정의하는 특성입니다.

[863 페이지의 『작업 단위 커밋 및 백아웃』](#)

이 정보는 작업 단위에서 발생한 복구 가능한 가져오기 및 넣기 조작을 커밋하고 백아웃하는 방법을 설명합니다.

[873 페이지의 『트리거를 사용한 IBM MQ 애플리케이션 시작』](#)

트리거와 트리거를 사용하여 IBM MQ 애플리케이션을 시작하는 방법에 대해 학습하십시오.

[891 페이지의 『MQI 및 클러스터에 대한 작업』](#)

클러스터와 관련된 호출 및 리턴 코드에 대한 특수 옵션이 있습니다.

[895 페이지의 『IBM MQ for z/OS의 애플리케이션 사용 및 작성』](#)

IBM MQ for z/OS 애플리케이션은 여러 다른 환경에서 실행되는 프로그램으로 구성할 수 있습니다. 이는 둘 이상의 환경에서 사용 가능한 기능을 활용할 수 있음을 의미합니다.

▶ z/OS 사용하여 애플리케이션 IMS 작성 IBM MQ

애플리케이션에서 IMS를 IBM MQ 사용할 때 고려해야 할 추가 사항이 있습니다. 여기에는 어떤 MQ API 호출을 사용할 수 있는지, 그리고 동기화 지점(syncpoint)을 위한 메커니즘이 포함됩니다.

다음 링크를 통해 에서 애플리케이션 IMS 작성에 대해 IBM MQ for z/OS 자세히 알아보세요:

- [72 페이지의 『IMS 애플리케이션의 동기점』](#)
- [72 페이지의 『IMS 애플리케이션에서 MQI 호출』](#)

제한사항

어댑터를 IMS 사용하는 애플리케이션이 호출할 수 있는 IBM MQ API에는 제한 사항이 있습니다.

다음 IBM MQ API 호출은 IMS 어댑터를 사용하는 애플리케이션 내에서 지원되지 않습니다.

- MQCB
- MQCB_FUNCTION
- MQCTL

관련 개념

75 페이지의 『IMS 브릿지 애플리케이션 작성』

이 주제에는 IBM MQ - IMS 브릿지를 사용하기 위해 애플리케이션을 작성하는 데 대한 정보가 포함되어 있습니다.

z/OS IMS 애플리케이션의 동기점

IMS 애플리케이션에서 GU(get unique)부터 IOPCB 및 CHKP(checkpoint)와 같은 IMS 호출을 사용하여 동기점을 설정합니다.

이전 체크포인트 이후로 모든 변경을 백아웃하려면 IMS ROLB(롤백) 호출을 사용할 수 있습니다. 자세한 내용은 문서의 IMS [ROLB 호출](#)을 참조하십시오.

큐 관리자는 2단계 커밋 프로토콜의 참가자입니다. IMS 동기점 관리자는 코디네이터입니다.

열려 있는 모든 핸들은 (배치 또는 비메시지 구동 BMP 환경을 제외한) 동기점에서 IMS 어댑터에 의해 처리완료 됩니다. 이는 다른 사용자가 다음 작업 단위를 시작할 수 있고 MQPUT 또는 MQGET 호출 시가 아니라 MQCONN, MQCONNx 및 MQOPEN 호출 시에 IBM MQ 보안 검사가 수행되기 때문입니다.

그러나 입력 대기(WFI) 또는 의사 입력 대기(PWFI) 환경에서 IMS는 다음 메시지가 도착하거나 QC 상태 코드가 애플리케이션으로 리턴될 때까지 핸들을 닫도록 IBM MQ에 알리지 않습니다. 애플리케이션이 IMS 리전에서 대기 중이고 이러한 핸들 중 하나라도 트리거된 큐에 속하면 해당 큐가 열려 있기 때문에 트리거가 발생하지 않습니다. 이러한 이유로 WFI 또는 PWFI 환경에서 실행 중인 애플리케이션은 다음 메시지에 대해 GU부터 IOPCB를 수행하기 전에 큐 핸들을 명확하게 MQCLOSE해야 합니다.

IMS 애플리케이션(BMP 또는 MPP)이 MQDISC 호출을 발행하면, 열린 큐는 처리완료되지만 암시적 동기점이 사용되지 않습니다. 애플리케이션이 정상적으로 종료되면 열린 큐는 모두 닫히고 암시적 커밋이 발생합니다. 애플리케이션이 비정상적으로 종료되는 경우에는 열린 큐가 모두 닫히고 암시적 백아웃이 발생합니다.

z/OS IMS 애플리케이션에서 MQI 호출

이 정보를 사용하여 서버 애플리케이션 및 조회 애플리케이션에서 MQI 호출의 사용에 대해 알아볼 수 있습니다.

이 절에서는 다음 유형의 IMS 애플리케이션에서 MQI 호출의 사용에 대해 설명합니다.

- 72 페이지의 『서버 애플리케이션』
- 74 페이지의 『조회 애플리케이션』

서버 애플리케이션

다음은 MQI 서버 애플리케이션 모델에 대한 개요입니다.

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

```

·End do
·Close queue/Disconnect
·END

```

샘플 프로그램 CSQ4ICB3은 C/370에서 이 모델을 사용하는 BMP의 구현을 표시합니다. 프로그램은 먼저 IMS 그리고 IBM MQ와의 통신을 설정합니다.

```

main()
----
Call InitIMS
If IMS initialization successful
Call InitMQM
If IBM MQ initialization successful
Call ProcessRequests
Call EndMQM
End-if
End-if

Return

```

IMS 초기화는 프로그램이 메시지 구동 또는 배치 지향 BMP로 호출되었는지 판별하고 IBM MQ 큐 관리자 연결 및 큐 핸들을 적절하게 제어합니다.

```

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

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

Return to calling function

```

IBM MQ 초기화는 큐 관리자에 연결하여 큐를 엽니다. 메시지 구동 BMP에서 각 IMS 동기점이 사용된 후에 초기화가 호출됩니다. 배치 지향 BMP의 경우 프로그램 시동 중에만 호출됩니다.

```

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

```

```

Build 'connect failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function

```

MPP에서 서버 모델의 구현은 MPP에서 호출당 단일 작업 단위를 처리한다는 사실에 의해 영향을 받습니다. 그 이유는 동기점(GU)이 사용되면 연결 및 큐 핸들이 닫히고 다음 IMS 메시지가 전달되기 때문입니다. 이 제한사항은 다음 중 하나로 부분적으로 해결할 수 있습니다.

- **단일 작업 단위 내에서 여러 메시지 처리**

다음에 포함됩니다.

- 메시지 읽기
- 필수 업데이트 처리
- 응답 넣기

모든 메시지가 처리되거나 메시지 최대 세트 수가 처리될 때까지 동기점이 사용되는 시간에 반복됩니다.

애플리케이션의 특정 유형만(예를 들어, 단순 데이터베이스 업데이트 또는 조회) 이 방법으로 접근할 수 있습니다. MQI 응답 메시지를 핸들링되는 MQI 메시지의 진원지 권한을 사용하여 넣을 수 있더라도 IMS 자원 업데이트의 보안 관련 내용을 주의깊게 고려해야 합니다.

- **MPP의 호출당 하나의 메시지를 처리하고 사용 가능한 모든 메시지를 처리하기 위해 MPP의 다중 스케줄링을 보장합니다.**

IBM MQ IMS 트리거 모니터 프로그램(CSQQTRMN)을 사용하여 IBM MQ 큐에 메시지가 있고 이를 제공하는 애플리케이션이 없는 경우 MPP 트랜잭션을 스케줄링하십시오.

트리거 모니터가 MPP를 시작하는 경우 다음 COBOL 코드 추출에 표시된 대로 큐 관리자 이름 및 큐 이름이 프로그램에 전달됩니다.

```

* Data definition extract
01 WS-INPUT-MSG.
05 IN-LL1          PIC S9(3) COMP.
05 IN-ZZ1          PIC S9(3) COMP.
05 WS-STRINGPARM   PIC X(1000).
01 TRIGGER-MESSAGE.
COPY CMQTM2L.
*
* Code extract
GU-IOPCB SECTION.
MOVE SPACES TO WS-STRINGPARM.
CALL 'CBLTDLI' USING GU,
IOPCB,
WS-INPUT-MSG.
IF IOPCB-STATUS = SPACES
MOVE WS-STRINGPARM TO MQTMC.
* ELSE handle error
*
* Now use the queue manager and queue names passed
DISPLAY 'MQTMC-QMGRNAME' = '
MQTMC-QMGRNAME OF MQTMC '='.
DISPLAY 'MQTMC-QNAME' = '
MQTMC-QNAME OF MQTMC '='.

```

CSQQTRMN을 사용하여 BMP를 트리거할 수 없더라도 장기 실행 태스크가 예상되는 서버 모델은 배치 처리 영역에서 제대로 지원됩니다.

조회 애플리케이션

조회 또는 업데이트를 시작하는 일반적인 IBM MQ 애플리케이션은 다음과 같이 작동합니다.

- 사용자로부터 데이터 수집
- 하나 이상의 IBM MQ 메시지 넣기
- 응답 메시지 가져오기(대기해야 할 수 있음)
- 사용자에게 응답 제공

IBM MQ 큐로 넣어지는 메시지는 커밋될 때까지 다른 IBM MQ 애플리케이션에서 사용할 수 없기 때문에 동기점 밖으로 넘겨나 IMS 애플리케이션을 두 개의 트랜잭션으로 분할해야 합니다.

조회에 단일 메시지 넣기가 포함된 경우 동기점 없음 옵션을 사용할 수 있습니다. 그러나 조회가 좀 더 복잡하거나 자원 업데이트가 관련된 경우에는 실패가 발생하거나 동기점 조정을 사용하지 않으면 일관성 문제점이 발생할 수 있습니다.

이 문제를 해결하려면 프로그램 간 메시지 스위치를 사용하여 MQI MPP 트랜잭션을 호출로 분할할 IMS 수 있습니다. 이에 대한 자세한 내용은 [여기]를 참조하십시오 [IMS 시스템 간 통신 \(ISC\)](#). 조회 프로그램이 MPP에서 구현될 수 있습니다.

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

IMS 브릿지 애플리케이션 작성

이 주제에는 IBM MQ - IMS 브릿지를 사용하기 위해 애플리케이션을 작성하는 데 대한 정보가 포함되어 있습니다.

IBM MQ - IMS 브릿지에 대한 정보는 [IMS 브릿지](#)를 참조하십시오.

다음 링크를 사용하여 IBM MQ for z/OS에서 IMS 브릿지 애플리케이션 작성에 대해 자세히 알아보십시오.

- 75 페이지의 『IMS 브릿지에서 메시지를 처리하는 방법』
- 915 페이지의 『트랜잭션 프로그램을 IMS 작성하여 IBM MQ』

관련 개념

71 페이지의 『사용하여 애플리케이션 IMS 작성 IBM MQ』

애플리케이션에서 IMS 를 IBM MQ 사용할 때 고려해야 할 추가 사항이 있습니다. 여기에는 어떤 MQ API 호출을 사용할 수 있는지, 그리고 동기화 지점(syncpoint)을 위한 메커니즘이 포함됩니다.

IMS 브릿지에서 메시지를 처리하는 방법

IBM MQ - IMS 브릿지를 사용하여 IMS 애플리케이션으로 메시지를 전송하는 경우 메시지를 특별한 형식으로 생성해야 합니다.

또한 대상 IMS 시스템의 XCF 그룹 및 멤버 이름을 지정하는 저장 클래스로 정의된 큐에 IBM MQ 메시지를 배치해야 합니다. MQ-IMS 브릿지 큐 또는 단순히 브릿지 큐로 알려져 있습니다.

IBM MQ-IMS 브릿지는 QSGDISP(QMGR)로 정의된 경우 또는 QSGDISP(SHARED)를 NOSHARE 옵션과 함께 사용하여 정의된 경우 브릿지 큐에 대한 독점 입력 액세스(MQOO_INPUT_EXCLUSIVE)가 필요합니다.

사용자는 메시지를 IMS 애플리케이션에 보내기 전에 IMS에 사인온할 필요가 없습니다. MQMD 구조의 *UserIdentifier* 필드에서 사용자 ID는 보안 검사에 사용됩니다. 검사 레벨은 IBM MQ가 IMS에 연결할 때 판별되고 [IMS 브릿지에 대한 애플리케이션 액세스 제어](#)에서 설명됩니다. 의사 사인온을 구현되도록 할 수 있습니다.

IBM MQ - IMS 브릿지는 다음 유형의 메시지를 허용합니다.

- IMS 트랜잭션 데이터 및 MQIIH 구조를 포함하는 메시지(MQIIH에서 설명됨):

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

참고:

1. 각괄호 []는 선택적 다중 세그먼트를 나타냅니다.
 2. MQMD 구조의 *Format* 필드를 MQFMT_IMS로 설정하여 MQIIH 구조를 사용합니다.
- IMS 트랜잭션 데이터를 포함하지만 MQIIH 구조는 없는 메시지:

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

IBM MQ는 메시지 데이터의 유효성을 검증하여 LL 바이트의 합에 MQIIH(있는 경우)의 길이를 더하면 메시지 길이와 동일한지 확인합니다.

IBM MQ - IMS 브릿지가 브릿지 큐에서 메시지를 가져오는 경우 다음과 같이 메시지를 처리합니다.

- 메시지에 MQIIH 구조가 포함되어 있는 경우 브릿지는 MQIIH를 확인하고(MQIIH 참조) OTMA 헤더를 빌드하고 메시지를 IMS에 송신합니다. 트랜잭션 코드가 입력 메시지에 지정됩니다. 코드가 LTERM인 경우 IMS는 DFS1288E 메시지로 응답합니다. 트랜잭션 코드가 명령을 나타내는 경우 IMS는 명령을 실행합니다. 그렇지 않은 경우 메시지는 트랜잭션을 위해 IMS의 큐에 넣어집니다.
- 메시지에 IMS 트랜잭션 데이터가 포함되어 있지만 MQIIH 구조는 없는 경우 IMS 브릿지는 다음을 가정합니다.
 - 트랜잭션 코드는 5-12바이트의 사용자 데이터입니다.
 - 트랜잭션은 비대화식 모드입니다.
 - 트랜잭션은 커밋 모드 0(commit-then-send)입니다.
 - MQMD에서 *Format*은 MFSMapName(입력에서)으로 사용됩니다.
 - 보안 모드는 MQISS_CHECK입니다.

응답 메시지도 MQIIH 구조체 없이 생성되며, MQMD의 값은 *Format* 출력 IMS 메시지의 MFSMapName MQMD에서 가져옵니다.

IBM MQ - IMS 브릿지는 각 IBM MQ 큐에 대해 하나 이상의 Tpipe를 사용합니다.

- 동기화된 Tpipe는 커밋 모드 0(COMMIT_THEN_SEND)을 사용하는 모든 메시지에 사용됩니다(IMS /DIS TMEMBER client TPIPE xxxx 명령의 상태 필드에서 SYN과 함께 표시).
- 비동기화된 Tpipe는 커밋 모드 1(SEND_THEN_COMMIT)을 사용하는 모든 메시지에 사용됩니다.

Tpipe는 처음 사용될 때 IBM MQ에 의해 작성됩니다. 비동기화된 Tpipe는 IMS가 재시작될 때까지 존재합니다. 동기화된 Tpipe는 IMS가 콜드 스타트될 때까지 존재합니다. 이 Tpipe를 사용자가 삭제할 수 없습니다.

IBM MQ - IMS 브릿지에서 메시지를 처리하는 방법에 대한 자세한 정보는 다음 주제를 참조하십시오.

- [77 페이지의 『IBM MQ 메시지를 IMS 트랜잭션 유형에 매핑』](#)
- [77 페이지의 『메시지를 IMS 큐에 넣을 수 없는 경우』](#)
- [77 페이지의 『IMS 브릿지 피드백 코드』](#)
- [78 페이지의 『IMS 브릿지의 메시지에 있는 MQMD 필드』](#)
- [79 페이지의 『IMS 브릿지의 메시지에 있는 MQIIH 필드』](#)
- [80 페이지의 『IMS의 응답 메시지』](#)
- [80 페이지의 『IMS 트랜잭션에서 대체 응답 PCB 사용』](#)
- [80 페이지의 『IMS의 요청되지 않은 메시지 송신』](#)
- [80 페이지의 『메시지 세그먼트화』](#)
- [81 페이지의 『교량과의 통신 IMS 메시지 데이터 변환』](#)

관련 개념

[915 페이지의 『트랜잭션 프로그램을 IMS 작성하여 IBM MQ』](#)

트랜잭션을 IMS 처리하기 위해 필요한 코딩은 해당 IMS 트랜잭션이 요구하는 IBM MQ 메시지 형식과 반환 가능한 응답 범위에 따라 달라집니다. 그러나 애플리케이션이 IMS 화면 형식 정보를 핸들링할 때 몇 가지 고려할 사항이 있습니다.

z/OS IBM MQ 메시지를 IMS 트랜잭션 유형에 맵핑
메시지와 IBM MQ 트랜잭션 IMS 유형 간의 매핑을 설명하는 테이블.

표 4. 메시지가 IBM MQ 트랜잭션 IMS 유형에 어떻게 매핑되는가		
IBM MQ 메시지 유형	Commit-then-send(모드 0) - 동기화된 IMS Tpipe 사용	Send-then-commit(모드 1) - 동기화되지 않은 IMS Tpipe 사용
지속적 IBM MQ 메시지	- 복구 가능한 전체 기능 트랜잭션 - 복구 불가능한 트랜잭션은 IMS에 의해 거부됩니다.	- 빠른 경로 트랜잭션 - 대화식 트랜잭션 - 전체 기능 트랜잭션
비지속적 IBM MQ 메시지	- 복구할 수 없는 전체 기능 트랜잭션 - 복구 가능한 트랜잭션은 IMS V8, APAR PQ61404 및 IMS의 모든 이후 버전에서 허용됩니다.	- 빠른 경로 트랜잭션 - 대화식 트랜잭션 - 전체 기능 트랜잭션

참고: IMS 명령은 커밋 모드 0의 지속적 IBM MQ 메시지를 사용할 수 없습니다. 자세한 내용은 [커밋 모드 \(commitMode\)](#)를 참조하십시오.

z/OS 메시지를 IMS 큐에 넣을 수 없는 경우
메시지를 IMS 큐에 넣을 수 없는 경우 수행할 조치에 대해 알아보십시오.

메시지를 IMS 큐에 넣을 수 없는 경우 IBM MQ에서 다음 조치를 수행합니다.

- 메시지가 올바르지 않기 때문에 IMS에 메시지를 넣을 수 없는 경우 메시지는 데드-레터 큐에 넣어지고 메시지가 시스템 콘솔로 송신됩니다.
- 메시지가 올바르지만 IMS에 의해 거부되고, IBM MQ에서 오류 메시지를 시스템 콘솔로 송신하는 경우 메시지는 IMS 감지 코드를 포함하며 IBM MQ 메시지가 데드-레터 큐에 넣어집니다. IMS 감지 코드가 001A인 경우 IMS는 실패의 이유가 포함되어 있는 IBM MQ 메시지를 응답 큐에 송신합니다.

참고: 앞서 나열된 환경에서 IBM MQ가 어떠한 이유로 인해 메시지를 데드-레터 큐에 넣을 수 없는 경우 메시지는 원래 IBM MQ 큐로 리턴됩니다. 오류 메시지가 시스템 콘솔에 송신되고 해당 큐로부터 더 이상 추가 메시지는 송신되지 않습니다.

메시지를 다시 송신하려면 다음 중 **하나**를 수행하십시오.

- 큐에 해당하는 IMS에서 Tpipe를 중지하고 다시 시작합니다.
- 큐를 GET(DISABLED)로 변경한 후 다시 GET(ENABLED)로 변경합니다.
- IMS 또는 OTMA를 중지하고 다시 시작합니다.
- IBM MQ 서브시스템을 중지하고 다시 시작합니다.
- 메시지 오류 이외의 다른 이유로 IMS에서 메시지를 거부하는 경우 IBM MQ 메시지가 원래 큐로 리턴되고 IBM MQ는 큐 처리를 중지하고 오류 메시지가 시스템 콘솔에 송신됩니다.

예외 보고 메시지가 필요한 경우 브릿지는 진원지 권한을 사용하여 응답 대상 큐에 메시지를 넣습니다. 메시지를 큐에 넣을 수 없는 경우에는 브릿지 권한을 사용하여 데드-레터 큐에 보고 메시지를 넣습니다. DLQ에 넣을 수 없다면 제거됩니다.

z/OS IMS 브릿지 피드백 코드

IMS 감지 코드는 일반적으로 CSQ2001I와 같은 IBM MQ 콘솔 메시지(예: 감지 코드 0x001F)에서 16진 형식의 출력입니다. 데드-레터 큐로 넣기되는 메시지의 데드 레터 헤더에 표시된 대로 IBM MQ 피드백 코드는 10진수입니다.

IMS 브릿지 피드백 코드는 NACK 감지 코드 0x001A에 대해 301에서 399 또는 600에서 855 범위에 있습니다. 다음과 같이 IMS-OTMA 감지 코드에서 맵핑됩니다.

1. IMS-OTMA 감지 코드는 16진수에서 10진수로 변환됩니다.
2. 1번 계산 결과에 300을 더하여 코드를 IBM MQFeedback 얻습니다.
3. IMS-OTMA 감지 코드 0x001A, 소수점 26은 특수한 경우입니다. 600-855 범위의 Feedback 코드가 생성됩니다.
 - a. IMS-OTMA 이유 코드가 16진수에서 10진수로 변환됩니다.
 - b. a 의 계산 결과에 600을 더하여 피드백 IBM MQ 코드를 얻습니다.

IMS-OTMA 감지 코드에 대한 정보는 [NAK 메시지의 OTMA 감지 코드를 참조하십시오.](#)

z/OS IMS 브릿지의 메시지에 있는 MQMD 필드
IMS 브릿지의 메시지에 있는 MQMD 필드에 대해 알아보십시오.

원래 메시지의 MQMD는 OTMA 헤더의 사용자 데이터 섹션에 있는 IMS에 의해 이동됩니다. 메시지가 IMS에서 생성되는 경우 IMS 목적지 해석 엑시트에 의해 빌드됩니다. IMS에서 수신된 메시지의 MQMD는 다음과 같이 빌드됩니다.

StrucID
"MD"

버전
MQMD_VERSION_1

보고서
MQRO_NONE

MsgType
MQMT_REPLY

만기
MQIIH_PASS_EXPIRATION이 MQIIH의 플래그 필드에 설정되어 있는 경우 이 필드에는 남은 만기 시간이 포함됩니다. 그 밖에 MQEI_UNLIMITED로 설정됩니다.

Feedback
MQFB_NONE

Encoding
MQENC.Native(z/OS 시스템의 인코딩)

CodedCharSetId
MQCCSI_Q_MGR(z/OS 시스템의 CodedCharSetId)

형식
MQFMT_IMS(입력 메시지의 MQMD.Format이 MQFMT_IMS인 경우), 그렇지 않으면 IOPCB.MODNAME

Priority
입력 메시지의 MQMD.Priority

Persistence
커밋 모드에 따라 달라집니다. CM-1인 경우 입력 메시지의 MQMD.Persistence이며 CM-0인 경우 지속성은 IMS 메시지의 복구 가능성과 일치합니다.

MsgId
MQMD.MsgId(MQRO_PASS_MSG_ID인 경우), 그렇지 않으면 새 MsgId(기본값)

CorrelId
MQRO_PASS_CORREL_ID인 경우 입력 메시지의 MQMD.CorrelId. 그렇지 않으면 입력 메시지의 MQMD.MsgId(기본값)

BackoutCount
0

ReplyToQ
공백

ReplyToQMGr

공백(MQPUT 동안 큐 관리자에 의해 로컬 qmgr 이름으로 설정)

UserIdentifier

입력 메시지의 MQMD.UserIdentifier

AccountingToken

입력 메시지의 MQMD.AccountingToken

ApplIdentityData

입력 메시지의 MQMD.ApplIdentityData

PutApplType

오류가 없는 경우 MQAT_XCF이며 그렇지 않으면 MQAT_BRIDGE

PutApplName

오류가 없는 경우 <XCFgroupName><XCFmemberName>이며 그렇지 않으면 QMGR 이름입니다.

PutDate

메시지를 넣은 날짜

PutTime

메시지를 넣은 시간

ApplOriginData

공백

 **IMS** 브릿지의 메시지에 있는 MQIIH 필드
IMS 브릿지의 메시지에 있는 MQIIH 필드에 대해 알아보십시오.
IMS에서 수신된 메시지의 MQIIH는 다음과 같이 빌드됩니다.

StrucId

"IIH"

버전

1

StrucLength

84

Encoding

MQENC_NATIVE

CodedCharSetId

MQCCSI_Q_MGR

형식

MQIIH.ReplyToFormat이 공백이 아닌 경우 입력 메시지의 MQIIH.ReplyToFormat이며 그렇지 않으면 IOPCB.MODNAME.

플래그

0

LTermOverride

OTMA 헤더의 LTERM 이름(Tpipe)

MFSMapName

OTMA 헤더의 맵 이름

ReplyToFormat

공백

Authenticator

응답 메시지가 MQ-IMS 브릿지 큐로 놓여지는 경우 입력 메시지의 MQIIH.Authenticator이며 그렇지 않으면 공백입니다.

TranInstanceId

대화 중인 경우 OTMA 헤더의 대화 ID/서버 토큰입니다. V14 이전의 IMS 버전에서는 대화 중이 아닌 경우 이 필드가 항상 널입니다. IMS V14 이상에서는 IMS가 대화 중이 아니더라도 이 필드를 설정할 수 있습니다.

TranState

대화 중인 경우 "C"이며 그렇지 않은 경우 공백입니다.

CommitMode

OTMA 헤더의 커미트 모드("0" 또는 "1")

SecurityScope

공백

예약됨

공백

▶ z/OS IMS의 응답 메시지

IMS 트랜잭션이 IOPCB에 ISRT되면 원래 LTERM 또는 TPIPE로 메시지가 다시 라우팅됩니다.

응답 메시지로 IBM MQ에 표시됩니다. IMS로부터의 응답 메시지가 원래 메시지에 지정된 응답 대상 큐에 놓여집니다. 메시지를 응답 대상 큐에 놓을 수 없는 경우 브릿지 권한을 사용하여 데드-레터 큐에 메시지를 넣습니다. 메시지를 데드-레터 큐에 넣을 수 없는 경우에는 부정 수신확인이 IMS로 송신되어 메시지를 수신할 수 없다고 전달합니다. 메시지에 대한 책임은 IMS로 리턴됩니다. 커미트 모드 0을 사용 중인 경우 해당 Tpipe의 메시지가 브릿지로 송신되지 않으며 IMS 큐에 남게 됩니다. 즉, 다시 시작할 때까지 추가적인 메시지가 송신되지 않습니다. 커미트 모드 1을 사용 중인 경우에는 다른 작업을 계속할 수 있습니다.

응답에 MQIIH 구조가 있는 경우 해당 형식 유형은 MQFMT_IMS입니다. 그렇지 않은 경우 형식 유형은 메시지를 삽입할 때 사용된 IMS MOD 이름에 의해 지정됩니다.

▶ z/OS IMS 트랜잭션에서 대체 응답 PCB 사용

IMS 트랜잭션이 대체 응답 PCB를 사용하면(ALTPCB에 ISRTR하거나 수정 가능한 PCB에 대한 CHNG 호출을 발행) 메시지를 다시 라우팅해야 하는지 여부를 판별하기 위해 사전 라우팅 엑시트(DFSYPRX0)가 호출됩니다.

메시지가 다시 라우팅되는 경우 목적지를 확인하기 위해 목적지 해석 엑시트(DFSYDRU0)가 호출되며 헤더 정보를 준비합니다. 해당 엑시트 프로그램에 대한 정보는 [IMS에서 OTMA 엑시트 사용 및 사전 라우팅 엑시트 DFSYPRX0](#)를 참조하십시오.

출력에서 조치가 취해지지 않는 한, IOPCB 또는 ALTPCB로 전송되는 모든 트랜잭션 IMS IBM MQ 출력은 동일한 큐 매니저로 반환됩니다.

▶ z/OS IMS의 요청되지 않은 메시지 송신

IMS에서 IBM MQ 큐로 메시지를 전송하려면 ALTPCB에 ISRT하는 IMS 트랜잭션을 호출해야 합니다.

메시지의 MQMD를 올바르게 빌드할 수 있도록 IMS의 요청되지 않은 메시지를 라우팅하고 OTMA 사용자 데이터를 빌드하기 위해 사전 라우팅 및 목적지 해석 엑시트를 작성해야 합니다. 이러한 엑시트 프로그램에 대한 정보는 [사전 라우팅 엑시트 DFSYPRX0 및 목적지 해석 엑시트](#)의 내용을 참조하십시오.

참고: IBM MQ - IMS 브릿지는 수신되는 메시지가 응답인지 또는 요청되지 않은 메시지인지 알지 못합니다. 메시지와 함께 도착한 OTMA UserData를 기반으로 응답의 MQMD 및 MQIIH를 빌드하면서 각각의 경우에서 동일한 방법으로 메시지를 처리합니다.

요청되지 않은 메시지는 새 Tpipe를 작성할 수 있습니다. 예를 들어, 기존의 IMS 트랜잭션이 새 LTERM(예: PRINT01)으로 전환되었지만 구현에서 출력이 OTMA를 통해 전달되어야 하는 경우 새 Tpipe(이 예에서는 PRINT01이라고 함)가 작성됩니다. 기본적으로, 이는 비동기화된 Tpipe입니다. 구현에서 메시지가 복구 가능해야 하는 경우 목적지 해석 엑시트 출력 플래그를 설정하십시오. 자세한 정보는 [IMS 사용자 정의 안내서](#)를 참조하십시오.

▶ z/OS 메시지 세그먼트화

단일 또는 다중 세그먼트 입력을 예상하는 것으로 IMS 트랜잭션을 정의할 수 있습니다.

원래 IBM MQ 애플리케이션은 하나 이상의 LLZZ-데이터 세그먼트로 다음 MQIIH 구조 뒤에 사용자 입력을 구성해야 합니다. IMS 메시지의 모든 세그먼트는 단일 MQPUT으로 송신된 단일 IBM MQ 메시지에 포함되어야 합니다.

LLZZ-데이터 세그먼트의 최대 길이는 IMS/OTMA(32,767바이트)에 의해 정의됩니다. 전체 IBM MQ 메시지 길이는 LL 바이트의 합에 MQIIH 구조의 길이를 더한 값입니다.

응답의 모든 세그먼트가 단일 IBM MQ 메시지에 포함됩니다.

MQFMT_IMS_VAR_STRING 형식의 메시지에 32KB 한계의 추가적인 제한사항이 있습니다. ASCII 혼합 CCSID 메시지의 데이터가 EBCDIC 혼합 CCSID 메시지로 변환되는 경우 SBCS와 DBCS 문자 간에 상태 전이가 발생할 때마다 시프트인 바이트 또는 시프트아웃 바이트가 추가됩니다. 32KB 제한사항은 메시지의 최대 크기에 적용됩니다. 즉, 메시지의 LL 필드는 32KB를 초과할 수 없기 때문에 메시지는 모든 시프트인 및 시프트아웃 문자를 포함하여 32KB를 초과하지 않아야 합니다. 메시지를 빌드하는 애플리케이션은 이를 허용해야 합니다.

▶ z/OS 교량과의 통신 IMS 메시지 데이터 변환

데이터 변환은 스토리지 클래스에 대해 정의된 XCF 정보를 가지고 있는 목적지 큐에 메시지를 넣는 경우 분배 큐잉 기능(필요한 경우 엑시트를 호출할 수 있는) 또는 인트라 그룹 큐잉 에이전트(엑시트의 사용을 지원하지 않는)에 의해 수행됩니다. 메시지가 발행/구독에 의해 큐로 전달되면 데이터 변환이 수행되지 않습니다.

필요한 엑시트의 경우 CSQXLIB DD 명령문에서 참조하는 데이터 세트의 분배 큐잉 기능에 사용 가능해야 합니다. 이는 모든 IBM MQ 플랫폼에서 - IMSIBM MQ 브릿지를 사용하여 애플리케이션에 IMS 메시지를 보낼 수 있음을 의미합니다.

변환 오류가 있는 경우 메시지는 변환되지 않은 큐에 놓입니다. 브릿지는 헤더 형식을 인식할 수 없기 때문에 결국 IBM MQ - IMS 브릿지에 의해 오류로 처리됩니다. 변환 오류가 발생하면 오류 메시지가 z/OS 콘솔로 송신됩니다.

일반적으로 데이터 변환에 대한 자세한 정보는 729 페이지의 『데이터 변환 엑시트 작성』의 내용을 참조하십시오.

IBM MQ - IMS 브릿지에 메시지 송신

변환이 올바르게 수행되었는지 확인하려면 큐 관리자에게 메시지 형식을 알려야 합니다.

메시지에 MQIIH 구조가 있는 경우 MQMD의 *Format*을 내장 형식 MQFMT_IMS로 설정해야 하며 MQIIH의 *Format*은 메시지 데이터를 설명하는 형식의 이름으로 설정되어야 합니다. MQIIH가 없는 경우 MQMD의 *Format*을 형식 이름으로 설정하십시오.

(LLZZ 이외의) 데이터가 모두 문자 데이터(MQCHAR)인 경우 내장 형식 MQFMT_IMS_VAR_STRING을 (MQIIH 또는 MQMD에서 적절한) 형식 이름으로 사용하십시오. 그렇지 않으면, 사용자 형식에 대한 데이터 변환 엑시트를 제공해야 하는 경우 사용자 고유 형식 이름을 사용하십시오. 엑시트는 데이터 자체뿐만 아니라 메시지에서 LLZZ의 변환을 핸들링해야 합니다(그러나, 메시지 시작에서 MQIIH를 핸들링할 필요는 없습니다).

애플리케이션에서 *MFSMapName*을 사용하는 경우 대신 MQFMT_IMS으로 메시지를 사용할 수 있으며 MQIIH의 *MFSMapName* 필드에서 IMS 트랜잭션에 전달된 맵 이름을 정의할 수 있습니다.

IBM MQ - IMS 브릿지에서 메시지 수신

MQIIH 구조가 IMS에 보내는 중인 원래 메시지에 있는 경우 응답 메시지에도 해당 구조가 존재합니다.

응답이 올바르게 변환되었는지 확인하려면 다음을 수행하십시오.

- 원래 메시지에 MQIIH 구조가 있는 경우 원래 메시지의 MQIIH *ReplytoFormat* 필드에서 응답 메시지에 원하는 형식을 지정하십시오. 이 값은 응답 메시지의 MQIIH *Format* 필드에 사용됩니다. 모든 출력 데이터가 LLZZ<문자 데이터> 양식인 경우 특히 유용합니다.
- 원래 메시지에 MQIIH 구조가 없는 경우 응답 메시지에 원하는 형식을 IOPCB에 대한 IMS 애플리케이션 ISRT의 MFS MOD 이름으로 지정하십시오.

개발 JMS/Jakarta Messaging 및 Java 애플리케이션

IBM MQ 세 가지 Java 언어 인터페이스를 제공합니다: IBM MQ classes for Jakarta Messaging, IBM MQ classes for JMS, 그리고 IBM MQ classes for Java.

▶ JM 3.0 IBM MQ classes for Jakarta Messaging

IBM MQ classes for Jakarta Messaging 메시징 시스템으로서 IBM MQ Jakarta Messaging 인터페이스를 구현하는 Jakarta Messaging 제공자입니다. 이 기술은 환경에서 Jakarta EE 실행되는 애플리케이션을 또는 IBM MQ 과 같은 Jakarta Connectors Architecture 기업 정보 시스템(EIS)에 Db2 연결하는 표준화된 방법을 제공합니다.

자세한 정보는 84 페이지의 『IBM MQ classes for Jakarta Messaging를 사용해야 하는 이유가 무엇입니까?』 및 87 페이지의 『IBM MQ - Java API 선택』의 내용을 참조하십시오.

➤ JMS 2.0 IBM MQ classes for JMS

IBM MQ classes for JMS 메시징 시스템으로서 IBM MQ JMS 인터페이스를 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.

자세한 정보는 86 페이지의 『IBM MQ classes for JMS를 사용해야 하는 이유가 무엇입니까?』 및 87 페이지의 『IBM MQ - Java API 선택』의 내용을 참조하십시오.

IBM MQ classes for Java

IBM MQ classes for Java 사용자가 환경에서 IBM MQJava 를 사용할 수 있도록 합니다. IBM MQ classes for Java를 사용하면 Java 애플리케이션을 IBM MQ 클라이언트로 IBM MQ에 연결하거나 IBM MQ 큐 관리자에 직접 연결할 수 있습니다.

IBM MQ classes for Java 는 네이티브 IBM MQ API인 메시지 큐 인터페이스(MQI)를 캡슐화하며, 다른 객체 지향 인터페이스와 동일한 객체 모델을 사용하는 반면, IBM MQ classes for JMS 와 IBM MQ classes for Jakarta Messaging 는 Java Community Process 각각 Oracle 의 메시징 인터페이스와 Java 를 구현합니다.

자세한 정보는 321 페이지의 『IBM MQ classes for Java를 사용해야 하는 이유가 무엇입니까?』 및 87 페이지의 『IBM MQ - Java API 선택』의 내용을 참조하십시오.

참고:

- **Stabilized** IBM은 IBM MQ classes for Java에 대해 추가적인 개선 계획이 없으며 IBM MQ 8.0에 제공된 레벨에서 기능적으로 안정되어 있습니다. IBM MQ classes for Java를 사용하는 기존 애플리케이션은 계속 완벽하게 지원되지만, 새 기능은 추가되지 않으며, 개선 요청은 거부됩니다. 완전히 지원이란 IBM MQ 시스템 요구사항의 변경에 따라 필요한 모든 변경사항과 함께 결합이 수정되는 것을 의미합니다.
- IBM MQ classes for Java는 IMS에서 지원되지 않습니다.
- IBM MQ classes for Java는 WebSphere® Liberty에서 지원되지 않습니다. 이는 IBM MQ Liberty 메시징 기능 또는 일반 JCA 지원에서는 사용될 수 없습니다. 자세한 내용은 [J2EE/JEE 환경에서 WebSphere MQJava 인터페이스 사용](#)을 참조하십시오.

버전별 Java 주요 변경 사항

때로는 버전 Java 변경으로 인해, IBM MQ classes for JMS, 또는 IBM MQ classes for Java IBM MQ classes for Jakarta Messaging애플리케이션에서 경고 메시지가 표시되거나 문제가 발생할 수 있습니다. 이러한 변경 사항들은 이 섹션에 요약되어 있습니다.

IBM Semeru Runtime 25

TLS_RSA_* CipherSpecs 가 비활성화되었습니다

TLS_RSA_* CipherSpecs 는. IBM Semeru Runtime 25에서 비활성화됩니다. 사용 중인 IBM MQ classes for JMSJakarta Messaging 애플리케이션이 다음 IBM MQ classes for Java 중 하나를 사용하는 경우 CipherSpecs, 다음 조치 중 하나를 수행하십시오:

- 지원되는 CipherSpec 를 사용하도록 애플리케이션을 업데이트하십시오.
- 파일의 java.security 비활성화 목록에서 해당 알고리즘을 제거하십시오.

자세한 내용은 [CipherSpecsTLS 그리고 CipherSuites 에서 IBM MQ classes for JMS](#) 및 [CipherSpecsTLS 그리고 CipherSuites 에서 IBM MQ classes for Java](#).

경고 메시지

- 응용 프로그램을 실행할 때 다음과 Java 25 같은 경고가 표시된다면, 응용 프로그램을 시작할 때 --enable-native-access=ALL-UNNAMEDJVM 매개변수를 전달하여 이 경고를 숨길 수 있습니다:

```
WARNING: A restricted method in java.lang.System has been called
WARNING: java.lang.System::loadLibrary has been called by
com.ibm.mq.jmqi.local.LocalMQ in an unnamed module (file:/opt/mqm/java/lib/
com.ibm.mq.allclient.jar)
WARNING: Use --enable-native-access=ALL-UNNAMED to avoid a warning for callers in this
module
```


WARNING: Restricted methods will be blocked in a future release unless native access is enabled

- (JRE25Java EE) Java Runtime Environment 25 와 WebSphere Liberty 같은 Java 2 Platform, Enterprise Edition 환경에서 애플리케이션을 실행할 때 다음과 같은 경고가 표시되면, 오류를 해결하려면 환경 Java 변수를 다음과 `JAVA_TOOL_OPTIONS=-Djdk.xml.enableExtensionFunctions=true` 같이 설정하십시오:

```
Errors while applying transformations: javax.xml.transform.TransformerException:
java.lang.RuntimeException: Use of
the extension element 'redirect' is not allowed when extension functions are disabled
by the secure processing
feature or the property 'jdk.xml.enableExtensionFunctions'. To enable extension
functions, set
'jdk.xml.enableExtensionFunctions' to 'true'
```

IBM Semeru Runtime 21

기본 인코딩이 변경되었습니다

에서 IBM Semeru Runtime 21 기본 인코딩은 UTF-8 이며, 이전에 Java 지원되던 버전에서는 기본 인코딩이 운영 체제의 기본값이었습니다(예를 들어, IBM-1047z/OS). 이 변경 사항은 텍스트 파일이나 속성 파일 등 'UTF-8'에 포함되지 않은 파일의 읽기 또는 쓰기 작업을 애플리케이션이 처리하는 방식에 영향을 미칠 수 있습니다. 예를 들어, EBCDIC 형식의 파일들. 다음 옵션 중 하나를 사용하여 기본 인코딩 문제를 해결할 수 있습니다:

- 응용 Java 프로그램을 실행할 때 명령줄에서 `file.encoding` 매개변수를 설정하여 Java runtime environment 사용할 인코딩을 지정하십시오. 예를 들면, 다음과 같습니다.

```
java -Dfile.encoding=IBM-1047 myApplication
```

이 COMPAT 키워드는 운영 체제의 기본값을 지정합니다:

```
java -Dfile.encoding=COMPAT myApplication
```

- 모든 파일 읽기/쓰기 작업에 대해 인코딩을 명시적으로 지정하도록 Java 애플리케이션을 업데이트하십시오.

이 변경 사항에 대한 자세한 내용은 문서의 [“파일 인코딩 및 기본 문자 z/OS 집합으로 UTF-8 IBM Semeru Runtime 21”](#)을 참조하십시오.

사용 IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 는 와 함께 제공되는 IBM MQ 메시징 Java 공급자입니다. 이 메시징 공급자들은 및 Jakarta Messaging JMS 사양에서 정의된 인터페이스를 구현하는 것 외에도, 메시징 Java API에 두 가지 확장 기능을 추가합니다.

JM 3.0 부터 IBM MQ 9.3.0, Jakarta Messaging 3.0 는 새로운 애플리케이션 개발을 지원합니다. IBM MQ 9.3.0 기존 애플리케이션에 대한 지원을 JMS 2.0 계속하고 있습니다. 동일한 애플리케이션에서 두 Jakarta Messaging 3.0 JMS 2.0 API를 모두 사용하는 것은 지원되지 않습니다.

참고: 이 경우 Jakarta Messaging 3.0, JMS 사양에 대한 제어권이 에서 Oracle 로 넘어갑니다 Java Community Process. 그러나 “javax”라는 이름에 대한 통제권은 여전히 Oracle 유지하고 있으며, 이 이름은 커뮤니티 Java 프로세스로 전환되지 않은 다른 Java 기술들에서도 사용되고 있습니다. 따라서, 는 기능적으로는 Jakarta Messaging 3.0 와 동일하지만 명명 방식에는 JMS 2.0 몇 가지 차이가 있습니다:

- 3.0 버전의 공식 명칭은 이지, Jakarta Messaging 가 Java Message Service 아닙니다.
- 패키지 및 상수 이름 앞에는 가 붙으며 jakarta, 는 붙지 javax 않습니다. 예를 들어, 메시징 서비스 제공자와의 초기 연결은 JMS 2.0 `javax.jms.Connection` 객체이고,에서는 Jakarta Messaging 3.0 `jakarta.jms.Connection` 객체입니다.

JMS 2.0 javax.jms 패키지는 JMS 인터페이스를 정의하며, 각 JMS 제공자는 특정 메시징 제품에 대해 이러한 인터페이스를 구현합니다. IBM MQ classes for JMS 는 의 JMS 인터페이스를 IBM MQ 구현하는 JMS 제공자입니다.

JM 3.0 jakarta.jms 패키지는 Jakarta Messaging 인터페이스를 정의하며, 각 Jakarta Messaging 제공자는 특정 메시징 제품에 대해 이러한 인터페이스를 구현합니다. IBM MQ classes for Jakarta Messaging 는 의 Jakarta Messaging 인터페이스를 IBM MQ구현하는 Jakarta Messaging 제공자입니다.

ConnectionFactory 과 Destination 객체는 관리 대상 객체여야 한다고 명시하고 Jakarta MessagingJMS 있습니다. 관리자는 중앙 저장소에서 관리 대상 객체를 생성 및 유지 관리하며, 애플리케이션은 JMSJakarta Messaging (JNDI) Java Naming Directory Interface 를 사용하여 이러한 객체를 검색합니다.

JMS 2.0 JMS 2.0관리자는 관리 도구 **JMSAdmin** 또는 IBM MQ ExplorerJMS 를 IBM MQ 사용하여 중앙 저장소에서 관리 대상 개체를 생성하고 유지 관리할 수 있습니다.

JM 3.0 IBM MQ ExplorerJakarta Messaging 3.0의 경우, 를 사용하여 JNDI를 관리할 수 없습니다. **JMS30Admin**JNDI 관리 기능은 의 **JMSAdmin** 변형인 Jakarta Messaging 3.0 에서 지원됩니다.

와 Jakarta Messaging 는 공통점이 많으므로 JMS , 이 주제에서 에 JMS 대한 언급은 두 가지 모두를 가리키는 것으로 볼 수 있습니다. 필요한 경우 차이점을 강조 표시했습니다.

IBM MQ classes for JMS에서는 두 개의 확장 기능 세트도 JMS API에 제공합니다. 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.

IBM MQ JMS 확장 기능

IBM MQ classes for JMS MQConnectionFactory,, MQQueue, MQTopic 객체와 같은 객체에 구현된 확장 기능을 포함합니다. 이러한 오브젝트에는 IBM MQ에 특정한 특성 및 메소드가 있습니다. 오브젝트는 관리 대상 오브젝트일 수 있거나 애플리케이션은 런타임에 오브젝트를 동적으로 작성할 수 있습니다. 이러한 확장들은 IBM MQJMS 확장들이라고 불린다.

IBM JMS 확장 기능

IBM MQ classes for JMS 또한 메시징 시스템이나 Java 사용되는 IBM MQ 프로그래밍 언어에 특화되지 않은, API에 JMS 대한 보다 일반적인 확장 기능 세트를 제공합니다. 이러한 확장은 확장이라고 IBMJMS 불리며 다음과 같은 광범위한 목표를 가지고 있습니다:

- 제공자 IBMJMS 간에 더 높은 수준의 일관성을 제공하기 위해.
- 두 IBM 메시징 시스템 간에 브릿지 애플리케이션을 작성하기 쉽게 하기 위해.
- 한 IBMJMS 공급자에서 다른 공급자로 애플리케이션을 더 쉽게 이식할 수 있도록 하기 위해.

확장 기능은 IBM MQ Message Service Client (XMS) for C/C++ 및 IBM MQ Message Service Client (XMS) for .NET에 제공된 기능과 비슷한 기능을 제공합니다.

관련 개념

[IBM MQ Java 언어 인터페이스](#)

관련 태스크

136 페이지의 『작성 IBM MQ classes for JMS/Jakarta Messaging 신청서』

모델에 JMS 대한 간략한 소개 후, 본 절에서는 및 IBM MQ classes for Jakarta Messaging 응용 프로그램을 IBM MQ classes for JMS 작성하는 방법에 대한 상세한 지침을 제공합니다.

JM 3.0 IBM MQ classes for Jakarta Messaging를 사용해야 하는 이유가 무엇입니까?

IBM MQ classes for Jakarta Messaging 사용 시 조직에서 기존 Jakarta Messaging 기술을 재사용할 수 있거나 애플리케이션이 Jakarta Messaging 제공자 및 기본 IBM MQ 구성으로부터 더욱 독립적이 될 수 있는 등 다양한 이점이 있습니다.

IBM MQ classes for Jakarta Messaging 사용 혜택 요약

IBM MQ classes for Jakarta Messaging를 사용하면 기존 Jakarta Messaging 기술을 재사용하고 애플리케이션 독립성을 제공할 수 있습니다.

- Jakarta Messaging 기술을 재사용할 수 있습니다.

IBM MQ classes for Jakarta Messaging 메시징 시스템으로서 IBM MQ Jakarta Messaging 인터페이스를 구현하는 Jakarta Messaging 제공자입니다. 조직이 를 처음 IBM MQ접하지만 이미 (또는 JMS) Jakarta Messaging 애플리케이션 개발 기술을 보유하고 있다면, 와 함께 제공되는 다른 API 중 하나를 사용하는 것보다 익숙한 Jakarta Messaging IBM MQAPI를 사용하여 리소스에 IBM MQ 접근하는 것이 더 쉬울 수 있습니다.

- Jakarta Messaging 의 필수적인 Jakarta EE부분이다.

Jakarta Messaging는 Jakarta EE 플랫폼에서 메시징에 사용할 자연적인 API입니다. Jakarta EE 호환 가능한 모든 애플리케이션 서버에는 Jakarta Messaging 제공자가 포함되어 있어야 합니다. 애플리케이션 클라이언트, 서블릿, Java 서버 페이지(JSP), 엔터프라이즈 Java 빈(EJB), 메시지 주도 빈(MDB)에서 Jakarta Messaging 사용할 수 있습니다. 특히 Jakarta EE 애플리케이션이 메시지를 비동기적으로 처리하기 위해 MDB 를 사용하고 모든 메시지가 Jakarta Messaging 메시지로 MDB에 전달됨에 유의하십시오.

- 연결 팩토리 및 목적지는 애플리케이션으로 하드 코딩되기보다 중앙 저장소에 Jakarta Messaging 관리 대상 오브젝트로 저장될 수 있습니다.

관리자는 중앙 저장소에서 Jakarta Messaging 관리 대상 오브젝트를 작성하고 유지보수할 수 있으며, IBM MQ classes for Jakarta Messaging 애플리케이션은 Java Naming Directory Interface(JNDI)을(를) 사용하여 이러한 오브젝트를 검색할 수 있습니다. Jakarta Messaging 연결 팩토리 및 목적지는 큐 관리자 이름, 채널 이름, 연결 옵션, 큐 이름 및 토픽 이름과 같은 IBM MQ 특정 정보를 캡슐화합니다. 연결 팩토리 및 목적지가 관리 대상 오브젝트로 저장된 경우, 이 정보는 애플리케이션으로 하드 코딩되지 않습니다. 따라서 이 배열은 기본 IBM MQ 구성으로부터 독립성의 정도를 애플리케이션에 제공합니다.

- Jakarta Messaging는 애플리케이션 이식성을 제공할 수 있는 산업 표준 API입니다.

애플리케이션은 Jakarta Messaging 관리 객체로 저장된 연결 팩토리 및 목적지를 검색하는 데 JNDI 사용할 수 있으며, 메시징 작업을 수행하기 위해 (Jakarta Messaging 3.0) jakarta.jms 패키지에 정의된 인터페이스만 사용할 수 있습니다. 해당 애플리케이션은 이후 와 IBM MQ classes for Jakarta Messaging 같은 어떤 Jakarta Messaging 공급자에도 완전히 독립적이며, 애플리케이션 자체를 변경하지 않고도 한 Jakarta Messaging 공급자에서 다른 공급자로 이식할 수 있습니다.

특정 애플리케이션 환경에서 가 사용할 수 없는 JNDI 경우, IBM MQ classes for Jakarta Messaging 애플리케이션은 API의 Jakarta Messaging 확장을 사용하여 런타임 시점에 동적으로 연결 팩토리와 목적지를 생성하고 구성할 수 있습니다. 해당 애플리케이션은 완전히 독립적으로 작동하지만, 를 IBM MQ classes for Jakarta Messaging Jakarta Messaging 프로바이더로 지정하여 연결됩니다.

- 브릿지 애플리케이션은 Jakarta Messaging를 사용하여 더 쉽게 작성할 수 있습니다.

브릿지 애플리케이션은 메시징 시스템에서 메시지를 수신하여 이를 다른 메시징 시스템으로 송신하는 애플리케이션입니다. 브릿지 애플리케이션 기록은 제품별 API 및 메시지 형식을 사용함으로써 복잡해질 수 있습니다. 대신 각 메시징 시스템당 하나씩, 두 개의 Jakarta Messaging 제공자를 사용하여 브릿지 애플리케이션을 기록할 수 있습니다. 그런 다음 애플리케이션은 단 하나의 API인 Jakarta Messaging API만 사용하며 Jakarta Messaging 메시지만 처리합니다.

배치 가능한 환경

Jakarta EE 애플리케이션 서버와의 통합을 제공하려면 Jakarta EE 표준에 자원 어댑터를 제공하기 위한 메시징 제공자가 필요합니다. 사양에 Jakarta Connectors Architecture 따라, 인증된 Jakarta EE 환경 내에서 메시징 기능을 제공하기 IBM MQ 위해 를 Jakarta Messaging 사용하는 리소스 어댑터를 제공합니다. 자세한 내용은 을 참조하십시오 391 페이지의 『WebSphere Liberty 그리고 IBM MQ 리소스 어댑터』.

참고: Jakarta EEWebSphere Application Server 현재는 지원하지 않습니다.

Jakarta EE 환경 외부에 OSGi 및 JAR 파일이 제공되며, 이는 IBM MQ classes for Jakarta Messaging를 더 쉽게 얻게 해줍니다. 이러한 JAR 파일은 독립적으로 또는 Maven과 같은 소프트웨어 관리 프레임워크 내에서 더 쉽게 배포할 수 있습니다. 자세한 내용은 을 참조하십시오 124 페이지의 『와 IBM MQ classes for JMS 를 IBM MQ classes for Jakarta Messaging 각각 획득하다』.

관련 개념

IBM MQ Jakarta Messaging 클래스: 개요

87 페이지의 『IBM MQ - Java API 선택』

IBM MQ 세 가지 Java 언어 인터페이스를 제공합니다.

JMS 2.0 IBM MQ classes for JMS를 사용해야 하는 이유가 무엇입니까?

IBM MQ classes for JMS 사용 시 조직에서 기존 JMS 기술을 재사용할 수 있거나 애플리케이션이 JMS 제공자 및 기본 IBM MQ 구성으로부터 더욱 독립적이 될 수 있는 등 다양한 이점이 있습니다.

IBM MQ classes for JMS 사용 혜택 요약

IBM MQ classes for JMS를 사용하면 기존 JMS 기술을 재사용하고 애플리케이션 독립성을 제공할 수 있습니다.

참고: JMS 2.0 에 의해 대체되었습니다 Jakarta Messaging. IBM MQ classes for JMS JMS 2.0 표준을 계속 지원하지만, 메시징에 Java 대한 향후 개선 사항은 오직 에서만 등장할 것이므로, 결국 Jakarta Messaging에서만 나타날 IBM MQ classes for Jakarta Messaging것입니다. IBM MQ classes for JMS 기존 JMS 2.0 애플리케이션의 유지 관리 및 확장에만 권장됩니다. IBM MQ classes for Jakarta Messaging 신규 개발에 있어 선호되는 기술이어야 한다.

- JMS 기술을 재사용할 수 있습니다.

IBM MQ classes for JMS 메시징 시스템으로서 IBM MQ JMS 인터페이스를 구현하는 JMS 제공자입니다. 조직이 를 처음 IBM MQ 접하지만 이미 애플리케이션 개발 JMS 역량을 보유하고 있다면, 와 함께 제공되는 다른 API들보다 익숙한 JMS API를 사용하여 IBM MQ 리소스에 IBM MQ 접근하는 것이 더 쉬울 수 있습니다.

- JMS는 Java Platform, Enterprise Edition(Java EE)의 내부 부분입니다.

JMS는 Java EE 플랫폼에서 메시징에 사용할 자연적인 API입니다. Java EE 호환 가능한 모든 애플리케이션 서버에는 JMS 제공자가 포함되어 있어야 합니다. 애플리케이션 클라이언트, 서블릿, Java 서버 페이지(JSP), 엔터프라이즈 Java 빈(EJB), 메시지 주도 빈(MDB)에서 JMS 사용할 수 있습니다. 특히 Java EE 애플리케이션이 메시지를 비동기적으로 처리하기 위해 MDB를 사용하고 모든 메시지가 JMS 메시지로 MDB에 전달됨에 유의 하십시오.

- 연결 팩토리 및 목적지는 애플리케이션으로 하드 코딩되기보다 중앙 저장소에 JMS 관리 대상 오브젝트로 저장 될 수 있습니다.

관리자는 중앙 저장소에서 JMS 관리 대상 오브젝트를 작성하고 유지보수할 수 있으며, IBM MQ classes for JMS 애플리케이션은 Java Naming Directory Interface(JNDI)을(를) 사용하여 이러한 오브젝트를 검색할 수 있습니다. JMS 연결 팩토리 및 목적지는 큐 관리자 이름, 채널 이름, 연결 옵션, 큐 이름 및 토픽 이름과 같은 IBM MQ 특정 정보를 캡슐화합니다. 연결 팩토리 및 목적지가 관리 대상 오브젝트로 저장된 경우, 이 정보는 애플리케이션으로 하드 코딩되지 않습니다. 따라서 이 배열은 기본 IBM MQ 구성으로부터 독립성의 정도를 애플리케이션에 제공합니다.

- JMS는 애플리케이션 이식성을 제공할 수 있는 산업 표준 API입니다.

애플리케이션은 JMS 관리 객체로 저장된 연결 팩토리 및 목적지를 검색하는 데 JNDI 사용할 수 있으며, 메시징 작업을 수행하기 위해 javax.jms 패키지에 정의된 인터페이스만 사용할 수 있습니다. 해당 애플리케이션은 이후 와 IBM MQ classes for JMS 같은 어떤 JMS 공급자에도 완전히 독립적이며, 애플리케이션 자체를 변경하지 않고도 한 JMS 공급자에서 다른 공급자로 이식할 수 있습니다.

특수한 애플리케이션 환경에서 JNDI를 사용할 수 없는 경우, IBM MQ classes for JMS 애플리케이션이 런타임 시에 동적으로 연결 팩토리 및 목적지를 작성 및 구성하기 위해 JMS API에 대한 확장을 사용할 수 있습니다. 해당 애플리케이션은 완전히 독립적으로 작동하지만, 의 IBM MQ classes for JMS 제공자로 JMS 지정됩니다.

- 브릿지 애플리케이션은 JMS를 사용하여 더 쉽게 작성할 수 있습니다.

브릿지 애플리케이션은 메시징 시스템에서 메시지를 수신하여 이를 다른 메시징 시스템으로 송신하는 애플리케이션입니다. 브릿지 애플리케이션 기록은 제품별 API 및 메시지 형식을 사용함으로써 복잡해질 수 있습니다. 대신 각 메시징 시스템당 하나씩, 두 개의 JMS 제공자를 사용하여 브릿지 애플리케이션을 기록할 수 있습니다. 그런 다음 애플리케이션은 단 하나의 API인 JMS API만 사용하며 JMS 메시지만 처리합니다.

배치 가능한 환경

Java EE 애플리케이션 서버와의 통합을 제공하려면 Java EE 표준에 자원 어댑터를 제공하기 위한 메시징 제공자가 필요합니다. (JCA) Java EE Connector Architecture 사양에 따라, 는 인증된 Java EE 환경 내에서 메시징 기능을 제공하기 IBM MQ 위해 를 JMS 사용하는 리소스 어댑터를 제공합니다.

내부에서 IBM MQ classes for Java 사용할 수는 Java EE 있지만, 이 API는 해당 목적을 위해 설계되거나 최적화되지 않았습니다. 에 대한 IBM MQ classes for Java 고려 사항에 대한 자세한 Java EE 내용은 을 참조하십시오 [322 페이지의 『애플리케이션 IBM MQ classes for Java 실행 Java EE』](#).

Java EE 환경 외부에 OSGi 및 JAR 파일이 제공되며, 이는 IBM MQ classes for JMS를 더 쉽게 얻게 해줍니다. 이러한 JAR 파일은 독립적으로 또는 Maven과 같은 소프트웨어 관리 프레임워크 내에서 더 쉽게 배포할 수 있습니다. 자세한 내용은 을 참조하십시오 [124 페이지의 『와 IBM MQ classes for JMS 를 IBM MQ classes for Jakarta Messaging 각각 획득하다』](#).

관련 개념

[IBM MQ Jakarta Messaging 클래스: 개요](#)

[84 페이지의 『IBM MQ classes for Jakarta Messaging를 사용해야 하는 이유가 무엇입니까?』](#)

IBM MQ classes for Jakarta Messaging 사용 시 조직에서 기존 Jakarta Messaging 기술을 재사용할 수 있거나 애플리케이션이 Jakarta Messaging 제공자 및 기본 IBM MQ 구성으로부터 더욱 독립적이 될 수 있는 등 다양한 이점이 있습니다.

[87 페이지의 『IBM MQ - Java API 선택』](#)

IBM MQ 세 가지 Java 언어 인터페이스를 제공합니다.

IBM MQ - Java API 선택

IBM MQ 세 가지 Java 언어 인터페이스를 제공합니다.

- IBM MQ classes for Jakarta Messaging
- IBM MQ classes for JMS
- IBM MQ classes for Java

IBM MQ classes for Jakarta Messaging

IBM MQ classes for Jakarta Messaging 해당 Jakarta Messaging 3.0 API를 사용하여 작성된 애플리케이션이 메시징 공급자로 IBM MQ 활용할 수 있도록 합니다.

Jakarta Messaging 애플리케이션 Java 내 메시징의 전략적 방향입니다.

Jakarta Messaging 3.0 기능적으로 와 동등하므로 JMS 2.0, 자세한 내용은 을 참조하십시오 [83 페이지의 『사용 IBM MQ classes for JMS/Jakarta Messaging』](#).

IBM MQ classes for JMS

IBM MQ classes for JMS 해당 JMS 2.0 API를 사용하여 작성된 애플리케이션이 메시징 공급자로 IBM MQ 활용할 수 있도록 합니다.

는 Jakarta Messaging 를 대체하며 JMS, 기존 애플리케이션이나 를 지원하지 않는 IBM MQ classes for JMS 환경(예: WebSphere Application Server)에서 Jakarta Messaging 사용하도록 권장됩니다.

동일한 애플리케이션에서 IBM MQ classes for JMS 와 를 IBM MQ classes for Jakarta Messaging 동시에 사용하는 것은 지원되지 않습니다.

자세한 정보는 [83 페이지의 『사용 IBM MQ classes for JMS/Jakarta Messaging』](#) 의 내용을 참조하십시오.

IBM MQ classes for Java

Stabilized 응용 프로그램이 리소스에 JavaIBM MQ 접근하기 위해 사용할 수 있는 다른 API는 로컬 메시징 서비스(LMS)로 IBM MQ classes for Java, 프로그램이 메시징 공급자로 IBM MQ 사용할 수 있는 객체 지향 IBM MQAPI를 제공합니다. 그러나 기능적으로 에 포함된 수준에서 IBM MQ 8.0 안정화됩니다 IBM MQ classes for Java . 자세한 정보는 [321 페이지의 『IBM MQ classes for Java를 사용해야 하는 이유가 무엇입니까?』](#) 의 내용을 참조하십시오. 기존 애플리케이션은 계속해서 완전히 지원되지만, 새로운 IBM MQ classes for Java 애플리케이션은 를 사용해야 합니다 IBM MQ classes for Jakarta Messaging.

와 IBM MQ classes for JMS 의 공통된 특징 IBM MQ classes for Jakarta Messaging

IBM MQ classes for JMS 그리고 의 지점 간(point-to-point) 및 게시/구독(IBM MQ classes for Jakarta Messaging IBM MQ publish/subscribe) 메시징 기능 모두에 대한 접근을 제공합니다. JMS 표준 메시징 모델에 대한 지원을 제공하는 JMS 메시지 송신뿐 아니라 애플리케이션은 추가 헤더 없이 메시지를 송수신할 수도 있으며 따라서 다른 IBM MQ 애플리케이션(예: C MQI 애플리케이션)과 상호 작용할 수 있습니다. MQMD 및 MQ 메시징 페이로드를 완전히 제어할 수 있습니다.

메시징 스트리밍, 비동기 넣기 및 보고 메시지 같은 추가 IBM MQ 기능도 사용 가능합니다.

IBM MQ PCF 관리 메시지는 제공된 PCF 헬퍼 클래스를 사용하여 JMS API를 통해 송수신될 수 있으며 큐 관리자를 관리하는 데 사용될 수 있습니다.

최근에 추가된 IBM MQ 기능들, 예를 들어 비동기적 소비 및 자동 재연결은 에서 IBM MQ classes for Java 사용할 수 없지만, 와 IBM MQ classes for JMS 에서 사용할 IBM MQ classes for Jakarta Messaging 수 있습니다.

개선사항 요청

와 를 IBM MQ classes for JMS 통해 제공되지 않는 IBM MQ classes for Jakarta Messaging 기능에 IBM MQ 대한 접근이 필요한 경우, 아이디어를 제안할 수 있습니다.

IBM 그런 다음 구현이 또는 IBM MQ classes for Jakarta Messaging IBM MQ classes for JMS 구현에서 가능한지, 또는 따를 수 있는 모범 사례가 있는지 여부를 조언할 수 있습니다.

추가 메시징 기능의 경우 IBM이 열린 표준에 대한 기여자이므로 이러한 기능은 JCP 프로세스의 일부로 격상될 수 있습니다. 이는 자카르타 메시징에만 적용됩니다.

관련 정보

[IBM 아이디어 포털에 오신 것을 환영합니다](#)

[JMS Java 사양 검토 절차](#)

[PCF 메시지를 송신하기 위해 JMS 사용](#)

IBM 3.0 IBM MQ classes for Jakarta Messaging의 전제조건

이 주제에서는 IBM MQ classes for Jakarta Messaging를 사용하기 전에 알아야 하는 사항을 설명합니다. IBM MQ classes for Jakarta Messaging 애플리케이션을 개발하고 실행하려면 필수조건으로 특정 소프트웨어 컴포넌트가 있어야 합니다.

IBM MQ classes for Jakarta Messaging의 필수조건에 대한 정보는 [시스템 요구 사항 IBM MQ](#)의 내용을 참조하십시오.

IBM MQ classes for Jakarta Messaging 애플리케이션을 개발하려면 SDK(Java SE Software Development Kit)가 필요합니다. 운영 체제에서 지원되는 JDK에 대한 자세한 내용은 [시스템 요구 사항 IBM MQ](#)의 내용을 참조하십시오.

IBM MQ classes for Jakarta Messaging 애플리케이션을 실행하려면 다음 소프트웨어 컴포넌트가 필요합니다.

- IBM MQ 큐 관리자.
- A Java runtime environment (JRE), 애플리케이션을 실행하는 각 시스템마다.
-  IBM i의 경우 운영 체제의 옵션 30인 Qshell.
-  에 대해 z/OS, z/OS UNIX System Services (z/OS UNIX).

JSSE IBM 공급자는 인증된 FIPS 암호화 공급자를 포함하므로, 즉시 사용 가능한 규정 준수 FIPS 140-2 상태로 프로그래밍 방식으로 구성할 수 있습니다. 따라서, FIPS 140-2  또는 FIPS 140-3 준수는 에서 직접 지원될 IBM MQ classes for Jakarta Messaging 수 있습니다.

Oracle 의 JSSE 공급자는 인증된 FIPS 암호화 공급자를 구성할 수 있지만, 즉시 사용 가능하지 않으며 프로그래밍 방식으로 구성할 수 없습니다. 따라서, 이 경우  또는 FIPS 140-3 준수를 FIPS 140-2 직접 활성화할 수 없습니다 IBM MQ classes for Jakarta Messaging . 이러한 준수를 수동으로 사용할 수 있도록 할 수 있지만 IBM은 현재 해당 지침을 제공할 수 없습니다.

사용 중인 Java 가상 머신(JVM)과 운영 체제의 TCP/IP 구현에서 해당 주소를 지원하는 경우 IPv6, 애플리케이션에서 IBM MQ classes for Jakarta Messaging Internet Protocol 버전 6(IPv6) 주소를 사용할 수 있습니다. 관리 IBM MQJakarta Messaging 도구, **JMS30Admin**, 또한 주소를 IPv6 허용합니다. 이 도구에 대한 자세한 내용은 관리 도구를 사용하여 및 Jakarta Messaging 객체를 JMS 구성하는 방법을 참조하십시오.

관리 IBM MQJMS 도구를 사용하여 IBM MQ Explorer (JNDI) Java Naming Directory Interface 를 통해 관리되는 개체를 저장하는 디렉터리 서비스에 액세스합니다. IBM MQ classes for Jakarta Messaging 애플리케이션에서는 JNDI도 사용하여 디렉터리 서비스에서 관리 대상 오브젝트를 검색할 수 있습니다.

참고: IBM MQ ExplorerJakarta Messaging 3.0의 경우, 를 사용하여 JNDI를 관리할 수 없습니다. **JMS30Admin**JNDI 관리 기능은 의 **JMSAdmin** 변형인 Jakarta Messaging 3.0 에서 지원됩니다.

서비스 제공자는 JNDI 호출을 디렉터리 서비스에 매핑하여 디렉터리 서비스에 액세스를 제공하는 코드입니다. fscontext.jar 및 providerutil.jar 파일의 파일 시스템 서비스 제공자는 IBM MQ classes for Jakarta Messaging와 함께 제공됩니다. 파일 시스템 서비스 공급자는 로컬 파일 시스템을 기반으로 한 디렉터리 서비스에 대한 접근을 제공합니다.

LDAP 서버를 기반으로 디렉터리 서비스를 사용하려는 경우 LDAP 서버를 설치하고 구성하거나 기존 LDAP 서버에 대한 액세스가 있어야 합니다. 특히 Java 오브젝트를 저장하도록 LDAP 서버를 구성해야 합니다. LDAP 서버를 설치하고 구성하는 방법에 대한 정보는 서버와 함께 제공되는 문서를 참조하십시오.

JMS 2.0 IBM MQ classes for JMS의 전제조건

이 주제에서는 IBM MQ classes for JMS를 사용하기 전에 알아야 하는 사항을 설명합니다. IBM MQ classes for JMS 애플리케이션을 개발하고 실행하려면 필수조건으로 특정 소프트웨어 컴포넌트가 있어야 합니다.

IBM MQ classes for JMS의 필수조건에 대한 정보는 시스템 요구 사항 IBM MQ의 내용을 참조하십시오.

IBM MQ classes for JMS 애플리케이션을 개발하려면 SDK(Java SE Software Development Kit)가 필요합니다. 운영 체제에서 지원되는 JDK에 대한 자세한 내용은 시스템 요구 사항 IBM MQ의 내용을 참조하십시오.

IBM MQ classes for JMS 애플리케이션을 실행하려면 다음 소프트웨어 컴포넌트가 필요합니다.

- IBM MQ 큐 관리자.
- A Java runtime environment (JRE), 애플리케이션을 실행하는 각 시스템마다.
-  IBM i의 경우 운영 체제의 옵션 30인 Qshell.
-  에 대해 z/OS, z/OS UNIX System Services (z/OS UNIX).

JSSE IBM 공급자는 인증된 FIPS 암호화 공급자를 포함하므로, 즉시 사용 가능한 규정 준수 FIPS 140-2 상태로 프로그래밍 방식으로 구성할 수 있습니다. 따라서, FIPS 140-2  또는 FIPS 140-3 준수는 및 IBM MQ classes for Java 에서 직접 IBM MQ classes for JMS지원될 수 있습니다.

Oracle 의 JSSE 공급자는 인증된 FIPS 암호화 공급자를 구성할 수 있지만, 즉시 사용 가능하지 않으며 프로그래밍 방식으로 구성할 수 없습니다. 따라서, 이 경우 와 IBM MQ classes for Java 는  또는 FIPS IBM MQ classes for JMS 140-3 FIPS 140-2 준수를 직접 활성화할 수 없습니다. 이러한 준수를 수동으로 사용할 수 있도록 할 수 있지만 IBM은 현재 해당 지침을 제공할 수 없습니다.

사용 중인 Java 가상 머신(JVM)과 운영 체제의 TCP/IP 구현에서 해당 주소를 지원하는 경우 IPv6, 애플리케이션에서 IBM MQ classes for JMS Internet Protocol 버전 6 (IPv6) 주소를 사용할 수 있습니다. IBM MQ JMS 관리 도구(관리 도구를 사용하여 JMS 오브젝트 구성 참조)는 IPv6 주소도 승인합니다.

관리 IBM MQJMS 도구를 사용하여 IBM MQ Explorer (JNDI) Java Naming Directory Interface 를 통해 관리되는 개체를 저장하는 디렉터리 서비스에 액세스합니다. IBM MQ classes for JMS 애플리케이션에서는 JNDI도 사용하여 디렉터리 서비스에서 관리 대상 오브젝트를 검색할 수 있습니다. 서비스 제공자는 JNDI 호출을 디렉터리 서비스에 매핑하여 디렉터리 서비스에 액세스를 제공하는 코드입니다. fscontext.jar 및 providerutil.jar 파일의 파일 시스템 서비스 제공자는 IBM MQ classes for JMS과(와) 함께 제공됩니다. 파일 시스템 서비스 제공자는 로컬 파일 시스템을 기반으로 디렉터리 서비스에 대한 액세스를 제공합니다.

LDAP 서버를 기반으로 디렉터리 서비스를 사용하려는 경우 LDAP 서버를 설치하고 구성하거나 기존 LDAP 서버에 대한 액세스가 있어야 합니다. 특히 Java 오브젝트를 저장하도록 LDAP 서버를 구성해야 합니다. LDAP 서버를 설치하고 구성하는 방법에 대한 정보는 서버와 함께 제공되는 문서를 참조하십시오.

설치 및 구성 IBM MQ classes for JMS/Jakarta Messaging

이 섹션에서는 IBM MQ classes for JMS 설치 시 생성되는 IBM MQ classes for Jakarta Messaging 디렉터리와 파일을 설명하고, 설치 후 IBM MQ classes for Jakarta Messaging 및 IBM MQ classes for JMS 구성 방법을 안내합니다.

관련 개념

386 페이지의 『리소스 IBM MQ 어댑터 사용』

리소스 IBM MQ 어댑터는 애플리케이션 서버에서 실행 중인 애플리케이션이 리소스에 IBM MQ 접근할 수 있도록 합니다. 리소스 어댑터는 인바운드 및 아웃바운드 통신을 지원합니다.

IBM MQ classes for JMS에 설치된 항목

IBM MQ classes for JMS를 설치할 때 여러 파일과 디렉터리가 작성됩니다. Windows에서 환경 변수를 자동으로 설정하여 설치 중에 일부 구성을 수행합니다. 다른 플랫폼과 특정 Windows 환경에서 환경 변수를 설정해야 IBM MQ classes for JMS 애플리케이션을 실행할 수 있습니다.

대부분의 운영 체제에서 IBM MQ classes for JMS는 IBM MQ를 설치할 때 선택적 컴포넌트로 설치됩니다.

IBM MQ 설치에 대한 자세한 정보는 다음을 참조하십시오.

- ▶ **Multi** IBM MQ 설치
- ▶ **z/OS** IBM MQ for z/OS 설치

중요사항: 92 페이지의 『IBM MQ classes for JMS/Jakarta Messaging 재배포 가능한 JAR 파일』에서 설명하는 재배포 가능 JAR 파일과 별개로 IBM MQ classes for JMS JAR 파일이나 고유 라이브러리를 다른 시스템 또는 IBM MQ classes for JMS가 설치된 시스템의 다른 위치에 복사하는 기능은 지원되지 않습니다.

설치 디렉터리

90 페이지의 표 5에서는 각 플랫폼에 IBM MQ classes for JMS 파일이 설치된 위치를 표시합니다.

표 5. IBM MQ classes for JMS 설치 디렉터리		
플랫폼	디렉터리	
▶ Linux and Linux	▶ AIX AIX	MQ_INSTALLATION_PATH/java
▶ Windows Windows		MQ_INSTALLATION_PATH\java
▶ IBM i IBM i		/QIBM/ProdData/mqm/java
▶ z/OS z/OS		MQ_INSTALLATION_PATH/java

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

설치 디렉터리에는 다음 콘텐츠가 포함됩니다.

- MQ_INSTALLATION_PATH\java\lib 디렉터리에 있는 재배포 가능 JAR 파일을 포함하는 IBM MQ classes for JMS JAR 파일.
- 네이티브 Java 인터페이스를 사용하는 애플리케이션에서 활용되는 네이티브 IBM MQ 라이브러리들.
네이티브 라이브러리는 디렉터리에 MQ_INSTALLATION_PATH\java\lib64 설치됩니다.
IBM MQ 고유 라이브러리에 관한 자세한 정보는 96 페이지의 『JNI(Java Native Interface) 라이브러리 구성』의 내용을 참조하십시오.
- 121 페이지의 『와 IBM MQ classes for JMS 함께 제공되는 스크립트 Jakarta Messaging』에 설명된 추가 스크립트. 이러한 스크립트는 MQ_INSTALLATION_PATH\java\bin 디렉터리에 있습니다.
- IBM MQ classes for JMS API의 스펙입니다. API 스펙을 포함하는 HTML 페이지를 생성하기 위해 Javadoc 도구가 사용되었습니다.

HTML 페이지는 MQ_INSTALLATION_PATH\java\doc\WMQJMSClasses 디렉터리에 있습니다.

- **ALW** AIX, Linux, and Windows에서는 이 서브디렉토리에 개별 HTML 페이지가 포함됩니다.
- **IBM i** IBM i에서 HTML 페이지는 `wmqjms_javadoc.jar` 파일에 있습니다.
- **z/OS** z/OS에서 HTML 페이지는 `wmqjms_javadoc.jar` 파일에 있습니다.
- OSGi 지원. OSGi 번들은 `java\lib\OSGi` 디렉토리에 설치되며 [122 페이지의 『IBM MQ classes for JMS를 사용하여 OSGi 지원』](#)에 설명되어 있습니다.
- 리소스 IBM MQ 어댑터는 어떤 Java Platform, Enterprise Edition 7 (Java EE 7) 또는 Jakarta EE 호환 애플리케이션 서버에도 배포될 수 있습니다.

IBM MQ 자원 어댑터는 `MQ_INSTALLATION_PATH\java\lib\jca` 디렉토리에 있습니다. 자세한 정보는 [386 페이지의 『리소스 IBM MQ 어댑터 사용』](#)의 내용을 참조하십시오.

- **Windows** Windows에서, 디버깅에 사용할 수 있는 기호는 `MQ_INSTALLATION_PATH\java\lib\symbols` 디렉토리에 설치됩니다.

설치 디렉토리에는 다른 IBM MQ 컴포넌트에 속한 파일도 포함될 수 있습니다.

샘플 애플리케이션

JMS 2.0 샘플 애플리케이션은 IBM MQ classes for JMS와 함께 제공됩니다. [91 페이지의 표 6](#)에서는 각 플랫폼에서 샘플 애플리케이션이 설치된 위치를 표시합니다.

JM 3.0 현재 IBM MQ classes for Jakarta Messaging 새로운 샘플을 준비 중입니다.

JMS 2.0

표 6. 샘플 디렉터리 IBM MQ classes for JMS	
플랫폼	디렉토리
Linux and Linux	<code>MQ_INSTALLATION_PATH/samp/jms</code>
AIX	<code>MQ_INSTALLATION_PATH/samp/jms</code>
Windows Windows	<code>MQ_INSTALLATION_PATH\tools\jms</code>
IBM i IBM i	<code>/QIBM/ProdData/mqm/java/samples/jms</code>
z/OS z/OS	<code>MQ_INSTALLATION_PATH/java/samples/jms</code>

이 표에서 `MQ_INSTALLATION_PATH`는 IBM MQ가 설치된 상위 레벨 디렉토리를 나타냅니다.

설치 후 애플리케이션을 컴파일하고 실행하는 구성 태스크를 수행해야 합니다.

[94 페이지의 『환경 변수 IBM MQ classes for JMS 설정Jakarta Messaging』](#) 샘플 IBM MQ classes for JMS 애플리케이션을 실행하는 데 필요한 클래스패스를 설명합니다. 이 주제에서는 IBM MQ classes for JMS와 함께 제공되는 스크립트를 실행하는 데 설정해야 하는 환경 변수와 특수 상황에서 참조해야 하는 추가 JAR 파일을 설명합니다.

애플리케이션 추적 및 로깅과 같은 특성을 제어하려면 구성 특성 파일을 제공해야 합니다. IBM MQ classes for JMS 구성 특성 파일은 [98 페이지의 『Jakarta Messaging 구성 IBM MQ classes for JMS 파일』](#)에 설명되어 있습니다.

관련 개념

[자원 어댑터를 배치하는 문제점](#)

관련 태스크

[118 페이지의 『IBM MQ classes for JMS 샘플 애플리케이션 사용』](#)

IBM MQ classes for JMS 샘플 애플리케이션은 JMS API의 공통 기능 개요를 제공합니다. 이를 사용하여 설치 및 메시징 서버 설정을 확인하고 자신만의 애플리케이션을 빌드하는 데 도움을 받을 수 있습니다.

IBM MQ classes for JMS/Jakarta Messaging 재배포 가능한 JAR 파일

이동 가능한 JAR 파일은 또는 을 IBM MQ classes for JMS 실행해야 하는 IBM MQ classes for Jakarta Messaging 시스템으로 이동할 수 있습니다.

중요사항:

- 이동 가능한 JAR 파일에 설명된 이동 가능한 JAR 파일을 제외하고, 또는 IBM MQ classes for Jakarta Messaging JAR 파일이나 IBM MQ classes for JMS 네이티브 라이브러리를 다른 시스템으로 복사하거나, 또는 가 IBM MQ classes for Jakarta Messaging IBM MQ classes for JMS 설치된 시스템 내 다른 위치로 복사하는 것은 지원되지 않습니다.
- Java EE 애플리케이션 서버(예: WebSphere Application Server 또는 WebSphere Liberty)에 배치된 애플리케이션 내 재배포 가능 JAR 파일을 포함하지 마십시오. 이러한 환경에서 IBM MQ 자원 어댑터가 대신 배치 및 사용됩니다. WebSphere Application Server는 IBM MQ 자원 어댑터를 임베드하므로, 이 환경에 수동으로 배치하지 않아도 됩니다.
- 클래스 로더 충돌을 방지하기 위해 동일한 Java 런타임 내 여러 애플리케이션에서 재배포 가능 JAR 파일을 번들로 제공하는 것은 권장되지 않습니다. 이 시나리오에서는 런타임의 Java 클래스패스에 재배포 가능한 IBM MQ JAR 파일을 사용할 수 있도록 하십시오.
- 애플리케이션에서 재배포 가능 JAR 파일을 번들로 제공하는 경우 재배포 가능 JAR 파일에서 설명한 대로, 모든 필수 JAR 파일을 포함해야 합니다. 애플리케이션 유지보수의 일환으로 번들 JAR 파일을 업데이트할 적절한 절차를 마련해야 합니다. 이를 통해 또는 IBM MQ classes for JMS 가 IBM MQ classes for Jakarta Messaging 최신 상태를 유지하고 알려진 문제가 재처리되도록 보장해야 합니다.

재배포 가능 JAR 파일

기업 내에서 다음 파일들은 실행이 필요한 시스템으로 IBM MQ classes for JMS 이동될 수 있습니다 IBM MQ classes for Jakarta Messaging:

- bcpkix-jdk15to18.jar [92 페이지의 『4』](#)
- bcpkix-jdk18on.jar [92 페이지의 『3』](#)
- bcprov-jdk15to18.jar [92 페이지의 『4』](#)
- bcprov-jdk18on.jar [92 페이지의 『3』](#)
- bcutil-jdk15to18.jar [92 페이지의 『4』](#)
- bcutil-jdk18on.jar [92 페이지의 『3』](#)
-  com.ibm.mq.allclient.jar [92 페이지의 『1』](#)
-  com.ibm.mq.jakarta.client.jar [92 페이지의 『2』](#)
- fscontext.jar
- jakarta.jms-api.jar
- jms.jar
- org.json.jar
- providerutil.jar

참고:

1. JMS 2.0 및 JMS 1.1
2. [Jakarta Messaging 3.0](#)
3. 시작 IBM MQ 9.4.0
4. 이전 IBM MQ 9.4.0

JMS JAR 파일

jms.jar 와 JMS 2.0 JMS 1.1 인터페이스를 포함합니다 - 이들은 로 명명됩니다 javax.jms.*.

JM 3.0 jakarta.jms-api.jar Jakarta Messaging 3.0 인터페이스를 포함합니다 - 이들은 다음과 같이 명명됩니다 jakarta.jms.*.

fscontext.jar 그리고 providerutil.jar

애플리케이션이 파일 시스템 컨텍스트를 사용하여 JNDI 검색을 수행하는 경우 fscontext.jar 및 providerutil.jar 파일이 필요합니다.

바운시 캐슬 보안 제공업체 및 CMS 지원 JAR 파일

Bouncy Castle 보안 제공자 및 CMS 지원 JAR 파일이 필요합니다. 자세한 정보는 [AMS로 비IBM JRE에 대한 지](#)원을 참조하십시오.

다음 JAR 파일이 필요합니다.

- bcpkix-jdk18on.jar
- bcprov-jdk18on.jar
- bcutil-jdk18on.jar

org.json.jar

해당 org.json.jar 파일이 필요합니다. 이것은 JSON 형식의 클라이언트 채널 정의 테이블(CCDT)을 파싱할 때 와 IBM MQ classes for Jakarta Messaging IBM MQ classes for JMS 에 의해 사용됩니다. 또한 객체와 com.ibm.mq.jms.MQSessioncom.ibm.mq.jms.MQConnection 객체에 대한 toString() 메서드를 호출하여 해당 객체의 JSON 표현을 생성할 때도 사용됩니다.

com.ibm.mq.allclient.jar 그리고 com.ibm.mq.jakarta.client.jar

파일 com.ibm.mq.allclient.jar 과 에는 IBM MQ classes for Jakarta MessagingIBM MQ classes for JMS,, IBM MQ classes for Java, PCF 및 com.ibm.mq.jakarta.client.jar 헤더 클래스가 포함되어 있습니다. 이 JAR 파일을 새 위치로 이동할 경우, 해당 새 위치에 최신 IBM MQ 수정 패키지를 지속적으로 적용하여 유지 관리되도록 조치를 취해야 합니다. 또한 임시 해결책을 적용하는 경우 해당 파일 사용 사실을 IBM 지원팀에 반드시 알려십시오.

파일 com.ibm.mq.allclient.jar 및 의 버전을 확인하려면 com.ibm.mq.jakarta.client.jar 다음 명령을 사용하십시오:

JM 3.0

```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
java -jar com.ibm.mq.allclient.jar
```

다음 예제는 이 명령의 샘플 출력을 표시합니다.

```
C:\Program Files\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:      Java Message Service Client
Version:   9.3.0.0
Level:     p000-L140428.1
Build Type: Production
Location:  file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:      WebSphere MQ classes for Java Message Service
Version:   9.3.0.0
Level:     p000-L140428.1
Build Type: Production
Location:  file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:      WebSphere MQ JMS Provider
Version:   9.3.0.0
Level:     p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
```



```

Location:    file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
Name:        Common Services for Java Platform, Standard Edition
Version:     9.3.0.0
Level:       p000-L140428.1
Build Type:  Production
Location:    file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

```

환경 변수 IBM MQ classes for JMS 설정 Jakarta Messaging

애플리케이션을 컴파일하고 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 실행하기 전에, 환경 **CLASSPATH** 변수 설정에 아카이브(JAR IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta MessagingJava) 파일이 포함되어야 합니다. 사용자의 요구사항에 따라 다른 JAR 파일을 클래스 경로에 추가해야 할 수 있습니다. 제공된 스크립트를 실행하려면 다른 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging환경 변수를 설정해야 합니다.

시작하기 전에

JM 3.0 IBM MQ 9.3.0부터, Jakarta Messaging 3.0 는 새로운 애플리케이션 개발에 지원됩니다. IBM MQ 9.3.0 그리고 향후에도 기존 애플리케이션에 대한 지원을 JMS 2.0 계속할 예정입니다. 동일한 애플리케이션에서 두 JMS 2.0Jakarta Messaging 3.0 API를 모두 사용하는 것은 지원되지 않습니다. 자세한 내용은 '[JMS/Jakarta Messaging용 IBM MQ 클래스 사용](#)'을 참조하십시오.

중요사항: 옵션을 Java 설정하여 `-Xbootclasspath` 를 포함하는 것은 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 지원되지 않습니다.

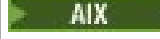
이 태스크 정보

애플리케이션을 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 컴파일하고 실행하려면 다음 표에 표시된 대로 플랫폼 및 Java 메시징 버전에 대한 설정을 **CLASSPATH** 사용하십시오. 또는 환경 변수를 사용하는 대신 명령줄에서 클래스 **java** 경로를 직접 지정할 수 있습니다.

JMS 2.0 설정에는 샘플 디렉터리가 포함되어 있으므로 IBM MQ classes for JMS, 샘플 IBM MQ classes for JMS 애플리케이션을 컴파일하고 실행할 수 있습니다.

JM 3.0 현재 IBM MQ classes for Jakarta Messaging 새로운 샘플을 준비 중입니다.

JM 3.0

표 7. CLASSPATH 애플리케이션 IBM MQ classes for Jakarta Messaging 컴파일 및 Jakarta Messaging 3.0 실행을 위한 설정	
플랫폼	CLASSPATH 설정
 AIX	CLASSPATH= MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jakarta.client.jar:
 Linux	CLASSPATH= MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jakarta.client.jar:
 IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar:
 Windows	CLASSPATH= MQ_INSTALLATION_PATH\java\lib\com.ibm.mq.jakarta.client.jar;
 z/OS	CLASSPATH= MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jakarta.client.jar;

JMS 2.0

표 8. **CLASSPATH** 애플리케이션 IBM MQ classes for JMS 컴파일 및 JMS 2.0 실행을 위한 설정, 샘플 애플리케이션 포함

플랫폼	CLASSPATH 설정
 AIX	CLASSPATH= MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.allclient.jar: MQ_INSTALLATION_PATH/samp/jms/samples:
 Linux	CLASSPATH= MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.allclient.jar: MQ_INSTALLATION_PATH/samp/jms/samples:
 IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: /QIBM/ProdData/mqm/java/samples/jms/samples:
 Windows	CLASSPATH= MQ_INSTALLATION_PATH\java\lib\com.ibm.mq.allclient.jar; MQ_INSTALLATION_PATH\tools\jms\samples;
 z/OS	CLASSPATH= MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.allclient.jar: MQ_INSTALLATION_PATH/java/samples/jms/samples:

이 표에서는 가 IBM MQ 설치된 상위 MQ_INSTALLATION_PATH 디렉터리를 나타냅니다.

JAR 파일의 매니페스트에는 애플리케이션에 IBM MQ classes for JMS 필요한

com.ibm.mq.jakarta.client.jar 또는 com.ibm.mq.allclient.jar 대부분의 다른 JAR 파일에 대한 참조가 포함되어 있으므로, 이러한 JAR 파일을 클래스 경로에 추가할 필요가 없습니다. 이러한 JAR 파일에는 JNDI(Java Naming Directory Interface)를 사용하여 디렉토리 서비스에서 관리 대상 오브젝트를 검색하는 애플리케이션과 JTA(Java Transaction API)를 사용하는 애플리케이션에서 필요한 파일이 포함됩니다.

그러나 다음과 같은 상황에서는 추가 JAR 파일을 클래스 경로에 포함시켜야 합니다.

- com.ibm.mq.exits 패키지에 정의된 인터페이스 대신에 com.ibm.mq 패키지에 정의된 채널 엑시트 인터페이스를 구현하는 채널 엑시트 클래스를 사용하는 경우에는 IBM MQ classes for Java JAR 파일, com.ibm.mq.jar을(를) 클래스 경로에 추가해야 합니다.
- 애플리케이션이 JNDI를 사용하여 디렉토리 서비스에서 관리 대상 오브젝트를 검색하는 경우에는 다음 JAR 파일도 클래스 경로에 추가해야 합니다.
 - fscontext.jar
 - providerutil.jar
- 애플리케이션이 JTA를 사용하면 클래스 경로에 jta.jar도 추가해야 합니다.

참고: 이러한 추가 JAR 파일은 애플리케이션 실행이 아닌 애플리케이션 컴파일에만 필요합니다.

제공된 스크립트는 다음 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 환경 변수를 사용합니다:

MQ JAVA DATA_PATH

이 환경 변수는 로그 및 추적 출력을 위한 디렉토리를 지정합니다.

MQ JAVA INSTALL_PATH

이 환경 변수는 IBM MQ classes for JMS가 설치되어 있는 디렉토리를 지정합니다.

MQ JAVA_LIB_PATH

이 환경 변수는 이전 표에 표시된 대로 IBM MQ classes for JMS 라이브러리가 저장되는 디렉토리를 지정합니다.

프로시저

- 

설치 후 Windows, 명령어를 IBM MQ 실행하십시오 **setmqenv**.

이 명령을 먼저 실행하지 않으면 명령을 **dspmqver** 실행할 때 다음과 같은 오류 메시지가 나타날 수 있습니다:

AMQ8351: IBM MQ Java environment has not been configured
올바르게 설치되었는지, 또는 IBM MQJRE 기능이 설치되지 않았는지 확인하십시오.

참고: 이 메시지는 (JRE) IBM MQJava runtime environment 을 설치하지 않은 경우 예상되는 것입니다([추가 Windows 기능 필수 조건 확인](#) 참조).

- Linux AIX

시스템에서 AIX and Linux 환경 변수를 직접 설정하십시오:

JMS 2.0 환경 변수를 설정하려면 다음 스크립트 중 JMS 2.0하나를 사용하십시오:

- 32비트 JVM 를 사용 중인 경우 해당 스크립트를 사용하십시오 **setjmsenv**.
- 64비트 JVM 를 또는 Linux AIX 시스템에서 사용하는 경우, 스크립트 를 **setjmsenv64**사용하십시오.

JM 3.0 환경 변수를 설정하려면 다음 스크립트 중 Jakarta Messaging 3.0하나를 사용하십시오:

- 32비트 JVM 를 사용 중인 경우 해당 스크립트를 사용하십시오 **setjms30env**.
- 64비트 JVM 를 사용 중인 경우 해당 스크립트를 사용하십시오 **setjms30env64**.

이러한 스크립트MQ_INSTALLATION_PATH/java/bin 디렉토리에 있습니다. 여기서 MQ_INSTALLATION_PATH는 IBM MQ가 설치된 상위 레벨 디렉토리를 나타냅니다.

이 스크립트들은 다양한 방식으로 사용할 수 있습니다. 해당 스크립트를 기반으로 표에 표시된 대로 필요한 환경 변수를 설정하거나, 텍스트 편집기를 사용하여 해당 변수를 .profile 추가할 수 있습니다. 일반적인 설정이 아닌 경우에는 필요에 따라 스크립트 콘텐츠를 편집하십시오. 또는 JMS 시동 스크립트가 실행될 모든 세션에서 스크립트를 실행할 수 있습니다. 이 옵션을 선택하면 검증 JMS 과정에서 시작하는 모든 셸 창에서 스크립트를 실행해야 합니다:

- **JMS 2.0** 를 입력하려면 또는 를 . ./setjmsenv JMS 2.0 ./setjmsenv64입력하십시오.
- **JM 3.0** 를 입력하려면 또는 를 . ./setjms30env Jakarta Messaging 3.0 . ./setjms30env64입력하십시오.

IBM i 날짜에 환경 변수 를 **QIBM_MULTI_THREADED IBM i**로 설정해야 합니다. 이렇게 하면 단일 스레드된 애플리케이션을 실행하는 것과 동일한 방식으로 멀티스레드된 애플리케이션을 실행할 수 있습니다. 자세한 내용은 [JMS를 사용한 Java IBM MQ 설정에서](#) 참조하십시오.

관련 태스크

118 페이지의 『IBM MQ classes for JMS 샘플 애플리케이션 사용』

IBM MQ classes for JMS 샘플 애플리케이션은 JMS API의 공통 기능 개요를 제공합니다. 이를 사용하여 설치 및 메시징 서버 설정을 확인하고 자신만의 애플리케이션을 빌드하는 데 도움을 받을 수 있습니다.

관련 참조

121 페이지의 『와 IBM MQ classes for JMS 함께 제공되는 스크립트Jakarta Messaging』

사용 IBM MQ classes for JMS그리고 IBM MQ classes for Jakarta Messaging 시 수행해야 하는 일반적인 작업을 지원하기 위해 여러 스크립트가 제공됩니다.

JNI(Java Native Interface) 라이브러리 구성

IBM MQ classes for JMS 바인딩 전송을 사용하여 큐 관리자에 연결하거나, 클라이언트 전송을 사용하여 큐 관리자에 연결하고 자바 Java이외의 언어로 작성된 채널 종료 프로그램을 사용하는 애플리케이션은 자바 Java 네이티브 인터페이스(JNI) 라이브러리에 접근할 수 있는 환경에서 실행되어야 합니다.

시작하기 전에

WebSphere Application Server 환경 사용에 대한 자세한 정보는 [기본 라이브러리 정보로 IBM MQ 메시징 제공자 구성](#)을 참조하십시오.

이 태스크 정보

이 환경을 설정하려면, 애플리케이션을 IBM MQ classes for JMS 시작하기 전에 (JVM)가 Java Virtual Machine JNI 라이브러리를 불러올 수 있도록 환경의 라이브러리 경로를 구성해야 합니다.

IBM MQ 다음과 같은 두 가지 Java JNI 라이브러리를 제공합니다:

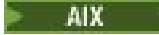
mqjbnd

이 라이브러리는 바인딩 전송을 사용하여 큐 관리자에 연결하는 애플리케이션에서 사용합니다. IBM MQ classes for JMS와 큐 관리자 사이의 인터페이스를 제공합니다. IBM MQ 10.0과 함께 설치된 mqjbnd 라이브러리는 IBM MQ 10.0(또는 이전) 큐 관리자에 연결하는 데 사용될 수 있습니다.

mqjexitstub02

Java이 라이브러리는 애플리케이션이 클라이언트 전송 방식을 사용하여 큐 관리자에 연결하고, 이외의 언어로 작성된 채널 종료 프로그램을 사용할 때 로드됩니다 IBM MQ classes for JMS .

각 플랫폼별 라이브러리의 위치는 표 1 에 나와 있습니다.

표 9. 각 플랫폼의 IBM MQ classes for JMS 라이브러리 위치	
플랫폼	JNI 라이브러리가 포함된 디렉터리
 AIX  Linux (POWER, x86-64 및 s390x 플랫폼)	MQ_INSTALLATION_PATH/java/lib64
 Windows	MQ_INSTALLATION_PATH\java\lib64
 z/OS	MQ_INSTALLATION_PATH/java/lib (31비트 및 64비트 라이브러리)

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

참고:  z/OS에서는 31비트 또는 64비트 JVM(Java Virtual Machine)을 사용할 수 있습니다. 사용할 JNI 라이브러리를 지정하지 않아도 됩니다. IBM MQ classes for JMS 자체에서 로드할 JNI 라이브러리를 판별합니다.

프로시저

1. JVM의 **java.library.path** 특성을 구성하십시오. 이는 두 가지 방법으로 수행될 수 있습니다.

- 다음 예에 표시된 대로 JVM 인수를 지정합니다.

```
-Djava.library.path=path_to_library_directory
```

 예를 들어, 기본 위치에 Linux 설치하는 JVM 의 경우 다음을 지정하십시오:

```
-Djava.library.path=/opt/mqm/java/lib64
```

- JVM이 자체 **java.library.path**을(를) 설정하도록 셸의 환경을 구성합니다. 이 경로는 IBM MQ를 설치한 위치 및 플랫폼에 따라 다릅니다. 예를 들어, 기본 IBM MQ 설치 위치를 지정하는 경우 다음 설정을 사용할 수 있습니다:

 `export LIBPATH=/usr/mqm/java/lib64:$LIBPATH`

 `export LD_LIBRARY_PATH=/opt/mqm/java/lib64:$LD_LIBRARY_PATH`

```
Windows set PATH=C:\Program Files\IBM\MQ\java\lib64;%PATH%
```

환경이 올바르게 구성되지 않은 경우 표시되는 예외 스택의 예는 다음과 같습니다.

```
Caused by: com.ibm.mq.jmqi.local.LocalMQ$4: CC=2;RC=2495;
AMQ8598: IBM MQ 네이티브 JNI 라이브러리 'mqjbnd'를 불러오지 못했습니다.
    at com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1268)
    at com.ibm.mq.jmqi.local.LocalMQ$1.run(LocalMQ.java:309)
    at java.security.AccessController.doPrivileged(AccessController.java:400)
    at com.ibm.mq.jmqi.local.LocalMQ.initialise_inner(LocalMQ.java:259)
    at com.ibm.mq.jmqi.local.LocalMQ.initialise(LocalMQ.java:221)
    at com.ibm.mq.jmqi.local.LocalMQ.<init>(LocalMQ.java:1350)
    at com.ibm.mq.jmqi.local.LocalServer.<init>(LocalServer.java:230)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
    at
    sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:86)
    at
    sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:58)
    at java.lang.reflect.Constructor.newInstance(Constructor.java:542)
    at com.ibm.mq.jmqi.JmqiEnvironment.getInstance(JmqiEnvironment.java:706)
    at com.ibm.mq.jmqi.JmqiEnvironment.getMQI(JmqiEnvironment.java:640)
    at
    com.ibm.msg.client.wmq.factories.WMQConnectionFactory.createV7ProviderConnection(WMQConnectionFactory.java:8437)
    ... 7 more
Caused by: java.lang.UnsatisfiedLinkError: mqjbnd (Not found in java.library.path)
    at java.lang.ClassLoader.loadLibraryWithPath(ClassLoader.java:1235)
    at java.lang.ClassLoader.loadLibraryWithClassLoader(ClassLoader.java:1205)
    at java.lang.System.loadLibrary(System.java:534)
    at com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1240)
    ... 20 more
```

2. 환경을 설정한 후, 다음 명령을 사용하여 애플리케이션을 IBM MQ classes for JMS 실행하십시오:

```
java application-name
```

여기서 *application-name* 는 실행할 애플리케이션의 IBM MQ classes for JMS 이름입니다.

Java runtime environment 32비트 환경에서 애플리케이션을 IBM MQ classes for JMS 실행할 경우, 32비트 환경이 지원되지 않고 필요한 JNI 라이브러리를 사용할 수 없기 때문에 Reason 코드 2495(MQRC_MODULE_NOT_FOUND)를 포함하는 IBM MQ 예외가 발생합니다 IBM MQ classes for JMS .

/Jakarta Messaging 구성 IBM MQ classes for JMS 파일

구성 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 파일은 구성에 사용되는 속성을 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 지정합니다.

참고: 구성 파일에 정의된 특성은 JVM 시스템 특성으로도 설정될 수 있습니다. 특성이 구성 파일에 설정되고 시스템 특성으로도 설정된 경우 시스템 특성이 우선합니다. 그러므로 필요한 경우 **java** 명령에서 시스템 특성으로 지정하여 구성 파일의 특성을 대체할 수 있습니다.

구성 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 파일의 형식은 표준 Java 속성 파일의 형식입니다. 설치 IBM MQ classes for JMS 디렉토리의 하위 bin 디렉토리에 라는 샘플 구성 **jms.config** 파일이 제공됩니다. 이 파일에서는 지원되는 모든 특성과 기본값을 문서화합니다.

구성 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 파일의 이름과 위치를 선택할 수 있습니다. 애플리케이션을 시작할 때 다음 형식의 **java** 명령을 사용하십시오.

```
java -Dcom.ibm.msg.client.config.location=config_file_url application_name
```

명령어에서는 구성 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 파일의 이름과 위치를 지정하는 균일 자원 위치 지정자(URLURL)입니다 *config_file_url* . http, file, ftp 및 jar 유형의 URL이 지원됩니다.

다음은 **java** 명령의 예입니다.

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config MyAppClass
```

이 명령은 구성 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 파일을 로컬 Windows 시스템의 D:\mydir\mjms.config 파일로 식별합니다.

애플리케이션이 시작될 때, 구성 파일의 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 내용을 읽고 지정된 속성을 내부 속성 저장소에 저장합니다. 명령어가 **java** 구성 파일을 식별하지 못하거나 구성 파일을 찾을 수 없는 경우, 모든 속성에 대해 기본값을 사용합니다 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging .

구성 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 파일은 애플리케이션과 큐 관리자 또는 브로커 간에 지원되는 모든 전송 방식과 함께 사용할 수 있습니다.

IBM MQ MQI client 구성 파일에 지정된 특성 대체

구성 IBM MQ MQI client 파일은 또한 를 구성하는 데 사용되는 속성을 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 지정할 수 있습니다. 그러나 IBM MQ MQI client 구성 파일에 지정된 특성은 애플리케이션이 클라이언트 모드로 큐 관리자에 연결할 때만 적용됩니다.

필요한 경우, 구성 IBM MQ MQI client 파일에서 속성을 지정하여 구성 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 파일의 모든 속성을 재정의할 수 있습니다. 속성을 재정의하려면 IBM MQ MQI client 설정 파일에는 다음 형식의 항목을 사용하십시오. IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 설정 파일:

```
com.ibm.mq.cfg.<stanza>.<propName>=<propValue>
```

항목의 변수는 다음을 의미합니다.

stanza

속성을 포함하는 IBM MQ MQI client 구성 파일의 스탠자 이름

propName

IBM MQ MQI client 구성 파일에 지정된 속성의 이름

propValue

IBM MQ MQI client 구성 파일에 지정된 속성 값을 대체하는 특성 값

예를 들어, 속성을 CIntSndBuffSize ` 65KB ` 로 설정하려면 구성 파일에 다음 항목을 추가하십시오:

```
com.ibm.mq.cfg.TCP.CIntSndBuffSize=65536
```

또는 구성 IBM MQ MQI client 파일의 속성을 재정의하려면 해당 속성을 **java** 명령줄에서 시스템 속성으로 지정할 수 있습니다. 이전 형식을 사용하여 특성을 시스템 특성으로 지정하십시오.

구성 IBM MQ MQI client 파일 내의 다음 속성들만이 관련이 있습니다 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging. 기타 속성을 지정하거나 대체하는 경우 효과가 없습니다. 특히, 클라이언트 구성 파일의 CHANNELS 스탠자 내 ChannelDefinitionDirectory 와 ChannelDefinitionFile 는 사용되지 않음을 유의하십시오. CCDT를 사용하는 방법에 대한 자세한 내용은 을 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 262 페이지의 『IBM MQ classes for JMS에서 클라이언트 채널 정의 테이블 사용』 참조하십시오.

표 10. 클라이언트 구성 파일의 스탠자에 포함되어 있는 속성	
스탠자	속성
클라이언트 구성 파일의 CHANNELS 스탠자	Put1DefaultAlwaysSync
클라이언트 구성 파일의 CHANNELS 스탠자	DefRecon
클라이언트 구성 파일의 CHANNELS 스탠자	ReconDelay
클라이언트 구성 파일의 CHANNELS 스탠자	PasswordProtection
클라이언트 구성 파일의 ClientExitPath 스탠자	ExitsDefaultPath
클라이언트 구성 파일의 ClientExitPath 스탠자	ExitsDefaultPath64
클라이언트 구성 파일의 ClientExitPath 스탠자	JavaExitsClasspath

표 10. 클라이언트 구성 파일의 스탠자에 포함되어 있는 속성 (계속)	
스탠자	속성
클라이언트 구성 파일의 JMQI 스탠자	useMQCSPauthentication
클라이언트 구성 파일의 MessageBuffer 스탠자	MaximumSize
클라이언트 구성 파일의 MessageBuffer 스탠자	PurgeTime
클라이언트 구성 파일의 MessageBuffer 스탠자	UpdatePercentage
클라이언트 구성 파일의 TCP 스탠자	ClntRcvBuffSize
클라이언트 구성 파일의 TCP 스탠자	ClntSndBuffSize
클라이언트 구성 파일의 TCP 스탠자	Connect_Timeout
클라이언트 구성 파일의 TCP 스탠자	KeepAlive

구성 IBM MQ MQI client 관련 자세한 내용은 다음을 참조하십시오. [IBM MQ MQI client 구성 파일, mqclient.ini](#)

Java JMS trace에 대한 표준 환경 추적 스탠자

표준 환경 추적 Java 설정 구문을 사용하여 및 IBM MQ classes for Jakarta Messaging IBM MQ classes for JMS 추적 기능을 구성하십시오.

com.ibm.msg.client.commonservices.trace.outputName = traceOutputName

V 10.0.0에서 IBM MQ 10.0.0의 `traceOutputName` 값은 추적 출력이 전송될 위치를 지정하며, 다음 위치 중 하나일 수 있습니다:

stdout (또는 System.out)

추적 출력을 표준 출력 스트림으로 전송합니다(대소문자 구분 없음).

stderr (또는 System.err)

추적 출력을 표준 오류 스트림으로 전송합니다(대소문자 구분 없음).

<file>

그 외의 값은 파일 경로로 간주되며, 해당 파일로 추적이 수행됩니다.

기본적으로 추적 파일은 애플리케이션의 현재 작업 디렉토리에 있는 추적 파일에 기록됩니다. 추적 파일의 이름은 애플리케이션이 실행 중인 환경에 따라 다릅니다.

- `mqjavaclient_%PID%.cl%u.trc` 애플리케이션이 재배포 가능한 JAR 파일 `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0)에서 IBM MQ classes for Jakarta Messaging 로드했거나, 재배포 가능한 JAR 파일 `com.ibm.mq.allclient.jar` (JMS 2.0)에서 IBM MQ classes for JMS 로드한 경우, 추적 정보는 라는 파일에 기록됩니다.
- 애플리케이션이 JAR 파일에서 IBM MQ classes for JMS 로드한 경우, 추적 `com.ibm.mqjms.jar` 정보는 라는 `mqjava_%PID%.cl%u.trc` 파일에 기록됩니다.

여기서, `%PID%`는 추적하는 애플리케이션의 프로세스 ID이고, `%u`는 서로 다른 Java 클래스 로더에서 추적을 실행하는 스레드 간 파일을 구별하기 위한 고유한 숫자입니다. 프로세스 ID가 사용 불가능한 경우, 난수가 생성되며 문자 `f`로 접두부가 지정됩니다. `%PID%`지정한 파일 이름에 프로세스 ID를 포함하려면 다음 문자열을 사용하십시오.

대체 디렉토리를 지정하는 경우, 이는 존재해야 하며 사용자는 이 디렉토리에 대한 쓰기 권한이 있어야 합니다. 쓰기 권한이 없으면 추적 출력이 `System.err`에 쓰여집니다.

V 10.0.0에서 속성은 **com.ibm.msg.client.commonservices.trace.status** 여전히 추적 기능의 활성화 여부를 제어하지만, **com.ibm.msg.client.commonservices.trace.outputName**가 `stdout` 또는 `stderr`로 설정되어 있고 지정된 스트림에 접근할 수 있는 경우, 클라이언트 추적을 위한 다음 파일 기반 구성 속성은 사용되지 않습니다:

- **com.ibm.msg.client.commonservices.trace.limit**

- **com.ibm.msg.client.commonservices.trace.count**
- **com.ibm.msg.client.commonservices.trace.append**
- **com.ibm.msg.client.commonservices.trace.compress**

com.ibm.msg.client.commonservices.trace.include = *includeList*

*includeList*는 추적되는 패키지 및 클래스의 목록이거나 특수 값 ALL 또는 NONE입니다.

패키지 또는 클래스 이름을 세미콜론 ;으로 분리하십시오. *includeList* 기본값은 ALL이며, 또는 IBM MQ classes for JMS 내의 모든 패키지와 IBM MQ classes for Jakarta Messaging 클래스를 추적합니다.

참고: 패키지를 포함하지만 해당 패키지의 서브패키지를 제외시킬 수 있습니다. 예를 들어, 패키지 a.b는 포함하지만 패키지 a.b.x는 포함하지 않은 경우에 추적에는 a.b.y 및 a.b.z의 모든 것이 포함되지만 a.b.x 또는 a.b.x.1은 포함되지 않습니다.

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*

*excludeList*는 추적되는 패키지 및 클래스의 목록이거나, 특수 값 ALL 또는 NONE입니다.

패키지 또는 클래스 이름을 세미콜론 ;으로 분리하십시오. *excludeList* 기본값은 NONE로 설정되어 있으므로, IBM MQ classes for JMS 또는 내의 패키지와 IBM MQ classes for Jakarta Messaging 클래스가 추적 대상에서 제외되지 않습니다.

참고: 패키지를 제외하지만 해당 패키지의 서브패키지를 포함시킬 수 있습니다. 예를 들어, 패키지 a.b는 제외하지만 패키지 a.b.x는 포함하는 경우에 추적에는 a.b.x 및 a.b.x.1의 모든 것이 포함되지만 a.b.y 또는 a.b.z는 포함되지 않습니다.

동일 레벨에서 포함됨 및 제외됨 모두로서 지정된 임의의 패키지 또는 클래스가 포함됩니다.

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayBytes*

*maxArrayBytes*는 바이트 배열에서 추적된 최대 바이트 수입니다.

*maxArrayBytes*가 양의 정수로 설정된 경우, 이는 추적 파일에 쓰여지는 바이트 배열의 바이트 수를 제한합니다. 이는 *maxArrayBytes* 쓰기 이후 바이트 배열을 자릅니다. *maxArrayBytes*를 설정하면 결과 추적 파일의 크기가 줄어들며, 애플리케이션의 성능에서 추적의 효과가 줄어듭니다.

이 특성에 대한 0의 값은 바이트 배열의 콘텐츠가 추적 파일에 송신되지 않음을 의미합니다.

기본값은 -1이며, 이는 추적 파일에 송신된 바이트 배열에서 바이트 수의 한계를 제거합니다.

com.ibm.msg.client.commonservices.trace.limit = *maxTraceBytes*

*maxTraceBytes*는 추적 출력 파일에 쓰여진 최대 바이트 수입니다.

*maxTraceBytes*는 *traceCycles*와 함께 작동됩니다. 작성된 추적 바이트 수가 한계에 근접하면 파일이 닫히며 새 추적 출력 파일이 시작됩니다.

0 값은 추적 출력 파일의 길이가 0임을 의미합니다. 기본값은 -1이며, 이는 추적 출력 파일에 쓰여지는 데이터의 양이 무제한임을 의미합니다.

com.ibm.msg.client.commonservices.trace.count = *traceCycles*

*traceCycles*는 순환되는 추적 출력 파일의 수입니다.

현재 추적 출력 파일이 *maxTraceBytes*에서 지정된 한계에 도달하면 파일이 닫힙니다. 추가 추적 출력은 시퀀스의 다음 추적 출력 파일에 쓰여집니다. 각각의 추적 출력 파일은 파일 이름에 추가된 숫자 접미부로 구분됩니다. 현재 또는 최신 추적 출력 파일은 mqjms.trc.0이며, 다음의 최신 추적 출력 파일은 mqjms.trc.1입니다. 보다 이전의 추적 파일은 한계에 도달할 때까지 동일한 번호 지정 패턴을 따릅니다.

*traceCycles*의 디폴트 값은 1입니다. *traceCycles*가 1인 경우, 현재 추적 출력 파일이 최대 크기에 도달할 때 파일이 닫히고 삭제됩니다. 동일한 이름의 새 추적 출력 파일이 시작됩니다. 따라서 한 번에 하나의 추적 출력 파일만이 존재합니다.

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*

*traceParameters*는 메소드 매개변수 및 리턴 값이 추적에 포함되는지 여부를 제어합니다.

*traceParameters*의 기본값은 TRUE입니다. *traceParameters*가 FALSE로 설정되면 메소드 서명만 추적됩니다.

com.ibm.msg.client.commonservices.trace.startup = startup

초기화 단계 동안 IBM MQ classes for Jakarta Messaging 와 IBM MQ classes for JMS 에 대한 리소스가 할당됩니다. 기본 추적 기능은 자원 할당 단계 중에 초기화됩니다.

*startup*이 TRUE로 설정되면 시동 추적이 사용됩니다. 추적 정보가 즉시 생성되며, 추적 기능 자체를 포함한 모든 컴포넌트의 설정이 포함됩니다. 시동 추적 정보를 사용하여 구성 문제점을 진단할 수 있습니다. 시동 추적 정보는 항상 *System.err*에 쓰여집니다.

*startup*의 기본값은 FALSE입니다.

*startup*은 초기화가 완료되기 전에 확인됩니다. 이러한 이유 때문에, Java 시스템 특성으로서 명령행에서 특성만 지정합니다. 또는 IBM MQ classes for Jakarta Messaging 구성 IBM MQ classes for JMS 파일에서 이를 지정하지 마십시오.

com.ibm.msg.client.commonservices.trace.compress = compressedTrace

*compressedTrace*를 TRUE로 설정하여 추적 출력을 압축할 수 있습니다.

*compressedTrace*의 기본값은 FALSE입니다.

*compressedTrace*가 TRUE로 설정되면 추적 출력이 압축됩니다. 기본 추적 출력 파일 이름의 확장자는 *.trz*입니다. 압축이 기본값인 FALSE로 설정되면, 파일의 확장자는 *.trc*이며 이는 압축되지 않음을 표시합니다. 그러나 추적 출력의 파일 이름이 *traceOutputName*에 지정되면 해당 이름이 대신 사용됩니다. 파일에는 접미부가 적용되지 않습니다.

압축 추적 출력은 압축되지 않은 추적 출력보다 크기가 작습니다. 입/출력이 적으므로 압축되지 않은 추적보다 빨리 기록될 수 있습니다. 압축된 추적은 압축되지 않은 IBM MQ classes for Jakarta Messaging 추적보다 와 IBM MQ classes for JMS 의 성능에 미치는 영향이 적습니다.

maxTraceBytes 및 *traceCycles*가 설정된 경우, 다중 압축 추적 파일이 다중 플랫폼 파일 대신에 작성됩니다.

또는 이 제어되지 않은 IBM MQ classes for JMS IBM MQ classes for Jakarta Messaging 방식으로 종료될 경우, 압축된 추적 파일이 유효하지 않을 수 있습니다. 따라서 트레이스 압축은 또는 IBM MQ classes for Jakarta Messaging 이 통제된 방식으로 종료될 때에만 IBM MQ classes for JMS 사용해야 합니다. 검사 중인 문제점 때문에 JVM 자체가 예상치 못하게 중지되지 않는 경우에만 추적 압축을 사용하십시오.

System.Halt() 종료 또는 비정상적 제어되지 않은 JVM 종료를 발생시킬 수 있는 문제점을 진단 중인 경우에는 추적 압축을 사용하지 마십시오.

com.ibm.msg.client.commonservices.trace.level = traceLevel

*traceLevel*은 추적의 필터링 레벨을 지정합니다. 정의된 추적 레벨은 다음과 같습니다.

- TRACE_NONE: 0
- TRACE_EXCEPTION: 1
- TRACE_WARNING: 3
- TRACE_INFO: 6
- TRACE_ENTRYEXIT: 8
- TRACE_DATA: 9
- TRACE_ALL: Integer.MAX_VALUE

각 추적 레벨에는 모든 하위 레벨이 포함됩니다. 예를 들어, 추적 레벨이 TRACE_INFO에서 설정된 경우에 TRACE_EXCEPTION, TRACE_WARNING 또는 TRACE_INFO의 정의된 레벨이 있는 추적 지점이 추적에 쓰여집니다. 다른 모든 추적 지점은 제외됩니다.

com.ibm.msg.client.commonservices.trace.standalone = standaloneTrace

*standaloneTrace*는 IBM MQ JMS 클라이언트 추적 서비스가 WebSphere Application Server 환경에서 사용되는지 여부를 제어합니다.

*standaloneTrace*가 TRUE로 설정된 경우에는 IBM MQ JMS 클라이언트 추적 특성을 사용하여 추적 구성을 판별합니다.

`standaloneTrace`가 FALSE로 설정되어 있으며 IBM MQ JMS 클라이언트가 WebSphere Application Server 컨테이너에서 실행 중인 경우에는 WebSphere Application Server 추적 서비스가 사용됩니다. 생성되는 추적 정보는 애플리케이션 서버의 추적 설정에 따라 다릅니다.

`standaloneTrace`의 기본값은 FALSE입니다.

관련 태스크

[IBM MQ classes for JMS 애플리케이션 추적](#)

로깅 스탠자

로깅 스탠자를 사용하여 IBM MQ classes for JMS 로그 기능을 구성할 수 있습니다.

다음 특성이 로깅 스탠자에 포함될 수 있습니다.

`com.ibm.msg.client.commonservices.log.outputName = path`

IBM MQ classes for JMS 로그 기능에서 사용하는 로그 파일의 이름입니다. 기본값은 `mqjms.log`이며, IBM MQ classes for JMS이(가) 실행 중인 Java Runtime Environment의 현재 작업 디렉토리에 기록됩니다.

특성은 다음 값 중 하나를 취할 수 있습니다.

- 단일 경로 이름
- 경로 이름의 쉼표로 구분된 목록(모든 데이터가 모든 파일에 로그됨)

각 경로 이름은 절대 또는 상대 경로 이름이거나 다음일 수 있습니다.

"stderr" 또는 "System.err"

표준 오류 스트림을 표시합니다.

"stdout" 또는 "System.out"

표준 출력 스트림을 표시합니다.

`com.ibm.msg.client.commonservices.log.maxBytes`

메시지 데이터를 로깅하기 위한 호출에서 로깅되는 최대 바이트 수입니다.

양의 정수

데이터는 로그 호출마다 해당 바이트 값까지 쓰여집니다.

0

데이터가 쓰여지지 않습니다.

-1

무제한 데이터가 쓰여집니다(기본값).

`com.ibm.msg.client.commonservices.log.limit`

하나의 로그 파일에 쓰여진 최대 바이트 수입니다(기본값은 262144).

양의 정수

데이터는 로그 파일마다 해당 바이트 값까지 쓰여집니다.

0

데이터가 쓰여지지 않습니다.

-1

무제한 데이터가 쓰여집니다.

`com.ibm.msg.client.commonservices.log.count`

순환되는 로그 파일의 수입니다. 각 파일이 `com.ibm.msg.client.commonservices.trace.limit`에 도달하면 다음 파일에서 추적이 시작되며 기본값은 3입니다.

양의 정수

순환되는 파일의 수입니다.

0

단일 파일입니다.

Java SE Specifics 스탠자

Java SE Specifics 스탠자를 사용하여 IBM MQ classes for JMS가 Java Standard Edition 환경에서 사용 중일 때 사용되는 특성을 구성할 수 있습니다.

com.ibm.msg.client.commonservices.j2se.produceJavaCore = TRUE | FALSE

IBM MQ classes for JMS 가 FDC 파일을 생성한 후 즉시 Java코어 파일이 기록되는지 여부를 판별합니다. TRUE로 설정되면 IBM MQ classes for JMS 가 실행 중인 JRE (Java Runtime Environment) 의 작업 디렉토리에 Java코어 파일이 생성됩니다.

true

Java Runtime Environment의 해당 수행 기능에 따라 JavaCore를 생성합니다.

False

JavaCore를 생성하지 않습니다. 이는 기본값입니다.

IBM MQ 특성 스탠자

IBM MQ 특성 스탠자를 사용하여 IBM MQ classes for JMS가 IBM MQ와 상호작용하는 방법에 영향을 주는 특성을 설정할 수 있습니다.

com.ibm.msg.client.wmq.compat.base.internal.MQQueue.smallMsgsBufferReductionThreshold

IBM MQ classes for JMS를 사용 중인 애플리케이션이 IBM MQ 메시징 제공자 마이그레이션 모드를 사용하여 IBM MQ 큐 관리자에 연결 중인 경우, IBM MQ classes for JMS는 메시지 수신 시에 4KB의 기본 버퍼 크기를 사용합니다. 애플리케이션이 가져오고자 하는 메시지가 4KB를 초과하는 경우, IBM MQ classes for JMS는 메시지를 수용하기에 충분히 크도록 버퍼 크기를 조정합니다. 그리고 후속 메시지가 수신되면 보다 큰 버퍼 크기가 사용됩니다.

이 특성은 버퍼 크기가 다시 4KB로 감소되는 시점을 제어합니다. 기본적으로, 보다 큰 버퍼 크기보다 작은 10개의 연속 메시지를 수신하는 경우 버퍼 크기는 다시 4KB로 줄어듭니다. 메시지가 수신될 때마다 버퍼 크기를 다시 4KB로 재설정하려면 특성을 0 값으로 설정하십시오.

0

버퍼가 항상 기본값으로 재설정됩니다.

10

이는 기본값입니다. 10번째 메시지 이후 버퍼 크기가 조정됩니다.

com.ibm.msg.client.wmq.receiveConversionCCSID

IBM MQ classes for JMS을(를) 사용 중인 애플리케이션이 IBM MQ 메시징 제공자 정상 모드를 사용하여 IBM MQ 큐 관리자에 연결하는 경우, receiveConversionCCSID 특성을 설정하여 큐 관리자에서 메시지를 수신하는 데 사용되는 MQMD 구조의 기본 CCSID 값을 대체할 수 있습니다. 기본적으로 MQMD에는 1208로 설정된 CCSID가 포함되지만, 이는 예를 들어 큐 관리자가 메시지를 이 코드 페이지로 변환할 수 없는 경우 변경될 수 있습니다.

올바른 값은 올바른 CCSID 번호 또는 다음 값 중 하나입니다.

-1

플랫폼 기본값을 사용합니다.

1208

이는 기본값입니다.

클라이언트 모드 특정 스탠자

클라이언트 모드 특정 스탠자를 사용하여 IBM MQ classes for JMS가 CLIENT 전송을 사용 중인 큐 관리자에 연결할 때 사용되는 특성을 지정할 수 있습니다.

com.ibm.mq.polling.RemoteRequestEntry

큐 관리자의 응답을 대기할 때 중단된 연결을 확인하기 위해 IBM MQ classes for JMS가 사용하는 폴링 간격을 지정합니다.

양의 정수

확인하기 전에 대기하는 시간(밀리초)입니다. 기본값은 10000 또는 10초입니다. 최소값은 3000이며, 더 낮은 값은 이 최소값과 동일한 방법으로 처리됩니다.

JMS 클라이언트 동작을 구성하는 데 사용된 특성
JMS 클라이언트의 동작을 구성하려면 이러한 특성을 사용하십시오.

com.ibm.mq.jms.SupportMQExtensions TRUE|FALSE

JMS 2.0 스펙은 특정 작동이 작업하는 방식에 대한 변경사항을 소개합니다. IBM MQ 8.0에는 변경된 동작을 이전 구현으로 되돌리기 위해 **TRUE**로 설정할 수 있는 **com.ibm.mq.jms.SupportMQExtensions** 특성이 포함되어 있습니다. JMS 2.0 애플리케이션 및 JMS 1.1 API를 사용하되 IBM MQ 8.0 IBM MQ classes for JMS에 대해 실행되는 일부 애플리케이션에 대해서도 변경된 동작을 일부 되돌려야 할 수 있습니다.

TRUE

SupportMQExtensions를 TRUE로 설정하여 다음 세 가지 기능 영역으로 돌아갑니다.

메시지 우선순위

메시지에는 우선순위 0 - 9가 지정될 수 있습니다. JMS 2.0 이전에는 메시지가 큐의 기본 우선순위가 사용됨을 나타내는 값 -1도 사용할 수 있었습니다. JMS 2.0에서는 메시지 우선순위 -1을 허용하지 않습니다. SupportMQExtensions를 켜면 -1 값을 사용할 수 있습니다.

클라이언트 ID

JMS 2.0 스펙에서는 연결을 작성할 때 널이 아닌 클라이언트 ID의 고유성을 확인하도록 요구합니다. SupportMQExtensions를 켜면 이 요구사항이 무시되며 클라이언트 ID를 재사용할 수 있음을 의미합니다.

NoLocal

JMS 2.0 스펙에서는 이 상수가 켜질 때 이용자가 동일한 클라이언트 ID에 의해 발행되는 메시지를 수신할 수 없어야 합니다. JMS 2.0 이전에는 자체 연결에 의해 발행되는 메시지의 수신을 방지할 수 있도록 이 속성이 구독자에서 설정되었습니다. SupportMQExtensions를 켜면 이 작동이 이전 구현으로 되돌아갑니다.

FALSE

동작의 변경사항이 유지됩니다.

com.ibm.msg.client.jms.ByteStreamReadOnlyAfterSend= TRUE|FALSE

IBM MQ 8.0.0 Fix Pack 2부터는 애플리케이션이 바이트 또는 스트림 메시지를 전송하고 나면 IBM MQ classes for JMS가 방금 전송한 메시지의 상태를 읽기 전용 또는 쓰기 전용으로 설정할 수 있습니다.

TRUE

오브젝트는 송신된 후에 읽기 전용으로 설정됩니다. 이 값을 설정하면 JMS 2.0 스펙과의 호환성을 유지합니다.

FALSE

오브젝트는 송신된 후에 쓰기 전용으로 설정됩니다. 이는 기본값입니다.

관련 개념

302 페이지의 『SupportMQExtensions 특성』

해당 JMS 2.0 사양은 특정 동작 방식에 변경 사항을 도입했습니다. IBM MQ 8.0 그리고 이후에는 속성을 포함하는데 **com.ibm.mq.jms.SupportMQExtensions**, 이 속성을 TRUE로 설정하면 변경된 동작들을 이전 구현 방식으로 되돌릴 수 있습니다.

'z/OS'의 'IBM MQ classes for JMS'에 대한 STEPLIB 구성

z/OS에서 런타임 시 사용된 STEPLIB에는 IBM MQ SCSQAUTH 및 SCSQANLE 라이브러리가 포함되어야 합니다. 시작 JCL에 또는 .profile 파일을 사용하여 이러한 라이브러리를 지정하십시오.

'z/OS UNIX System Services'에서 다음 코드 스니펫과 같이 '.profile'의 한 줄을 사용하여 'thlqual'을 'IBM MQ'를 설치할 때 선택한 상위 수준 데이터 집합 한정자로 대체하여 추가할 수 있습니다:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

다른 환경에서는 일반적으로 STEPLIB 연결에 SCSQAUTH 및 SCSQANLE를 포함하도록 시동 JCL을 편집해야 합니다.

```
STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR  
         DD DSN=thlqual.SCSQANLE,DISP=SHR
```

IBM MQ classes for JMS 및 소프트웨어 관리 도구

Apache Maven과 같은 소프트웨어 관리 도구는 IBM MQ classes for JMS 및 IBM MQ classes for Jakarta Messaging와 함께 사용할 수 있습니다.

많은 대형 개발 조직은 이러한 도구를 사용하여 써드파티 라이브러리의 저장소를 중앙 집중식으로 관리합니다.

와 IBM MQ classes for JMS 는 여러 개의 JAR IBM MQ classes for Jakarta Messaging 파일로 구성되어 있습니다. 이 API를 사용하여 Java 언어 애플리케이션을 개발 중인 경우에는 IBM MQ Server, IBM MQ Client 또는 IBM MQ Client SupportPac의 설치가 애플리케이션이 개발되는 시스템에서 필요합니다.

해당 도구를 사용하고 IBM MQ classes for JMS를 구성하는 JAR 파일을 중앙 관리되는 저장소에 추가하려면 다음과 같은 점을 관찰해야 합니다.

- 저장소 또는 컨테이너를 조직 내의 개발자만 사용할 수 있어야 합니다. 조직 외부의 배포는 허용되지 않습니다.
- 저장소는 단일 IBM MQ 릴리스 또는 수정팩에서 JAR 파일의 전체 일관된 세트를 포함해야 합니다.
- 사용자는 IBM 지원에서 제공하는 유지보수로 저장소를 업데이트해야 합니다.

다음 JAR 파일들을 저장소에 설치해야 합니다:

- **JMS 2.0** com.ibm.mq.allclient.jar 사용하는 jms.jar 경우 IBM MQ classes for JMS필수입니다.
- **JM 3.0** com.ibm.mq.jakarta.client.jar 사용하는 jakarta.jms-api.jar 경우 IBM MQ classes for Jakarta Messaging필수입니다.
- fscontext.jar 또는 IBM MQ classes for Jakarta Messaging 를 IBM MQ classes for JMS 사용 중이며 파일 시스템 JNDI 컨텍스트에 저장된 관리 대상 객체에 JMS 접근하는 경우 필수입니다.
- providerutil.jar 또는 IBM MQ classes for JMS 를 사용 IBM MQ classes for Jakarta Messaging 중이며 파일 시스템 JNDI 컨텍스트에 저장된 관리 대상 객체에 JMS 접근하는 경우 필수입니다.
- 바운시 캐슬 보안 공급자 및 CMS 지원 JAR 파일은 IBM 이외의 JRE 지원에 필요합니다. 자세한 정보는 [비IBM JRE](#) 지원을 참조하십시오.

아래에서 애플리케이션 IBM MQ classes for JMS 실행 Java security manager

IBM MQ classes for JMS는 Java security manager를 사용한 상태에서 실행할 수 있습니다. Java security manager를 사용한 상태로 애플리케이션을 실행하려면 적절한 정책 구성 파일로 JVM(Java Virtual Machine)을 구성해야 합니다.

적합한 정책 정의 파일을 생성하는 가장 간단한 방법은 제공된 Java runtime environmentJRE 정책 구성 파일을 수정하는 것입니다. 대부분의 시스템에서 이 파일은 디렉터리 JRElib/security/java.policy 상대 경로로 디렉터리에 위치합니다. 정책 구성 파일은 선호하는 편집기를 사용하거나, 제공된 정책 도구 프로그램을 사용하여 JRE 편집할 수 있습니다.

예제 정책 구성 파일

다음은 IBM MQ classes for JMS가 기본 보안 관리자에서 성공적으로 실행되도록 허용하는 정책 구성 파일의 예입니다. 특정 파일과 디렉토리의 위치를 지정하기 위해 이 파일을 사용자 정의해야 합니다.

MQ_INSTALLATION_PATH는 IBM MQ가 설치된 상위 레벨 디렉터리이고, MQ_DATA_DIRECTORY는 MQ 데이터 디렉토리의 위치이며 QM_NAME은 액세스가 구성되는 큐 관리자의 이름입니다.

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/*" {
    //We need access to these properties, mainly for tracing
    permission java.util.PropertyPermission "user.name","read";
    permission java.util.PropertyPermission "os.name","read";
    permission java.util.PropertyPermission "user.dir","read";
    permission java.util.PropertyPermission "line.separator","read";
    permission java.util.PropertyPermission "path.separator","read";
    permission java.util.PropertyPermission "file.separator","read";
    permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
    permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
    permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
    permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
    permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

    //Tracing - we need the ability to control java.util.logging
    permission java.util.logging.LoggingPermission "control";
}
```



```
// And access to create the trace file and read the log file - assumed to be in the current
directory
permission java.io.FilePermission "*" ,"read,write";

// We'd like to set up an mBean to control trace
permission javax.management.MBeanServerPermission "createMBeanServer";
permission javax.management.MBeanPermission "*" ,"*";

// We need to be able to read manifests etc from the jar files in the installation directory
permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-" ,"read";

//Required if mqclient.ini/mqs.ini configuration files are used
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini" ,"read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini" ,"read";

//For the client transport type.
permission java.net.SocketPermission "*" ,"connect,resolve";

//For the bindings transport type.
permission java.lang.RuntimePermission "loadLibrary.*";

//For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB" ,"read";

//For applications that use User Exits
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*" ,"read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*" ,"read";
permission java.lang.RuntimePermission "createClassLoader";

//Required for the z/OS platform
permission java.util.PropertyPermission "com.ibm.vm.bitmode" ,"read";

// Used by the internal ConnectionFactory implementation
permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

// Used for controlled class loading
permission java.lang.RuntimePermission "setContextClassLoader";

// Used to default the Application name in Client mode connections
permission java.util.PropertyPermission "sun.java.command" ,"read";

// Used by the IBM JSSE classes
permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace" ,"read";

//Required to determine if an IBM Java Runtime is running in FIPS mode,
//and to modify the property values status as required.
permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider" ,"read,write";
permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS" ,"read,write";
//Required if an IBM FIPS provider is to be used for SSL communication.
permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

// Required for non-IBM Java Runtimes that establish secure client
// transport mode connections using mutual TLS authentication
permission java.util.PropertyPermission "javax.net.ssl.keyStore" ,"read";
permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword" ,"read";
};
```

이 예에서 grant문은 IBM MQ classes for JMS에 필요한 권한을 포함합니다. 정책 구성 파일에서 이 grant문을 사용하려면 IBM MQ classes for JMS를 설치한 위치 및 애플리케이션을 저장하는 위치에 따라 경로 이름을 수정해야 합니다.

IBM MQ classes for JMS와 함께 제공되는 샘플 애플리케이션과 해당 애플리케이션을 실행하는 스크립트는 보안 관리자를 사용으로 설정하지 않습니다.

중요사항:

IBM MQ classes for JMS 추적 기능은 시스템 특성에 대한 추가 조회를 수행하고 추가 파일 시스템 작업도 수행하므로 추가 권한이 필요합니다.

추적이 사용으로 설정된 보안 관리자에서 실행하기에 적합한 템플릿 보안 정책 파일은 IBM MQ 설치의 samples/wmqjava 디렉토리에 example.security.policy(으)로 제공됩니다.

IBM MQ classes for JMS 애플리케이션의 설치 후 설정

이 주제에서는 IBM MQ classes for JMS 애플리케이션이 큐 관리자의 자원에 액세스하기 위해 필요한 사항을 설명합니다. 또한 연결 방식을 소개하고 애플리케이션이 클라이언트 모드에서 연결할 수 있도록 큐 관리자를 구성하는 방법을 설명합니다.

IBM MQ readme 파일을 확인하십시오. 이 주제의 정보를 대체하는 정보를 포함할 수 있습니다.

권한이 없는 사용자를 위해 권한을 부여해야 하는 JMS에서 사용하는 오브젝트
권한이 없는 사용자가 JMS에서 사용하는 큐에 액세스하려면 권한을 부여받아야 합니다. 모든 JMS 애플리케이션은 작업할 큐 관리자에 대한 권한을 부여받아야 합니다.

IBM MQ의 액세스 제어에 대한 자세한 내용은 [보안 설정](#)을 참조하십시오.

IBM MQ classes for JMS 애플리케이션은 큐 관리자에 대한 connect 및 inq 권한이 필요합니다. **setmqaut** 제어 명령을 사용하여 적절한 권한을 설정할 수 있습니다. 예를 들어 다음과 같습니다.

```
setmqaut -m QM1 -t qmgr -g jmsappsgroup +connect +inq
```

포인트-투-포인트 도메인에는 다음 권한이 필요합니다.

- MessageProducer 오브젝트에서 사용한 큐에는 put 권한이 필요합니다.
- MessageConsumer 및 QueueBrowser 오브젝트에서 사용하는 큐에는 get, inq 및 browse 권한이 필요합니다.
- QueueSession.createTemporaryQueue() 메소드는 QueueConnectionFactory 오브젝트의 TEMPMODEL 특성이 지정한 모델 큐에 대한 액세스 권한이 필요합니다. 기본적으로 이 모델 큐는 SYSTEM.TEMP.MODEL.QUEUE입니다.

이러한 큐가 알리어스 큐이면 대상 큐에 조회 권한이 필요합니다. 대상 큐가 클러스터 큐이면 찾아보기 권한도 필요합니다.

발행/구독 도메인에서 IBM MQ classes for JMS가 IBM MQ 메시징 제공자 마이그레이션 모드로 IBM MQ 큐 관리자에 연결하는 경우 다음 큐가 사용됩니다.

- SYSTEM.JMS.ADMIN.QUEUE
- SYSTEM.JMS.REPORT.QUEUE
- SYSTEM.JMS.MODEL.QUEUE
- SYSTEM.JMS.PS.STATUS.QUEUE
- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.SUBSCRIBER.QUEUE
- SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- SYSTEM.BROKER.CONTROL.QUEUE

IBM MQ 메시징 제공자 마이그레이션 모드에 대한 추가 정보는 [JMS PROVIDERVERSION](#) 특성 구성을 참조하십시오.

또한 IBM MQ classes for JMS가 이 모드의 큐 관리자에 연결하는 경우 메시지를 발행하는 모든 애플리케이션이 TopicConnectionFactory 또는 Topic 오브젝트가 지정한 스트림 큐에 액세스해야 합니다. 기본적으로 이 큐는 SYSTEM.BROKER.DEFAULT.STREAM입니다.

ConnectionConsumer, IBM MQ 자원 어댑터 또는 WebSphere Application Server IBM MQ 메시징 제공자를 사용하는 경우 추가 권한을 부여해야 합니다.

ConnectionConsumer가 읽을 큐에는 get, inq 및 browse 권한이 있어야 합니다. ConnectionConsumer에서 사용하는 시스템 데드-레터 큐 및 백아웃-리큐 큐 또는 보고서 큐에는 put 및 passall 권한이 있어야 합니다.

애플리케이션에서 IBM MQ 메시징 제공자 정상 모드를 사용하여 발행/구독 메시징을 수행할 때 애플리케이션에서 큐 관리자가 제공하는 통합 발행/구독 기능을 사용합니다. 사용된 토픽과 큐의 보안에 관한 정보는 [발행/구독 보안](#)을 참조하십시오.

IBM MQ classes for JMS의 연결 모드

IBM MQ classes for JMS 애플리케이션에서 클라이언트 또는 바인딩 모드로 큐 관리자에 연결할 수 있습니다. 클라이언트 모드에서 IBM MQ classes for JMS가 TCP/IP를 통해 큐 관리자에 연결합니다. 바인딩 모드에서 IBM MQ classes for JMS가 JNI(Java Native Interface)를 사용하여 큐 관리자에 직접 연결합니다.

바인딩 모드는 모든 환경에서 사용할 수 z/OS 있지만, 클라이언트 모드는 다음 환경에서만 사용할 수 있습니다:

- WebSphere Liberty WebSphere Application Server 프로파일은 모든 플랫폼(포함 z/OS)에서 모든 큐 관리자에 연결됩니다.
- 배치 환경에서 어떤 LPAR에서 실행 중인 큐 IBM MQ for z/OS 매니저에 연결할 때.

다른 플랫폼에서 실행 중인 애플리케이션은 바인딩 모드나 클라이언트 모드에서 큐 관리자에 연결할 수 있습니다.

현재 또는 이전에 지원되는 IBM MQ classes for JMS 버전을 현재 큐 관리자와 함께 사용할 수 있으며, 현재 또는 이전에 지원되는 큐 관리자 버전과 현재 IBM MQ classes for JMS 버전을 사용할 수 있습니다. 다른 버전을 혼합하면 이전 버전의 레벨로 기능이 제한됩니다.

다음 섹션에서는 각 연결 모드를 자세히 설명합니다.

클라이언트 모드

클라이언트 모드로 큐 관리자에 연결하기 위해 IBM MQ classes for JMS 애플리케이션이 큐 관리자가 실행 중인 시스템과 동일한 시스템 또는 다른 시스템에서 실행될 수 있습니다. 어느 경우든 IBM MQ classes for JMS가 TCP/IP를 통해 큐 관리자에 연결합니다.

바인딩 모드

바인딩 모드에서 큐 관리자에 연결하려면 큐 관리자가 실행 중인 시스템에서 IBM MQ classes for JMS 애플리케이션이 실행되어야 합니다.

IBM MQ classes for JMS가 JNI(Java Native Interface)를 사용하여 큐 관리자에 직접 연결합니다. 바인딩 전송을 사용하려면 플랫폼 네이티브 인터페이스 IBM MQJava 라이브러리에 IBM MQ classes for JMS 접근 가능한 환경에서 실행해야 합니다. 자세한 내용은 [플랫폼 96 페이지의 『JNI\(Java Native Interface\) 라이브러리 구성』](#) 참조하십시오.

IBM MQ classes for JMS는 *ConnectOption*의 다음 값을 지원합니다.

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR

IBM MQ classes for JMS에서 사용한 연결 옵션을 변경하려면 연결 팩토리 특성 *CONNOPT*를 수정하십시오.

연결 옵션에 대한 자세한 정보는 [759 페이지의 『MQCONN 호출을 사용하여 큐 관리자에 연결』](#)의 내용을 참조하십시오.

바인딩 전송을 사용하려면 사용 중인 Java Runtime Environment에서 IBM MQ classes for JMS가 연결 중인 큐 관리자의 CCSID(Coded Character Set Identifier)를 지원해야 합니다.

런타임 Java 환경에서 지원하는 CCSID를 확인하는 방법에 대한 자세한 내용은 [여기에서 확인할 수 IBM MQ 프로브 ID 21을 사용하여 생성된 FDC는 클래스를 IBM MQV7 또는 IBM MQV7 Java 클래스를 에 사용할 때 생성됩니다. JMS](#) 있습니다.

IBM MQ classes for JMS 애플리케이션이 클라이언트 모드로 연결할 수 있도록 큐 관리자 구성

IBM MQ classes for JMS 애플리케이션이 클라이언트 모드로 연결할 수 있도록 큐 관리자를 구성하려면 서버 연결 채널 정의를 작성하고 리스너를 시작해야 합니다.

서버 연결 채널 정의 작성

모든 플랫폼에서 MQSC 명령 DEFINE CHANNEL을 사용하여 서버 연결 채널 정의를 작성할 수 있습니다. 다음 예를 참조하십시오.

```
DEFINE CHANNEL(JAVA.CHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

IBM i

IBM i에서는 다음 예제와 같이 CL 명령 CRTMQMCHL을 대신 사용할 수 있습니다.

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN)  
TRPTYPE(*TCP)  
MQMNAME(QMGRNAME)
```

이 명령에서 *QMGRNAME*은 큐 관리자의 이름입니다.

Windows

Linux

Linux 및 Windows에서 IBM MQ Explorer를 사용하여 서버 연결 채널 정의도 작성할 수 있습니다.

z/OS

z/OS에서 조작 패널과 제어판을 사용하여 서버 연결 채널 정의를 작성할 수 있습니다.

채널의 이름(이전 예의 *JAVA.CHANNEL*)은 애플리케이션에서 큐 관리자에 연결하는 데 사용하는 연결 팩토리의 *CHANNEL* 특성으로 지정한 채널 이름과 같아야 합니다. *CHANNEL* 특성의 기본값은 *SYSTEM.DEF.SVRCONN*입니다.

리스너 시작

큐 관리자의 리스너가 아직 시작되지 않은 경우 지금 시작해야 합니다.

Multi

다중 플랫폼에서는 다음 예제에 표시된 대로 MQSC 명령인 *DEFINE LISTENER*를 사용하여 먼저 리스너 오브젝트를 작성한 후 MQSC 명령인 *START LISTENER*를 사용하여 리스너를 시작할 수 있습니다.

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)  
START LISTENER(LISTENER.TCP)
```

z/OS

z/OS에서 다음 예에 표시된 대로 *START LISTENER* 명령만 사용합니다. 단, 채널 시작기 주소 공간을 시작해야 리스너를 시작할 수 있다는 점에 유의하십시오.

```
START LISTENER TRPTYPE(TCP) PORT(1414)
```

IBM i

IBM i에서는 다음 예제와 같이 CL 명령 STRMQMLSR도 사용하여 리스너를 시작할 수 있습니다.

```
STRMQMLSR PORT(1414) MQMNAME(QMGRNAME)
```

이 명령에서 *QMGRNAME*은 큐 관리자의 이름입니다.

ALW

AIX, Linux, and Windows에서는 다음 예제와 같이 제어 명령인 *runmqcls*를 사용하여 리스너를 시작할 수도 있습니다.

```
runmqclsr -t tcp -p 1414 -m QMgrName
```

이 명령에서 *QMgrName*은 큐 관리자의 이름입니다.

Windows

Linux

Linux 및 Windows에서는 IBM MQ Explorer를 사용하여 리스너를 시작할 수도 있습니다.

z/OS

z/OS에서는 조작 패널과 제어판을 사용해서도 리스너를 시작할 수 있습니다.

리스너가 대기 중인 포트의 번호는 애플리케이션이 큐 관리자에 연결하는 데 사용하는 연결 팩토리의 *PORT* 특성에 지정된 포트 번호와 같아야 합니다. *PORT* 특성의 기본값은 *1414*입니다.

IBM MQ classes for JMS에 대한 포인트-투-포인트 IVT

포인트 투 포인트 설치 검증 테스트(IVT) 프로그램이 및 IBM MQ classes for JMS 와 함께 IBM MQ classes for Jakarta Messaging제공됩니다. 프로그램에서 바인딩 또는 클라이언트 모드로 큐 관리자에 연결하고 *SYSTEM.DEFAULT.LOCAL.QUEUE*라는 큐에 메시지를 송신한 다음 큐에서 메시지를 수신합니다. 프로그램에서 런타임 시 동적으로 필요한 모든 오브젝트를 작성하여 구성하거나 JNDI를 사용하여 디렉토리 서비스에서 관리 대상 오브젝트를 검색할 수 있습니다.

테스트가 자체 포함되어 있어 디렉토리 서비스를 사용하지 않아도 되므로 먼저 JNDI를 사용하지 않는 설치 확인 테스트를 실행하십시오. 관리 대상 오브젝트 설명은 [관리 도구를 사용하여 JMS 오브젝트 구성](#)을 참조하십시오.

JNDI를 사용하지 않는 포인트-투-포인트 설치 확인 테스트

이 테스트에서 IVT 프로그램이 런타임 시 동적으로 필요한 모든 오브젝트를 작성하고 구성하며 JNDI를 사용하지 않습니다.

Multi 다중 플랫폼에서는 IVT 프로그램을 실행하기 위한 스크립트가 제공됩니다. 스크립트는 시스템에서는 AIX and Linux 로, **IVTRun.bat**에서는 **IVTRun**로 호출됩니다. Windows. 스크립트는 설치 IBM MQ classes for JMS 디렉토리의 하위 bin 디렉토리에 위치합니다. 클래스패스에는 **com.ibm.mqjms.jar**이 포함되어야 합니다.

바인딩 모드로 테스트를 실행하려면 다음 명령을 입력하십시오.

```
IVTRun -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

클라이언트 모드에서 테스트를 실행하려면 먼저 설명된 대로 큐 관리자를 979 페이지의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』 설정하십시오. 사용할 채널은 기본값으로 **SYSTEM.DEF.SVRCONN**이며 **SYSTEM.DEFAULT.LOCAL.QUEUE**, 사용할 큐는 입니다. 그런 다음 다음 명령을 입력하십시오:

```
IVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]  
[-v providerVersion ] [-ccsid csid ] [-t]
```

z/OS 해당 시스템에는 z/OS 동등한 스크립트가 제공되지 않습니다. 대신, Java 클래스를 직접 호출하여 바인딩 모드에서 IVT를 실행합니다. IVT 프로그램의 기능적으로 동일한 두 인스턴스 중에서 선택합니다 z/OS:

- **com.ibm.mq.jms.MQJMSIVT(JMS 2.0)** IBM MQ classes for JMS 로 제공됩니다. 이 프로그램을 사용하려면 클래스패스에 또는 가 **com.ibm.mqjms.jar** 포함되어야 **com.ibm.mq.allclient.jar** 합니다.
- **com.ibm.mq.jakarta.jms.MQJMSIVT(Jakarta Messaging 3.0)** IBM MQ classes for Jakarta Messaging 로 제공됩니다. 이 프로그램을 사용하려면 클래스패스에 포함되어야 합니다 **com.ibm.mq.jakarta.client.jar**.

바인딩 모드에서 테스트를 실행하려면 다음 z/OS 명령을 입력하십시오:

```
java com.ibm.mq.jms.MQJMSIVT -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

명령의 매개변수는 다음을 나타냅니다.

-m qmgr

IVT 프로그램이 연결하는 큐 관리자의 이름입니다. 바인딩 모드에서 테스트를 실행하고 이 매개변수를 생략하면 IVT 프로그램이 기본 큐 관리자에 연결합니다.

-host hostname

큐 관리자가 실행 중인 시스템의 호스트 이름 또는 IP 주소입니다.

-port port

큐 관리자의 리스너가 대기 중인 포트의 번호입니다. 기본값은 1414입니다.

-channel channel

IVT 프로그램이 큐 관리자에 연결하는 데 사용하는 MQI 채널의 이름입니다. 기본값은 **SYSTEM.DEF.SVRCONN**입니다.

-v providerVersion

IVT 프로그램이 연결할 큐 관리자의 릴리스 레벨입니다.

이 매개변수는 MQQueueConnectionFactory 오브젝트의 PROVIDERVERSION 특성을 설정하는 데 사용하며 값은 PROVIDERVERSION 특성의 값과 동일한 올바른 값입니다. 올바른 값을 포함하여 이 매개변수에 대한 자세한 정보는 [JMS: PROVIDERVERSION 특성 변경사항 및 IBM MQ classes for JMS 오브젝트 특성](#)에 있는 PROVIDERVERSION 특성 설명을 참조하십시오.

기본값은 unspecified입니다.

-ccsid ccsid

연결에서 사용할 코드화된 문자 세트 또는 코드 페이지의 ID(CCSID)입니다. 기본값은 819입니다.

-t

추적이 사용 가능합니다. 기본적으로 추적이 사용 불가능합니다.

테스트에 성공하면 다음 샘플 출력과 비슷한 출력이 생성됩니다.

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2026. All
Rights Reserved.
WebSphere MQ classes for Java(tm) Message Service 7.0
Installation Verification Test
```

```
Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message
JMSMessage class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:414d5120514d5f6d6277202020202001edb14620005e03
JMSTimestamp: 1187170264000
JMSCorrelationID: null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 28
JMSXAppID: IBM MQ Client for Java
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_PutTime: 09310400
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSIVT
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

JNDI를 사용하는 포인트-투-포인트 설치 확인 테스트

Multi

이 테스트에서 IVT 프로그램은 JNDI를 사용하여 디렉토리 서비스에서 관리 대상 오브젝트를 검색합니다.

LDAP(Lightweight Directory Access Protocol) 서버 또는 로컬 파일 시스템을 기반으로 디렉토리 서비스를 구성해야 테스트를 실행할 수 있습니다. 디렉토리 서비스를 사용하여 관리 대상 오브젝트를 저장할 수 있도록 IBM MQ JMS 관리 도구도 구성해야 합니다. 이러한 필수조건에 대한 자세한 정보는 89 페이지의 『IBM MQ classes for JMS의 전제조건』의 내용을 참조하십시오. IBM MQ JMS 관리 도구를 구성하는 방법에 대한 정보는 [JMS 관리 도구 구성](#)을 참조하십시오.

IV 프로그램은 JNDI를 사용하여 디렉토리 서비스에서 MQQueueConnectionFactory 오브젝트와 MQQueue 오브젝트를 검색할 수 있어야 합니다. 이러한 관리 대상 오브젝트를 작성하기 위한 스크립트가 제공됩니다. 스크립트는 AIX and Linux 시스템에서 IVTSetup이라고 하고 Windows에서는 IVTSetup.bat라고 하며 IBM MQ

classes for JMS 설치 디렉토리의 bin 서브디렉토리에 있습니다. 스크립트를 실행하려면 다음 명령을 입력하십시오.

IVTSetup

이 스크립트는 관리 대상 오브젝트를 작성하기 위해 IBM MQ JMS 관리 도구를 호출합니다.

MQQueueConnectionFactory 오브젝트는 ivtQCF라는 이름으로 바인딩되고 모든 특성이 기본값으로 작성됩니다. 즉, IVT 프로그램이 바인딩 모드로 실행되며 기본 큐 관리자에 연결됩니다. IVT 프로그램을 클라이언트 모드로 실행하거나 기본 큐 관리자 이외의 큐 관리자에 연결하려면 IBM MQ JMS 관리 도구나 IBM MQ Explorer를 사용하여 MQQueueConnectionFactory 오브젝트의 적절한 특성을 변경해야 합니다. IBM MQ Explorer JMS 관리 도구를 사용하는 방법에 대한 정보는 [관리 도구를 사용하여 JMS 오브젝트 구성을 참조하십시오](#). IBM MQ Explorer를 사용하는 방법에 대한 정보는 [IBM MQ Explorer 소개](#) 또는 IBM MQ Explorer와 함께 제공되는 도움말을 참조하십시오.

MQQueue 오브젝트는 ivtQ라는 이름으로 바인딩되고 모든 특성이 기본값으로 작성됩니다. 단, 값이 SYSTEM.DEFAULT.LOCAL.QUEUE인 QUEUE 특성은 제외됩니다.

관리 대상 오브젝트를 작성한 경우 IVT 프로그램을 실행할 수 있습니다. JNDI를 사용하여 테스트를 실행하려면 다음 명령을 입력하십시오.

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

명령의 매개변수는 다음을 나타냅니다.

-url "providerURL"

디렉토리 서비스의 URL(Uniform Resource Locator)입니다. URL의 형식은 다음 중 하나일 수 있습니다.

- ldap://hostname/contextName - LDAP 서버를 기반으로 하는 디렉토리 서비스의 경우
- file:/directoryPath - 로컬 파일 시스템을 기반으로 하는 디렉토리 서비스의 경우

URL은 따옴표(")로 묶어야 합니다.

-icf initCtxFact

초기 컨텍스트 팩토리의 클래스 이름이며, 값은 다음 중 하나여야 합니다.

- com.sun.jndi.ldap.LdapCtxFactory - LDAP 서버를 기반으로 하는 디렉토리 서비스의 경우. 이는 기본값입니다.
- com.sun.jndi.fscontext.RefFSContextFactory - 로컬 파일 시스템을 기반으로 하는 디렉토리 서비스.

-t

추적이 사용 가능합니다. 기본적으로 추적이 사용 불가능합니다.

테스트에 성공하면 JNDI를 사용하지 않고 성공한 테스트와 비슷한 출력이 생성됩니다. 주된 차이점으로 이 출력에는 테스트에서 JNDI를 사용하여 MQQueueConnectionFactory 오브젝트와 MQQueue 오브젝트를 검색한다고 점이 표시됩니다.

반드시 필요한 것은 아니지만 IVTSetup 스크립트를 통해 작성한 관리 대상 오브젝트를 삭제하여 테스트 후에 정리를 하는 것이 좋습니다. 이 용도의 스크립트가 제공됩니다. 스크립트는 AIX and Linux 시스템에서는 IVTTidy라고 하며 Windows에서는 IVTTidy.bat이라고 하고 IBM MQ classes for JMS 설치 디렉토리의 bin 서브디렉토리에 있습니다.

포인트-투-포인트 설치 확인 테스트의 문제점 판별

Multi

설치 확인 테스트는 다음과 같은 이유로 인해 실패할 수 있습니다.

- IVT 프로그램에서 클래스를 찾을 수 없다는 메시지를 작성하면 94 페이지의 『[환경 변수 IBM MQ classes for JMS 설정Jakarta Messaging](#)』에 설명된 대로 클래스 경로가 올바르게 설정되었는지 확인하십시오.
- 테스트에 실패하고 다음 메시지가 표시될 수 있습니다.

```
Failed to connect to queue manager ' qmgr ' with connection mode ' connMode '
and host name ' hostname '
```

연관된 이유 코드는 2059입니다. 메시지의 변수는 다음을 나타냅니다.

QMGR

IVT 프로그램에서 연결하려는 큐 관리자의 이름입니다. IVT 프로그램에서 바인딩 모드로 기본 큐 관리자에 연결하려고 하면 이 메시지 삽입은 공백입니다.

connMode

연결 모드(Bindings 또는 Client)입니다.

hostname

큐 관리자가 실행 중인 시스템의 호스트 이름 또는 IP 주소입니다.

이 메시지는 IVT 프로그램이 연결하려는 큐 관리자를 사용할 수 없다는 의미입니다. 큐 관리자가 실행 중인지 확인하고 IVT 프로그램에서 기본 큐 관리자에 연결하려고 하면 큐 관리자가 시스템의 기본 큐 관리자로 정의되었는지 확인하십시오.

- 테스트에 실패하고 다음 메시지가 표시될 수 있습니다.

```
Failed to open MQ queue 'SYSTEM.DEFAULT.LOCAL.QUEUE'
```

이 메시지는 IVT 프로그램이 연결하는 큐 관리자에 SYSTEM.DEFAULT.LOCAL.QUEUE 큐가 없다는 의미입니다. 또는 큐가 있는 경우 메시지를 넣거나 가져오기에 큐가 사용되지 않으므로 IVT 프로그램에서 큐를 열 수 없습니다. 큐 관리자가 있으며 메시지 넣기와 가져오기에 사용되는지 확인하십시오.

- 테스트에 실패하고 다음 메시지가 표시될 수 있습니다.

```
Unable to bind to object
```

이 메시지는 LDAP 서버에 대한 연결이 있지만 LDAP 서버가 올바르게 구성되지 않았음을 나타냅니다. LDAP 서버가 Java 오브젝트를 저장하도록 구성되지 않았거나 접미부 또는 오브젝트의 권한이 올바르지 않습니다. 이 상황에 대한 추가 도움말은 LDAP 서버의 문서를 참조하십시오.

- 테스트에 실패하고 다음 메시지가 표시될 수 있습니다.

```
The security authentication was not valid that was supplied for
QueueManager ' qmgr ' with connection mode 'Client' and host name ' hostname '
```

이 메시지는 시스템에서 클라이언트 연결을 승인하도록 큐 관리자가 올바르게 설정되지 않았음을 나타냅니다. 세부사항은 979 페이지의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』을(를) 참조하십시오.

IBM MQ classes for JMS의 발행/구독 IVT

발행/구독 설치 확인 테스트(IVT) 프로그램은 IBM MQ classes for JMS와 함께 제공됩니다. 프로그램이 바인딩 또는 클라이언트 모드에서 큐 관리자에 연결되고 토픽을 구독하며 토픽에서 메시지를 발행한 다음 방금 발행한 메시지를 수신합니다. 프로그램에서 런타임 시 동적으로 필요한 모든 오브젝트를 작성하여 구성하거나 JNDI를 사용하여 디렉토리 서비스에서 관리 대상 오브젝트를 검색할 수 있습니다.

테스트가 자체 포함되어 있어 디렉토리 서비스를 사용하지 않아도 되므로 먼저 JNDI를 사용하지 않는 설치 확인 테스트를 실행하십시오. 관리 대상 오브젝트 설명은 [관리 도구를 사용하여 JMS 오브젝트 구성](#)을 참조하십시오.

JNDI를 사용하지 않는 발행/구독 설치 확인 테스트

이 테스트에서 IVT 프로그램이 런타임 시 동적으로 필요한 모든 오브젝트를 작성하고 구성하며 JNDI를 사용하지 않습니다.

IVT 프로그램을 실행하기 위한 스크립트가 제공됩니다. 스크립트는 AIX and Linux 시스템에서 PSIVTRun이라고 하며 Windows에서는 PSIVTRun.bat이라고 하고, IBM MQ classes for JMS 설치 디렉토리의 bin 서브디렉토리에 있습니다.

바인딩 모드로 테스트를 실행하려면 다음 명령을 입력하십시오.

```
PSIVTRun -nojndi [-m qmgr ] [-bqm brokerQmgr ] [-v providerVersion ] [-t]
```

클라이언트 모드에서 테스트를 실행하려면 먼저 979 페이지의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』에 설명된 대로 큐 관리자를 설정하여 사용할 채널의 기본값은 SYSTEM.DEF.SVRCONN으로 지정한 후 다음 명령을 입력하십시오.

```
PSIVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]  
[-bqm brokerQmgr ] [-v providerVersion ] [-ccsid ccsid ] [-t]
```

명령의 매개변수는 다음을 나타냅니다.

-m qmgr

IVT 프로그램이 연결하는 큐 관리자의 이름입니다. 바인딩 모드에서 테스트를 실행하고 이 매개변수를 생략하면 IVT 프로그램이 기본 큐 관리자에 연결합니다.

-host hostname

큐 관리자가 실행 중인 시스템의 호스트 이름 또는 IP 주소입니다.

-port port

큐 관리자의 리스너가 대기 중인 포트의 번호입니다. 기본값은 1414입니다.

-channel channel

IVT 프로그램이 큐 관리자에 연결하는 데 사용하는 MQI 채널의 이름입니다. 기본값은 SYSTEM.DEF.SVRCONN입니다.

-bqm brokerQmgr

브로커를 실행 중인 큐 관리자의 이름입니다. 기본값은 IVT 프로그램이 연결하는 큐 관리자의 이름입니다.

이 매개변수는 7 이상의 큐 관리자 버전 번호 v과(와) 관련이 없습니다.

-v providerVersion

IVT 프로그램이 연결할 큐 관리자의 릴리스 레벨입니다.

이 매개변수는 MQTopicConnectionFactory 오브젝트의 PROVIDERVERSION 특성을 설정하는 데 사용하며 값은 PROVIDERVERSION 특성의 값과 동일한 올바른 값입니다. 따라서 올바른 값을 포함하여 이 매개변수에 대한 자세한 정보는 [IBM MQ classes for JMS 오브젝트의 특성](#)에 있는 PROVIDERVERSION 특성에 대한 설명을 참조하십시오.

기본값은 unspecified입니다.

-ccsid ccsid

연결에서 사용할 코드화된 문자 세트 또는 코드 페이지의 ID(CCSID)입니다. 기본값은 819입니다.

-t

추적이 사용 가능합니다. 기본적으로 추적이 사용 불가능합니다.

테스트에 성공하면 다음 샘플 출력과 비슷한 출력이 생성됩니다.

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2026. All  
Rights Reserved.  
IBM MQ classes for Java Message Service 7.0  
Publish/Subscribe Installation Verification Test
```

```
Creating a TopicConnectionFactory  
Creating a Connection  
Creating a Session  
Creating a Topic  
Creating a TopicPublisher  
Creating a TopicSubscriber  
Creating a TextMessage  
Adding text  
Publishing the message to topic://MQJMS/PSIVT/Information  
Waiting for a message to arrive [5 secs max]...
```

```
Got message:  
JMSMessage class: jms_text  
JMSType: null  
JMSDeliveryMode: 2  
JMSExpiration: 0  
JMSPriority: 4  
JMSMessageID: ID:414d5120514d5f6d6277202020202001edb14620006706  
JMSTimestamp: 1187182520203
```

```

JMSCorrelationID: ID:414d5120514d5f6d6277202020202001edb14620006704
JMSDestination: topic://MQJMS/PSIVT/Information
JMSReplyTo: null
JMSRedelivered: false
JMSXUserID: mwwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 26
JMSXAppID: QM_mbw
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_ConnectionID: 414D5143514D5F6D6277202020202001EDB14620006601
JMS_IBM_PutTime: 12552020
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSPSIVT program
Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT finished

```

JNDI를 사용하는 발행/구독 설치 확인 테스트

이 테스트에서 IVT 프로그램은 JNDI를 사용하여 디렉토리 서비스에서 관리 대상 오브젝트를 검색합니다.

LDAP(Lightweight Directory Access Protocol) 서버 또는 로컬 파일 시스템을 기반으로 디렉토리 서비스를 구성해야 테스트를 실행할 수 있습니다. 디렉토리 서비스를 사용하여 관리 대상 오브젝트를 저장할 수 있도록 IBM MQ JMS 관리 도구도 구성해야 합니다. 이러한 필수조건에 대한 자세한 정보는 [89 페이지의 『IBM MQ classes for JMS의 전제조건』](#)의 내용을 참조하십시오. IBM MQ JMS 관리 도구를 구성하는 방법에 대한 정보는 [JMS 관리 도구 구성](#)을 참조하십시오.

IV 프로그램은 JNDI를 사용하여 디렉토리 서비스에서 MQTopicConnectionFactory 오브젝트와 MQTopic 오브젝트를 검색할 수 있어야 합니다. 이러한 관리 대상 오브젝트를 작성하기 위한 스크립트가 제공됩니다. 스크립트는 AIX and Linux 시스템에서 IVTSetup이라고 하고 Windows에서는 IVTSetup.bat라고 하며 IBM MQ classes for JMS 설치 디렉토리의 bin 서브디렉토리에 있습니다. 스크립트를 실행하려면 다음 명령을 입력하십시오.

```
IVTSetup
```

이 스크립트는 관리 대상 오브젝트를 작성하기 위해 IBM MQ JMS 관리 도구를 호출합니다.

MQTopicConnectionFactory 오브젝트는 ivtTCF라는 이름으로 바인딩되고 모든 특성이 기본값으로 작성됩니다. 즉, IVT 프로그램이 바인딩 모드로 실행되며 기본 큐 관리자에 연결되고 임베드된 발행/구독 함수를 사용합니다. IVT 프로그램을 클라이언트 모드에서 실행하거나, 기본 큐 관리자가 아닌 큐 관리자에 연결하거나, 임베드된 발행/구독 함수 대신 IBM Integration Bus를 사용하는 경우 IBM MQ JMS 관리 도구 또는 IBM MQ 탐색기를 사용하여 MQTopicConnectionFactory 오브젝트의 적절한 특성을 변경해야 합니다. IBM MQ JMS 관리 도구를 사용하는 방법에 대한 정보는 [관리 도구를 사용하여 JMS 오브젝트 구성](#)을 참조하십시오. IBM MQ 탐색기를 사용하는 방법에 대한 정보는 IBM MQ 탐색기와 함께 제공된 도움말을 참조하십시오.

MQTopic 오브젝트는 ivt라는 이름으로 바인딩되고 모든 특성이 기본값으로 작성됩니다. 단, 값이 MQJMS/PSIVT/Information인 TOPIC 특성은 제외입니다.

관리 대상 오브젝트를 작성한 경우 IVT 프로그램을 실행할 수 있습니다. JNDI를 사용하여 테스트를 실행하려면 다음 명령을 입력하십시오.

```
PSIVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

명령의 매개변수는 다음을 나타냅니다.

-url "providerURL"

디렉토리 서비스의 URL(Uniform Resource Locator)입니다. URL의 형식은 다음 중 하나일 수 있습니다.

- `ldap://hostname/contextName` - LDAP 서버를 기반으로 하는 디렉토리 서비스의 경우
- `file:/directoryPath` - 로컬 파일 시스템을 기반으로 하는 디렉토리 서비스의 경우

URL은 따옴표(")로 묶어야 합니다.

-icf *initCtxFact*

초기 컨텍스트 팩토리의 클래스 이름이며, 값은 다음 중 하나여야 합니다.

- `com.sun.jndi.ldap.LdapCtxFactory` - LDAP 서버를 기반으로 하는 디렉토리 서비스의 경우. 이는 기본값입니다.
- `com.sun.jndi.fscontext.RefFSContextFactory` - 로컬 파일 시스템을 기반으로 하는 디렉토리 서비스.

-t

추적이 사용 가능합니다. 기본적으로 추적이 사용 불가능합니다.

테스트에 성공하면 JNDI를 사용하지 않고 성공한 테스트와 비슷한 출력이 생성됩니다. 주된 차이점으로 이 출력에는 테스트에서 JNDI를 사용하여 `MQTopicConnectionFactory` 오브젝트와 `MQTopic` 오브젝트를 검색한다고 점이 표시됩니다.

반드시 필요한 것은 아니지만 `IVTSetup` 스크립트를 통해 작성한 관리 대상 오브젝트를 삭제하여 테스트 후에 정리를 하는 것이 좋습니다. 이 용도의 스크립트가 제공됩니다. 스크립트는 AIX and Linux 시스템에서는 `IVTTidy` 라고 하며 Windows에서는 `IVTTidy.bat`이라고 하고 IBM MQ classes for JMS 설치 디렉토리의 `bin` 서브디렉토리에 있습니다.

발행/구독 설치 확인 테스트의 문제점 판별

설치 확인 테스트는 다음과 같은 이유로 인해 실패할 수 있습니다.

- IVT 프로그램에서 클래스를 찾을 수 없다는 메시지를 작성하면 94 페이지의 『환경 변수 IBM MQ classes for JMS 설정Jakarta Messaging』에 설명된 대로 클래스 경로가 올바르게 설정되었는지 확인하십시오.
- 테스트에 실패하고 다음 메시지가 표시될 수 있습니다.

```
Failed to connect to queue manager 'qmgr' with
connection mode 'connMode' and host name 'hostname'
```

연관된 이유 코드는 2059입니다. 메시지의 변수는 다음을 나타냅니다.

QMGR

IVT 프로그램에서 연결하려는 큐 관리자의 이름입니다. IVT 프로그램에서 바인딩 모드로 기본 큐 관리자에 연결하려고 하면 이 메시지 삽입은 공백입니다.

connMode

연결 모드(Bindings 또는 Client)입니다.

hostname

큐 관리자가 실행 중인 시스템의 호스트 이름 또는 IP 주소입니다.

이 메시지는 IVT 프로그램이 연결하려는 큐 관리자를 사용할 수 없다는 의미입니다. 큐 관리자가 실행 중인지 확인하고 IVT 프로그램에서 기본 큐 관리자에 연결하려고 하면 큐 관리자가 시스템의 기본 큐 관리자로 정의되었는지 확인하십시오.

- 테스트에 실패하고 다음 메시지가 표시될 수 있습니다.

```
Unable to bind to object
```

이 메시지는 LDAP 서버에 대한 연결이 있지만 LDAP 서버가 올바르게 구성되지 않았음을 나타냅니다. LDAP 서버가 Java 오브젝트를 저장하도록 구성되지 않았거나 접미부 또는 오브젝트의 권한이 올바르지 않습니다. 이 상황에 대한 추가 도움말은 LDAP 서버의 문서를 참조하십시오.

- 테스트에 실패하고 다음 메시지가 표시될 수 있습니다.

```
The security authentication was not valid that was supplied for
QueueManager 'qmgr' with connection mode 'Client' and host name 'hostname'
```

이 메시지는 큐 관리자가 귀하의 시스템에서 클라이언트 연결을 수락하도록 올바르게 설정되지 않았음을 의미합니다. 자세한 정보는 979 페이지의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』의 내용을 참조하십시오.

▶ JMS 2.0 IBM MQ classes for JMS 샘플 애플리케이션 사용

IBM MQ classes for JMS 샘플 애플리케이션은 JMS API의 공통 기능 개요를 제공합니다. 이를 사용하여 설치 및 메시징 서버 설정을 확인하고 자신만의 애플리케이션을 빌드하는 데 도움을 받을 수 있습니다.

이 태스크 정보

자체 애플리케이션을 작성하는 데 도움이 필요하면 시작점으로서 샘플 애플리케이션을 사용할 수 있습니다. 각 애플리케이션에 대해 소스 버전과 컴파일 버전 모두가 제공됩니다. 샘플 소스 코드를 검토하고 애플리케이션의 필수 오브젝트(ConnectionFactory, Connection, Session, Destination 및 Producer 또는 Consumer 또는 모두)를 작성하고 애플리케이션의 작업 방법을 지정하는 데 필요한 특정 특성을 설정하는 핵심 단계를 식별하십시오. 자세한 정보는 136 페이지의 『작성 IBM MQ classes for JMS/Jakarta Messaging 신청서』의 내용을 참조하십시오. 샘플은 IBM MQ의 향후 릴리스에서 변경될 수 있습니다.

에 대해 JMS 2.0, 118 페이지의 표 11 표시된 위치는 샘플 IBM MQ classes for JMS 애플리케이션이 각 플랫폼에 설치되는 위치를 나타냅니다.

참고:

▶ JM 3.0 현재 IBM MQ classes for Jakarta Messaging 새로운 샘플을 준비 중입니다.

표 11. IBM MQ classes for JMS 샘플 애플리케이션에 대한 설치 디렉토리	
플랫폼	디렉토리
▶ AIX AIX ▶ Linux Linux	MQ_INSTALLATION_PATH/samp/jms/samples
▶ Windows Windows	MQ_INSTALLATION_PATH\tools\jms\samples
▶ IBM i IBM i	/qibm/proddata/mqm/java/samples/jms/samples
▶ z/OS z/OS	MQ_INSTALLATION_PATH/java/samples/jms

이 디렉토리에는 118 페이지의 표 12에서와 같이, 하나 이상의 샘플 애플리케이션을 포함하는 서브디렉토리가 있습니다.

표 12. IBM MQ classes for JMS 샘플 애플리케이션	
샘플 이름	설명
JmsBrowser.java	소비자 애플리케이션에서 수신하는 순서대로, 제거하지 않고 이름 지정된 큐에서 사용 가능한 모든 메시지를 찾는 JMS 큐 브라우저 애플리케이션입니다.
JmsConsumer.java	초기 컨텍스트에서 연결 팩토리 인스턴스 및 목적지 인스턴스를 검색하여 소비자 애플리케이션에서 수신하는 순서대로, 제거하지 않고 이름 지정된 큐에서 사용 가능한 모든 메시지를 찾는 JMS 큐 브라우저 애플리케이션입니다(샘플에서는 파일 시스템 컨텍스트만 지원).
JmsJndiConsumer.java	초기 컨텍스트에서 연결 팩토리 인스턴스 및 목적지 인스턴스를 검색하여 이름 지정된 목적지(큐 또는 토픽)큐에서 메시지를 수신하는 JMS 사용자(수신자 또는 구독자) 애플리케이션입니다(샘플에서는 파일 시스템 컨텍스트만 지원).
JmsJndiProducer.java	초기 컨텍스트에서 연결 팩토리 인스턴스 및 목적지 인스턴스를 검색하여 이름 지정된 목적지(큐 또는 토픽)에 단순 메시지를 전송하는 JMS 생성자(전송자 또는 발행자) 애플리케이션입니다(샘플에서는 파일 시스템 컨텍스트만 지원).
JmsProducer.java	이름 지정된 목적지(큐 또는 토픽)에 단순 메시지를 전송하는 JMS 생성자(전송자 또는 발행자) 애플리케이션입니다.
/interactive/	

표 12. IBM MQ classes for JMS 샘플 애플리케이션 (계속)	
샘플 이름	설명
SampleConsumerJava.java	토픽/큐에서 메시지를 수신합니다.
SampleProducerJava.java	토픽/큐로 메시지를 전송합니다.
/interactive/helper/	
BaseOptions.java	사용자 옵션 기능을 제공하기 위해 확장 가능한 추상 클래스입니다.
IsValidType.java	검증 검사 클래스에 대한 추상 클래스입니다.
JmsApp.java	소비자/생성자 옵션 기능을 제공하기 위해 확장 가능한 추상 클래스입니다.
Keys.java	샘플 애플리케이션의 옵션을 정의하는 키 세트입니다.
Literals.java	상수 리터럴 세트입니다.
MyContext.java	옵션이 제공되는 컨텍스트입니다.
Options.java	사용자 옵션의 기능을 제공합니다.
OptionsPresenter.java	현재 옵션이 제공되는 컨텍스트입니다.
/simple/	
SimpleAsyncPutPTP.java	포인트-투-포인트 메시징에 대한 단순애플리케이션입니다. 메시지는 비동기식으로 전송됩니다 (실행 후 삭제 메시징이라고도 함). 메시지는 수신되지 않습니다.
SimpleDurableSub.java	지속 가능한 구독 기능을 설명하는 단순 애플리케이션입니다.
SimpleJNDILookup.java	초기 컨텍스트를 사용하는 JMS 오브젝트의 검색을 설명하는 최소의 단순 애플리케이션입니다. 큐 관리자에 연결되지 않으며, 메시지는 전송 또는 수신되지 않습니다.
SimpleMQMDRead.java	JMS 애플리케이션에서 MQ 메시지 디스크립터(MQMD) 필드를 JMS 메시지 특성으로 사용하는 방법을 설명하는 단순 애플리케이션입니다. 메시지는 전송되지 않습니다. 사용 중인 큐는 일부 메시지로 채워진다고 가정합니다.
SimpleMQMDWrite.java	JMS 애플리케이션에서 MQ 메시지 디스크립터(MQMD) 필드를 작성하는 방법을 설명하는 단순 애플리케이션입니다. 메시지는 수신되지 않습니다.
SimplePTP.java	포인트-투-포인트 메시징을 위한 최소의 단순 애플리케이션입니다.
SimplePubSub.java	메시징 발행/구독을 위한 최소의 단순 애플리케이션입니다.
SimpleReadAheadPTP.java	포인트-투-포인트 메시징에 대한 단순애플리케이션입니다. 메시지는 큐 관리자에서 스트리밍됩니다(미리 읽기 기능이라고도 함). 메시지는 전송되지 않습니다. 사용 중인 큐는 일부 메시지로 채워진다고 가정합니다.

표 12. IBM MQ classes for JMS 샘플 애플리케이션 (계속)	
샘플 이름	설명
SimpleRequestor.java	요청자를 사용하여 요청 메시지를 전송하고 응답을 기다린 후 수신하는 단순 애플리케이션입니다. 참고: 일부 다른 애플리케이션은 요청 메시지를 처리하고 응답 메시지를 전송한다고 가정합니다.
SimpleResponder.java	메시지의 목적지에서 대기하고 메시지의 replyTo 목적지를 전송하는 단순 애플리케이션입니다. 애플리케이션은 SimpleRequestor 샘플과 함께 작동하도록 작성됩니다.
SimpleRetainedPub.java	보유된 발행물을 설명하는 단순 애플리케이션입니다. 메시지는 수신되지 않습니다.
SimpleWMQJMSPTP.java	포인트-투-포인트 메시지를 위한 최소의 단순 애플리케이션입니다.
SimpleWMQJMSPubSub.java	메시징 발행/구독을 위한 최소의 단순 애플리케이션입니다.

IBM MQ classes for JMS은(는) 샘플 애플리케이션을 실행하는 데 사용할 수 있는 `runjms` 스크립트를 제공합니다. 이 스크립트는 IBM MQ 환경을 설정하여 IBM MQ classes for JMS 샘플 애플리케이션을 실행할 수 있습니다.

120 페이지의 표 13에서는 각 플랫폼에서 스크립트 위치를 표시합니다.

표 13. <code>runjms</code> 스크립트의 위치	
플랫폼	디렉토리
 AIX  Linux	<code>MQ_INSTALLATION_PATH/java/bin/runjms</code>
 Windows	<code>MQ_INSTALLATION_PATH\java\bin\runjms.bat</code>
 IBM i	<code>/qibm/proddata/mqm/java/bin/runjms</code> 또는 <code>/qibm/proddata/mqm/java/bin/runjms64</code>
 z/OS	<code>MQ_INSTALLATION_PATH/java/bin/runjms</code>

`runjms` 스크립트를 사용하여 샘플 애플리케이션을 호출하려면 다음 단계를 완료하십시오.

프로시저

- 명령 프롬프트를 실행하고 실행하려는 샘플 애플리케이션을 포함하는 디렉토리로 이동하십시오.
- 다음 명령을 입력하십시오.

```
Path to the runjms script/runjms sample_application_name
```

샘플 애플리케이션은 필요한 매개변수 목록을 표시합니다.

- 다음 명령을 입력하여 다음 매개변수로 샘플을 실행하십시오.

```
Path to the runjms script/runjms sample_application_name parameters
```

예

 예를 들어, JmsBrowser 샘플을 Linux 실행하려면 다음 명령어를 입력하십시오:

```
cd /opt/mqm/samp/jms/samples  
/opt/mqm/java/bin/runjms JmsBrowser -m QM1 -d LQ1
```

관련 개념

90 페이지의 『IBM MQ classes for JMS에 설치된 항목』

IBM MQ classes for JMS를 설치할 때 여러 파일과 디렉토리가 작성됩니다. Windows에서 환경 변수를 자동으로 설정하여 설치 중에 일부 구성을 수행합니다. 다른 플랫폼과 특정 Windows 환경에서 환경 변수를 설정해야 IBM MQ classes for JMS 애플리케이션을 실행할 수 있습니다.

/와 IBM MQ classes for JMS 함께 제공되는 스크립트 Jakarta Messaging

사용 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 시 수행해야 하는 일반적인 작업을 지원하기 위해 여러 스크립트가 제공됩니다.

121 페이지의 표 14에서는 모든 스크립트와 해당 용도를 나열합니다. 스크립트는 설치 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 디렉토리의 하위 bin 디렉토리에 있습니다.

표 14. 제공된 IBM MQ classes for JMS 스크립트 그리고 IBM MQ classes for Jakarta Messaging	
유틸리티	다음을 사용하십시오.
Cleanup ¹	이 스크립트는 이전 릴리스와의 호환성을 위해 유지보수되지만 기능은 수행하지 않습니다. 구독 정보의 수동 정리는 더 이상 필요하지 않습니다.
DefaultConfiguration	Windows 이외의 플랫폼에서 기본 구성 애플리케이션을 실행합니다.
formatLog ¹	이 스크립트는 이전 릴리스와의 호환성을 위해 유지보수되지만 기능은 수행하지 않습니다. 이제 로그 출력이 읽기 가능한 텍스트로 생성됩니다.
IVTRun ¹ IVTSetup ¹ IVTTidy ¹	110 페이지의 『IBM MQ classes for JMS에 대한 포인트-투-포인트 IVT』에 설명된 대로 포인트-투-포인트 설치 확인 테스트에 사용합니다.
 JMS30Admin ¹	관리 IBM MQ Jakarta Messaging 도구를 실행합니다. 자세한 내용은 관리 도구 시작하기 를 참조하십시오.
 JMS30Admin.config	관리 도구의 구성 IBM MQ Jakarta Messaging 파일은 관리 JMS 도구 구성에 설명된 대로입니다.
 JMSAdmin ¹	관리 IBM MQ JMS 도구를 실행합니다. 자세한 내용은 관리 도구 시작하기 를 참조하십시오.
 JMSAdmin.config	관리 도구의 구성 IBM MQ JMS 파일은 관리 JMS 도구 구성에 설명된 대로입니다.
PSIVTRun ¹	114 페이지의 『IBM MQ classes for JMS의 발행/구독 IVT』에 설명된 대로 발행/구독 설치 확인 테스트 프로그램을 실행합니다.
PSReportDump.class	이 클래스는 이전 릴리스와의 호환성을 위해 유지보수되지만 아무 기능도 수행하지 않습니다.
 setjms30env ^{122 페이지의 『2』}	  에 대해 Jakarta Messaging 3.0, 는 에서 AIX and Linux 설명된 바와 같이 시스템에서 32비트 Java 가상 머신(94 페이지의 『환경 변수 IBM MQ classes for JMS 설정 Jakarta Messaging』 JVM)에서 IBM MQ classes for JMS 애플리케이션을 실행하기 위한 환경 변수를 설정합니다.

표 14. 제공된 IBM MQ classes for JMS 스크립트 그리고 IBM MQ classes for Jakarta Messaging (계속)

유틸리티	다음을 사용하십시오.
JMS 2.0 setjmsenv 122 페이지의 『2』	Linux AIX 에 대해 JMS 2.0, 는 에서 AIX and Linux 설명된 바와 같이 시스템에서 32비트 Java 가상 머신(94 페이지의 『환경 변수 IBM MQ classes for JMS 설정Jakarta Messaging』 JVM)에서 IBM MQ classes for JMS 애플리케이션을 실행하기 위한 환경 변수를 설정합니다.
JM 3.0 setjms30env64 122 페이지의 『2』	Linux AIX 에 대해 Jakarta Messaging 3.0, 는 에서 설명된 바와 같이 64비트 94 페이지의 『환경 변수 IBM MQ classes for JMS 설정Jakarta Messaging』 JVM 에서 AIX and Linux IBM MQ classes for JMS 애플리케이션을 실행하기 위한 환경 변수를 설정합니다.
JMS 2.0 setjmsenv64 122 페이지의 『2』	Linux AIX 에 대해 JMS 2.0, 는 에서 설명된 바와 같이 64비트 94 페이지의 『환경 변수 IBM MQ classes for JMS 설정Jakarta Messaging』 JVM 에서 AIX and Linux IBM MQ classes for JMS 애플리케이션을 실행하기 위한 환경 변수를 설정합니다.

참고:

1. 날짜에 Windows, 파일 이름은 확장자를 가지고 .bat있습니다.
2. 이 스크립트는 에서만 AIX and Linux 이용 가능합니다. 설치 후 Windows, 명령어를 IBM MQ 실행하십시오 **setmqenv**. 자세한 정보는 94 페이지의 『환경 변수 IBM MQ classes for JMS 설정Jakarta Messaging』의 내용을 참조하십시오.

IBM MQ classes for JMS를 사용하여 OSGi 지원

OSGi에서는 애플리케이션 배치를 번들로 지원하는 프레임워크를 제공합니다. OSGi 번들은 IBM MQ classes for JMS의 일부로 제공됩니다.

IBM MQ classes for JMS에는 다음 OSGi 번들이 포함됩니다.

com.ibm.msg.client.osgi.jmsversion_number.jar

IBM MQ classes for JMS의 공용 코드 계층입니다. JMS용 IBM MQ 클래스의 계층화된 아키텍처에 대한 정보는 JMS용 IBM MQ 클래스 아키텍처를 참조하십시오.

com.ibm.msg.client.osgi.jms.prereq_version_number.jar

공용 계층의 필수 Java 아카이브(JAR) 파일입니다.

com.ibm.msg.client.osgi.commonservices.j2se_version_number.jar

Java Platform, Standard Edition(Java SE) 애플리케이션의 공용 서비스입니다.

com.ibm.msg.client.osgi.nls_version_number.jar

공용 계층용 메시지입니다.

com.ibm.msg.client.osgi.wmq_version_number.jar

메시징 IBM MQ 제공업체는 IBM MQ classes for JMS. IBM MQ classes for JMS의 계층화된 아키텍처에 대한 정보는 JMS용 IBM MQ 클래스 아키텍처를 참조하십시오.

com.ibm.msg.client.osgi.wmq.prereq_version_number.jar

IBM MQ 메시징 제공자의 필수 JAR 파일입니다.

com.ibm.msg.client.osgi.wmq.nls_version_number.jar

IBM MQ 메시징 제공자의 메시지입니다.

com.ibm.mq.jakarta.osgi.allclient_version_number.jar

JM 3.0 이 JAR Jakarta Messaging 3.0 파일은 애플리케이션이 와 IBM MQ classes for JMS 를 모두 사용할 IBM MQ classes for Java수 있도록 하며, PCF 메시지를 처리하는 코드도 포함합니다.

com.ibm.mq.jakarta.osgi.allclientprereqs_version_number.jar

▶ **JM 3.0** 이 JAR 파일은 에 Jakarta Messaging 3.0 필요한 전제
com.ibm.mq.jakarta.osgi.allclient_version_number.jar조건을 제공합니다.

com.ibm.mq.osgi.allclient_version_number.jar

▶ **JMS 2.0** 이 JAR JMS 2.0 파일은 애플리케이션이 와 IBM MQ classes for JMS 를 모두 사용할 IBM MQ
classes for Java수 있도록 하며, PCF 메시지를 처리하는 코드도 포함합니다.

com.ibm.mq.osgi.allclientprereqs_version_number.jar

▶ **JMS 2.0** 이 JAR 파일은 에 JMS 2.0 필요한 전제
com.ibm.mq.osgi.allclient_version_number.jar조건을 제공합니다.

설치된 의 IBM MQ 버전 *version_number* 번호입니다.

번들은 IBM MQ 설치의 java/lib/OSGi 서브디렉토리 또는 Windows의 java\lib\OSGi 폴더에 설치됩니
다.

IBM MQ 8.0부터는 모든 새 애플리케이션에 com.ibm.mq.osgi.allclient_8.0.0.0.jar 및
com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar 번들을 사용하십시오. 이러한 번들을 사용하면
동일한 OSGi 프레임워크에서 IBM MQ classes for JMS와 IBM MQ classes for Java를 둘 다 실행할 수 없는 제
한사항이 제거됩니다. 그러나 다른 제한사항은 여전히 모두 적용됩니다.

IBM MQ 설치의 java/lib/OSGi 서브디렉토리 또는 Windows의 java\lib\OSGi 폴더에도 설치되는
com.ibm.mq.osgi.javaversion_number.jar 번들은 IBM MQ classes for Java의 일부입니다. 이 번들
은 IBM MQ classes for JMS가 로드된 OSGi 런타임 환경에 로드하지 않아야 합니다.

IBM MQ classes for JMS의 OSGi 번들은 OSGi 릴리스 4 스펙에 작성되었습니다. OSGi 릴리스 3 환경에서는 작
동하지 않습니다.

OSGi 런타임 환경이 필수 DLL 파일 또는 공유 라이브러리를 찾을 수 있도록 시스템 경로 또는 라이브러리 경로를
올바르게 설정해야 합니다.

IBM MQ classes for JMS의 OSGi 번들을 사용하는 경우 임시 토픽은 작동하지 않습니다. OSGi와 같은 다중 클래
스 로더 환경에서 클래스를 로드하는 데 관한 내재적인 문제점으로 인해 Java에 작성된 채널 엑시트 클래스도 지
원되지 않습니다. 사용자 번들에서는 IBM MQ classes for JMS 번들을 인지할 수 있지만 IBM MQ classes for
JMS 번들은 사용자 번들을 인지하지 못합니다. 결과적으로 IBM MQ classes for JMS 번들에서 사용된 클래스
로더는 사용자 번들에 있는 채널 엑시트 클래스를 로드할 수 없습니다.

OSGi에 대한 자세한 정보는 [OSGi Alliance](#) 웹 사이트를 참조하십시오.

▶ **z/OS** ▶ **MQ Adv. VUE** **JMS/Jakarta Messaging 클라이언트 연결성: 실행 중인 배치 애플리케이션에 대한 z/OS**

특정 조건 하에서, /Jakarta Messaging IBM MQ classes for JMS애플리케이션은 클라이언트 연결을 사용하여
z/OSz/OS 의 큐 관리자에 연결할 수 있습니다. 클라이언트 연결을 사용하면 토폴로지를 IBM MQ 단순화할 수 있
습니다.

다음 조건 중 하나에 해당하는 경우 지원됩니다::

- IBM MQ classes for JMS/가Jakarta Messaging 위치해 있을 IBM MQ 9.4.0때, 큐 관리자는 어떤 버전 아래에
서도, 어떤 PID 아래에서도 실행될 수 있습니다.
- IBM MQ classes for JMS/가 또는 Jakarta MessagingIBM MQ 9.3.0 에 위치하고 APAR PH56722IBM MQ
9.2.0 이 적용된 경우, 큐 관리자는 어떤 버전에서든, 어떤 PID 아래에서든 실행될 수 있습니다.
- IBM MQ classes for JMS/는Jakarta Messaging APAR PH56722 이 적용된 상태이거나 IBM MQ 9.2.0 적용
되지 않은 상태일 수 IBM MQ 9.3.0 있으며, 큐 관리자는 모든 버전에서 실행될 수 있지만 권한이 부여된 IBM
MQ Advanced for z/OS Value Unit Edition 상태로 실행되어야 하므로 해당 **ADVCAP** 매개변수가 **ENABLED**로
설정되어 있어야 합니다.

자세한 내용은 을 IBM MQ Advanced for z/OS Value Unit Edition 참조하십시오 [IBM MQ 제품 식별자 및 수출
정보](#).

자세한 내용은 [DISPLAY QMGR](#)을 참조하십시오. **ADVCAP** 또한 [QMGRPRODSTART QMGR](#)에 대한 자세한 내용은 해당 항목을 참조하십시오.

배치 환경만 지원되며, /Jakarta Messaging for CICS 또는 JMS/ Jakarta MessagingJMS for 환경은 지원되지 IMS 않습니다.

/Jakarta Messaging IBM MQ classes for JMS애플리케이션은 클라이언트 모드 연결을 사용하여 에서 z/OS 실행되지 않는 큐 관리자에 z/OS 연결할 수 없습니다.

클라이언트 모드로 z/OS 연결을 시도하는 애플리케이션이 IBM MQ classes for JMSJakarta Messaging 허용되지 않을 경우 예외 메시지가 JMSFMQ0005 발생합니다.

Advanced Message Security(AMS) 지원

IBM MQ classes for JMSJakarta Messaging 클라이언트 애플리케이션은 이 항목에서 앞서 설명한 조건에 따라 원격 z/OS 큐 관리자에 연결할 때 AMS 사용할 수 있습니다.

이러한 방식으로 AMS 사용하려면 클라이언트 애플리케이션은 에서 키 `jceracfks` 저장소 유형으로 `keystore.conf`를 사용해야 하며, 여기서:

- 특성 이름 접두부는 `jceracfks`이고 이 이름 접두부에서는 대소문자를 구분하지 않습니다.
- 키 저장소는 RACF 키링입니다.
- 비밀번호는 필요하지 않고 무시됩니다. 이는 RACF 키링에서 비밀번호를 사용하지 않기 때문입니다.
- 제공자를 지정하는 경우 제공자는 `IBMJCE`여야 합니다.

`jceracfks`를 AMS와 함께 사용하는 경우 키 저장소는 `safkeyring://user/keyring` 양식이어야 합니다. 여기서

- `safkeyring`은 리터럴이고 이 이름에서는 대소문자가 구분되지 않습니다.
- `user`는 키링을 소유하는 RACF 사용자 ID입니다.
- `keyring`은 RACF 키링의 이름이고, 키링의 이름에서는 대소문자가 구분됩니다.

다음 예제는 사용자에 대한 표준 `AMSJOHNDOE` 키링을 사용합니다:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

관련 개념

341 페이지의 [『Java 클라이언트 연결성: 배치 애플리케이션 실행 시 z/OS』](#)

특정 조건 하에서, 의 애플리케이션은 IBM MQ classes for Java 클라이언트 연결을 사용하여 z/OS z/OS 의 큐 관리자에 연결할 수 있습니다. 클라이언트 연결을 사용하면 토폴로지를 IBM MQ 단순화할 수 있습니다.

와 IBM MQ classes for JMS 를 IBM MQ classes for Jakarta Messaging 각각 획득하다

또는 을 IBM MQ classes for JMS 사용하여 애플리케이션을 실행하는 데 필요한 라이브러리 및 IBM MQ classes for Jakarta Messaging 유틸리티는 에서 다운로드할 수 있는 자동 압축 해제 Fix Central JAR 파일에 포함되어 있습니다. 이 작업을 수행하면 해당 파일들만 추출할 수 있습니다. 예를 들어 소프트웨어 관리 도구로의 배포나 독립형 클라이언트 애플리케이션 사용을 위한 경우에 해당합니다.

시작하기 전에

이 작업을 시작하기 전에, 컴퓨터에 (JRE) Java runtime environment 가 설치되어 있는지, 그리고 시스템 경로에 ()가 JRE 추가되었는지 확인하십시오.

이 설치 프로세스에서 사용된 Java 설치 프로그램은 루트 또는 특성 사용자로 실행 중일 필요가 없습니다. 유일한 요구 사항은 파일을 저장할 디렉터리에 대해 실행 중인 사용자가 쓰기 권한을 가져야 한다는 것입니다.

JM 3.0 IBM MQ 9.3.0부터, Jakarta Messaging 3.0 는 새로운 애플리케이션 개발에 지원됩니다. IBM MQ 9.3.0 그리고 향후에도 기존 애플리케이션에 대한 지원을 JMS 2.0 계속할 예정입니다. 동일한 애플리케이션

에서 두 JMS 2.0 Jakarta Messaging 3.0 API를 모두 사용하는 것은 지원되지 않습니다. 자세한 내용은 ‘JMS/Jakarta Messaging용 IBM MQ 클래스 사용’을 참조하십시오.

이 태스크 정보

다운로드 크기와 압축 해제 시간을 최소화하기 위해 자동 압축 해제 JAR 파일이 사용됩니다. 이 JAR 파일의 정확한 내용과 파일이 추출되는 하위 디렉터리는 IBM MQNET Framework 버전에 따라 다릅니다.

자체 압축 해제 JAR 파일을 실행하면 라이선스 IBM MQ 계약서가 표시되며, 이를 수락해야 합니다. 또한 추출 작업의 상위 디렉터리를 변경할 수 있습니다.

Java IBM MQ 9.3 1.3.0 이상 버전에서는 자동 압축 해제 JAR 파일이 파일을 다음 디렉터리 구조로 추출합니다:

wmq/JavaEE

IBM MQ 리소스 어댑터 EAR 및 RAR 파일.

JMS 2.0 다음 파일들은 및 JMS 1.1 JMS 2.0 객체와 함께 사용하기 위한 것입니다:

- wmq.jmsra.ivt.ear
- wmq.jmsra.rar

JM 3.0 객체와 Jakarta Messaging 3.0 함께 사용할 수 있는 동등한 집합도 있습니다:

- wmq.jakarta.jmsra.ivt.ear 설치 검증 테스트 파일을 포함합니다.
- wmq.jakarta.jmsra.rar 리소스 어댑터 파일을 포함합니다.

wmq/JavaSE

wmq/JavaSE/bin

와 **JMS30Admin** **JMSAdmin** 도구들. 객체를 나타내거나 Jakarta Messaging JMS 객체를 나타내는 엔티티를 JNDI 정의하는 데 사용됩니다.

JMS 2.0 다음 파일들은 및 JMS 1.1 JMS 2.0 객체와 함께 사용하기 위한 것입니다:

- JMSAdmin.bat
- JMSAdmin
- JMSAdmin.config

JM 3.0 객체와 Jakarta Messaging 3.0 함께 사용할 수 있는 동등한 집합도 있습니다:

- JMS30Admin.bat 해당 도구를 시작하는 데 Windows 사용되는 파일.
- JMS30Admin. 해당 도구를 및 UNIX Linux 플랫폼에서 시작하는 데 사용되는 스크립트.
- JMS30Admin.config 도구를 위한 샘플 구성 파일.

참고:

- 해당 **JMSAdmin** 도구가 자동 압축 해제 JAR 파일에 추가되기 전에는, 이 디렉터리의 파일들은 상위 디렉터리에 위치해 있었습니다 wmq/JavaSE.
- 자체 압축 해제 JAR 파일을 사용하여 설치된 클라이언트는 또는 **JMS30Admin** **JMSAdmin** 도구를 사용하여 파일 시스템 컨텍스트(.bindings 파일) 내에서 메시징 관리 Java 객체를 생성할 수 있습니다. 클라이언트는 이러한 관리 대상 오브젝트를 검색하고 사용할 수도 있습니다.
- **JMS 2.0** 사용을 위한 JMS 2.0 도구와 JMS 1.1 **JMSAdmin** 객체가 자체 실행 JAR 파일에 추가되었습니다 IBM MQ 9.2.0 Fix Pack 2 IBM MQ 9.2.2.
- **JM 3.0** 객체와 Jakarta Messaging 3.0 함께 사용하기 위한 도구가 **JMS30Admin** 자체 실행 JAR 파일에 추가되었습니다 IBM MQ 9.3.0.

wmq/JavaSE/lib

고급 메시지 보안은 암호화 메시지 구문(Cryptographic Message Syntax, CMS)을 지원하기 위해 다음과 같은 오픈소스 **Bouncy Castle** 패키지를 사용합니다. AMS를 통한 비 IBM JRE 지원 참조.

IBM MQ 9.4.0부터:

- bcpkix-jdk18on.jar
- bcprov-jdk18on.jar
- bcutil-jdk18on.jar

다음 파일들은 각각 특정 수준 JMS 또는 Jakarta Messaging 레벨에 해당하는 클래스를 포함합니다:

- **JMS 2.0** com.ibm.mq.allclient.jar (JMS 2.0 그리고 JMS 1.1)
- **JM 3.0** com.ibm.mq.jakarta.client.jar (Jakarta Messaging 3.0)

기타 필수 JAR 파일:

- fscontext.jar 파일 시스템 컨텍스트를 사용하여 JNDI 조회 작업을 수행하는 애플리케이션의 경우 필수입니다.
- **JM 3.0** jakarta.jms-api.jar. 인터페이스 및 Jakarta Messaging 3.0 예외 정의가 포함되어 있습니다.
- **JMS 2.0** jms.jar. 인터페이스 및 JMS 2.0 예외 정의가 포함되어 있습니다.
- org.json.jar 필수. JSON 형식의 클라이언트 채널 정의(CCDT) 파일을 해석하고, 해당 com.ibm.mq.jms.MQSession com.ibm.mq.jms.MQConnection 객체의 toString() 메서드가 호출될 때 IBM MQ classes for Jakarta Messaging JSON 표현을 생성할 수 IBM MQ classes for JMS 있도록 하는 클래스를 포함합니다.
- providerutil.jar 파일 시스템 컨텍스트를 사용하여 JNDI 조회 작업을 수행하는 애플리케이션의 경우 필수입니다.

참고: **Stabilized** com.ibm.mq.allclient.jar 그리고 com.ibm.mq.jakarta.client.jar 둘 다. IBM MQ classes for Java의 사본을 포함합니다. 그러나 에서 IBM MQ 9.0, 이러한 클래스들은 에서 제공된 수준에서 기능적으로 IBM MQ 8.0 안정화되었다고 선언됩니다. 보세요 [사용 중단, 안정화 및 제거 IBM MQ 9.0](#).

wmq/OSGi

IBM MQ OSGi 클라이언트 번들:

- **JM 3.0** com.ibm.mq.jakarta.osgi.allclient_V.R.M.F.jar
- **JM 3.0** com.ibm.mq.jakarta.osgi.allclientprereqs_V.R.M.F.jar
- **JMS 2.0** com.ibm.mq.osgi.allclient_V.R.M.F.jar
- **JMS 2.0** com.ibm.mq.osgi.allclientprereqs_V.R.M.F.jar

여기서 V.R.M.F는 버전, 릴리스, 수정 및 수정팩 번호입니다.

프로시저

1. IBM MQ Java / JMS 클라이언트 JAR 파일을 Fix Central에서 다운로드하십시오.
 - a) 다음 링크를 클릭하십시오. [IBM MQ Java / JMS 클라이언트](#).
 - b) 표시된 사용 가능한 수정사항 목록에서 사용하는 IBM MQ 버전에 대한 클라이언트를 찾으십시오. 예를 들면, 다음과 같습니다.

```
release level: 10.0.0.0-IBM-MQ-Install-Java-All
Long Term Support: 10.0.0.0 IBM MQ JMS and Java 'All Client'
```

그런 다음, 클라이언트 파일 이름을 클릭하고 다운로드 프로세스를 수행하십시오.

2. 파일을 다운로드한 디렉터리에서 추출을 시작하십시오.
추출을 시작하려면 다음 형식으로 명령어를 입력하십시오:

```
java -jar V.R.M.F-IBM-MQ-Install-Java-All.jar
```

where *V.R.M.F* 는 제품 버전 번호이며, 예를 들어 *10.0.0.0* 또한 *V.R.M.F-IBM-MQ-Install-Java-All.jar* 는 에서 다운로드된 Fix Central파일의 이름입니다.

예를 들어, IBM MQ 10.0.0 릴리스용 클라이언트를 JMS 추출하려면 다음 명령을 사용합니다:

```
java -jar 10.0.0.0-IBM-MQ-Install-Java-All.jar
```

참고: 이 설치를 수행하려면 컴퓨터에 설치되어 시스템 경로에 JRE 추가된 가 있어야 합니다.

명령을 입력하면, 다음 정보가 표시됩니다.

IBM MQ (V10.0)를 사용, 압축 해제 또는 설치하기 전에 다음 사항에 동의해야 합니다

1.의 조건 IBM 평가용 국제 라이선스 계약서

프로그램 2. IBM 국제 프로그램 라이선스 계약 및 추가 사항

license information. Please read the following license agreements carefully.

The license agreement is separately viewable using the
--viewLicenseAgreement option.

Press Enter to display the license terms now, or 'x' to skip.

3. 라이선스 조항을 검토하고 이에 동의하십시오.

a) 라이선스를 표시하려면 Enter를 누르십시오.

또는 라이선스 표시를 건너뛰려면 x를 누르십시오.

라이선스가 표시된 후, 또는 x를 누르면 즉시 다음 메시지가 표시됩니다:

Additional license information is separately viewable using the
--viewLicenseInfo option.

Press Enter to display additional license information now, or 'x' to skip.

b) 추가 라이선스 조항을 표시하려면 Enter를 누르십시오.

또는 추가 라이선스 약관 표시를 건너뛰려면 x 키를 누르십시오.

추가 라이선스 약관이 표시된 후, 또는 x 키를 누르면 즉시 다음 메시지가 표시됩니다:

By choosing the "I Agree" option below, you agree to the terms of the
license agreement and non-IBM terms, if applicable. 이 매개변수를
agree, select "I do not Agree".

Select [1] I Agree, or [2] I do not Agree:

c) 라이선스 계약에 동의하고 설치 디렉토리 선택을 계속하려면 1을 선택하십시오.

또는 2를 선택하여 설치를 즉시 종료하십시오.

1을 선택하면 다음과 유사한 메시지가 표시됩니다:

Enter directory for product files or leave blank to accept the default value.
기본 대상 디렉터리는 H:\downloads입니다

Target directory for product files?

4. 추출을 위한 상위 디렉터를 지정하십시오.

기본 위치는 현재 디렉토리입니다.

- 기본 위치에 제품 파일을 추출하려면 값을 지정하지 않고 Enter 키를 누르십시오.
- 다른 위치로 제품 파일을 추출하려면 파일을 추출할 디렉터리 이름을 지정한 후 Enter 키를 눌러 추출을 시작하십시오.

지정하는 디렉터리 이름은 이미 존재해서는 안 됩니다. 그렇지 않으면 추출을 시작할 때 오류가 보고되고 파일이 설치되지 않습니다.

지정된 디렉터리가 아직 존재하지 않는 경우, 해당 디렉터리가 생성되고 프로그램 파일이 이 디렉터리로 추출됩니다. 설치 과정에서 지정한 상위 디렉터리 내에 라는 이름의 새 *wmq* 디렉터리가 생성됩니다.

세 개의 서브디렉터리(JavaEE, JavaSE및 OSGi)가 다음 콘텐츠로 *wmq* 디렉터리에 작성됩니다.

JavaEE

 *wmq.jakarta.jmsra.ivt.ear*

 wmq.jakarta.jmsra.rar

 wmq.jmsra.ivt.ear

 wmq.jmsra.rar

JavaSE

이 디렉토리에는 다음의 하위 디렉토리와 파일이 포함되어 있습니다:

JavaSE/lib

bcpkix-jdk18on.jar

bcprov-jdk18on.jar

bcutil-jdk18on.jar

 com.ibm.mq.allclient.jar

 com.ibm.mq.jakarta.client.jar

fscontext.jar

jms.jar

org.json.jar

providerutil.jar

JavaSE/bin

JMSAdmin.bat

JMSAdmin

JMSAdmin.config

OSGi

 com.ibm.mq.jakarta.osgi.allclient_V.R.M.F.jar

 com.ibm.mq.jakarta.osgi.allclientprereqs_V.R.M.F.jar

 com.ibm.mq.osgi.allclient_V.R.M.F.jar

 com.ibm.mq.osgi.allclientprereqs_V.R.M.F.jar

추출이 완료되면 다음과 같은 예시와 같이 확인 메시지가 표시됩니다:

파일을 H:\downloads\wmq로 추출합니다
Successfully extracted all product files.

허용 목록에 IBM MQ classes for JMS추가하기Jakarta Messaging

객체 직렬화 및 Java 역직렬화 메커니즘이 잠재적 보안 위험으로 확인되었습니다. 허용 목록을 IBM MQ classes for JMS 통해 일부 직렬화 위험으로부터 IBM MQ classes for Jakarta Messaging 어느 정도 보호 기능을 제공합니다.

이 태스크 정보

Java 오브젝트 직렬화 및 직렬화 해제 메커니즘은 직렬화 해제가 임의의 Java 오브젝트를 인스턴스화하므로 잠재적인 보안 위험으로 식별되었으며, 여기서 다양한 문제점을 유발하기 위해 악의적으로 전송된 데이터에 대해 잠재적입니다. 직렬화의 주목할 만한 응용 분야 중 하나는 임의의 객체를 캡슐화하고 [Jakarta Messaging 3.0](#) 그리고 Java Message Service 2.0ObjectMessages 전송하기 위해 직렬화를 사용하는입니다.

직렬화 허용 목록은 직렬화가 제기하는 일부 위험에 대한 잠재적인 완화입니다. ObjectMessage에서 캡슐화하고 이로부터 추출할 수 있는 클래스를 명시적으로 지정함으로써 허용 목록은 일부 직렬화 위험으로부터 일정한 보호를 제공합니다.

관련 개념

[106 페이지의 『아래에서 애플리케이션 IBM MQ classes for JMS 실행 Java security manager』](#)

IBM MQ classes for JMS는 Java security manager를 사용한 상태에서 실행할 수 있습니다. Java security manager를 사용한 상태로 애플리케이션을 실행하려면 적절한 정책 구성 파일로 JVM(Java Virtual Machine)을 구성해야 합니다.

허용 목록 개념

에서 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging, 인터페이스 JMSObjectMessage 구현 시 클래스 허용 목록 지원이 제공됩니다. 이는 객체 직렬화 및 Java 역직렬화 메커니즘과 잠재적으로 관련된 일부 보안 위험에 대한 완화 방안을 제공합니다.

허용 목록에 IBM MQ classes for JMS 포함 IBM MQ classes for Jakarta Messaging

중요사항:

가능한 용어 허용 목록은 용어 화이트리스트로 대체되었습니다. 여기에는 이 항목에서 언급된 일부 Java 시스템 속성 이름이 포함됩니다. 기존 구성을 변경할 필요는 없습니다. 이전 시스템 특성 이름도 계속해서 작동합니다.

IBM MQ classes for JMS (JMS 2.0) 및 IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) 는 `ObjectMessageJMS` 인터페이스 구현 시 클래스의 허용 목록 등록을 지원합니다.

- **JMS 2.0** 해당 속성 IBM MQ classes for JMS이름은 입니다 **com.ibm.mq.jms.allowlist.***.
- **JM 3.0** 해당 속성 IBM MQ classes for Jakarta Messaging이름은 다음과 같습니다. **com.ibm.mq.jakarta.jms.allowlist.***

허용 목록은 ObjectMessage.setObject()로 직렬화될 수 있고 ObjectMessage.getObject()로 역직렬화될 수 있는 Java 클래스를 정의합니다.

- **JMS 2.0** ObjectMessage가 있는 허용 목록에 포함되어 있지 않은 클래스의 인스턴스를 직렬화 또는 역직렬화하려고 시도하면 javax.jms.MessageFormatException이 해당 원인으로 java.io.InvalidClassException과 함께 처리되도록 유발할 수 있습니다.
- **JM 3.0** 허용 목록(allowlist)에 포함되지 않은 클래스의 인스턴스를 `ObjectMessage`로 직렬화하거나 역직렬화하려는 시도는 `jakarta.jms.MessageFormatException` 예외를 발생시키며, 그 원인은 `java.io.InvalidClassException`입니다.

허용 목록 생성

중요사항: IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 허용 목록으로 배포할 수 없습니다. ObjectMessages를 사용하여 전송될 클래스를 선택하는 것은 애플리케이션 디자인 선택사항이며 IBM MQ는 이를 미연에 방지할 수 없습니다.

이러한 이유로, 허용 목록 메커니즘이 두 가지 모드의 작동을 위해 허용됩니다.

DISCOVERY

이 모드에서는 메커니즘이 ObjectMessage에서 직렬화 또는 역직렬화되도록 관찰된 모든 클래스를 보고하여 완전한 클래스 이름의 목록을 생성합니다.

ENFORCEMENT

이 모드에서는 메커니즘이 허용 목록에 없는 클래스를 직렬화 또는 역직렬화하려는 시도를 거부하여 허용 목록을 강제 실행합니다.

이 메커니즘을 사용하려는 경우 현재 직렬화 및 역직렬화된 클래스의 목록을 수집하려면 처음에 DISCOVERY 모드에서 실행해야 하며 허용 목록에 대한 기준으로 목록을 검토하고 이를 사용해야 합니다. 변경되지 않은 목록을 사용하는 것이 적절할 수 있지만 이를 수행하기로 결정하기 전에 목록을 먼저 검토해야 합니다.

허용 목록 메커니즘 제어

허용 목록 메커니즘을 제어하기 위해 다음 세 가지 시스템 특성을 사용할 수 있습니다.

com.ibm.mq.jms.allowlist (JMS 2.0) 및 com.ibm.mq.jakarta.jms.allowlist (Jakarta Messaging 3.0)

이 특성은 다음 방법으로 지정할 수 있습니다.

- 파일 URI 형식(즉, file:(으)로 시작)으로 허용 목록을 포함하는 파일의 경로 이름. DISCOVERY 모드에서는 이 파일이 허용 목록 메커니즘에 의해 작성됩니다. 파일이 존재하지 않아야 합니다. 파일이 존재하는 경우, 메커니즘이 이를 덮어쓰기보다 예외를 처리합니다. ENFORCEMENT 모드에서는 이 파일이 허용 목록 메커니즘에 의해 읽힙니다.
- 허용 목록을 구성하는 심표로 구분된 완전한 클래스 이름.

이 특성이 설정되어 있지 않으면 허용 목록 메커니즘은 비활성입니다.

Java security manager를 사용 중인 경우 IBM MQ classes for JMS JAR 파일이 이 파일에 대한 읽기 및 쓰기 액세스를 갖고 있는지 확인해야 합니다.

com.ibm.mq.jms.allowlist.discover (JMS 2.0) 및 com.ibm.mq.jakarta.jms.allowlist.discover (Jakarta Messaging 3.0)

- 이 특성이 설정되지 않았거나 False로 설정된 경우, 허용 목록 메커니즘이 ENFORCEMENT 모드에서 실행됩니다.
- 이 특성이 True로 설정되어 있고 허용 목록이 파일 URI로 지정된 경우, 허용 목록 메커니즘이 DISCOVERY 모드에서 실행됩니다.
- 이 특성이 True로 설정되어 있고 허용 목록이 클래스 이름의 목록으로 지정된 경우, 허용 목록 메커니즘은 적절한 예외를 처리합니다.
- 이 속성이 true로 설정되어 있고 허용 목록이 `com.ibm.mq.jms.allowlist` 또는 `com.ibm.mq.jakarta.jms.allowlist` 속성을 사용하여 지정되지 않은 경우, 허용 목록 메커니즘은 비활성화됩니다.
- 이 특성이 true로 설정되어 있고 허용 목록 파일이 이미 있는 경우, 허용 목록 메커니즘은 `java.io.InvalidClassException`을 발생시키고 항목은 파일에 추가되지 않습니다.

com.ibm.mq.jms.allowlist.mode (JMS 2.0) 및 com.ibm.mq.jakarta.jms.allowlist.mode (Jakarta Messaging 3.0)

다음 세 가지 방법 중 하나로 이 문자열 특성을 지정할 수 있습니다.

- 이 특성이 SERIALIZE로 설정된 경우에는 ENFORCEMENT 모드가 `ObjectMessage.setObject()` 메소드에서만 허용 목록 유효성 검증을 수행합니다.
- 이 특성이 DESERIALIZE로 설정된 경우에는 ENFORCEMENT 모드가 `ObjectMessage.getObject()` 메소드에서만 허용 목록 유효성 검증을 수행합니다.
- 이 특성이 설정되지 않거나 기타 값으로 설정된 경우에는 ENFORCEMENT 모드가 `ObjectMessage.getObject()` 메소드와 `ObjectMessage.setObject()` 메소드에서 허용 목록 유효성 검증을 수행합니다.

허용 목록 파일의 형식

이는 허용 목록 파일의 형식의 주요 기능입니다.

- 허용 목록 파일은 플랫폼별 줄 바꿈이 있는 기본 플랫폼 파일 인코딩입니다.

참고: 허용 목록 파일이 사용 중인 경우 이 파일은 항상 JVM에 대한 기본 파일 인코딩을 사용하여 기록되고 읽힙니다.

다음과 같은 방법으로 허용 목록 파일이 생성되면 괜찮습니다.

-  **z/OS** z/OS에서 실행되는 독립형 애플리케이션이 생성하여, 마찬가지로 z/OS에서 실행되는 다른 독립형 애플리케이션이 이를 사용합니다.
- 임의의 플랫폼에서 WebSphere Application Server의 내부에서 실행되는 애플리케이션이 생성하여 WebSphere Application Server의 다른 인스턴스에서 이를 사용합니다.
-  **Multi** IBM MQ for Multiplatforms에서 실행되는 독립형 애플리케이션에서 생성되며, IBM MQ for Multiplatforms에서 실행되는 기타 독립형 애플리케이션에서 사용되거나 임의의 플랫폼의 WebSphere Application Server 내부에서 실행되는 애플리케이션에서 사용됩니다.

그러나 WebSphere Application Server는 ASCII를 사용하고 독립형 JVM은 EBCDIC을 사용하므로 다음과 같은 방식으로 허용 목록 파일이 생성되는 경우 파일 인코딩 문제가 발생합니다.

- z/OS에서 생성되고, z/OS 이외의 플랫폼에서 실행되는 독립형 애플리케이션이나 WebSphere Application Server에서 사용됩니다.
 - WebSphere Application Server 또는 z/OS 이외의 플랫폼에서 실행되는 독립형 애플리케이션에서 생성되며, z/OS의 독립형 애플리케이션에서 사용됩니다.
 - 비어 있지 않은 각 행에는 완전한 클래스 이름이 포함되어 있습니다. 빈 행은 무시됩니다.
 - 주석이 포함될 수 있으며 행 끝에 오는 '#' 문자는 무시됩니다.
 - 아주 기본적인 와일드카드 메커니즘이 있습니다.
 - '*'는 클래스 이름의 **마지막** 요소일 수 있습니다.
 - '*'는 클래스 이름, 즉 패키지의 일부가 아닌 클래스의 **단일** 요소와 일치합니다.
- 따라서 `com.ibm.mq.*`은(는) `com.ibm.mq.MQMessage`에 일치하지만 `com.ibm.mq.jmqi.remote.api.RemoteFAP`에는 일치하지 않습니다.
- 와일드카드는 명확한 패키지 이름 없이 클래스용 기본 패키지에서 클래스에 대해 작동하지 않으므로 "*"의 클래스 이름은 거부됩니다.
- 잘못된 형식화된 허용 목록 파일. 예를 들어, 와일드카드가 마지막 요소가 아닌 `com.ibm.mq.*.Message`과 (와) 같은 항목이 포함된 파일은 `java.lang.IllegalArgumentException`을 발생시킵니다.
 - 빈 허용 목록 파일은 `ObjectMessage`의 사용이 전적으로 불가능하도록 하는 데 영향을 줍니다.

심표로 구분된 목록으로 허용 목록의 형식

동일한 와일드카드 메커니즘은 심표로 구분된 목록으로 허용 목록에 대해 사용 가능합니다.

- '*'는 명령행 또는 셸 스크립트 또는 배치 파일에 지정된 경우 운영 체제에 의해 확장될 수 있으므로 특별한 처리가 필요할 수 있습니다.
- '#' 주석 문자는 파일이 지정된 경우에만 적용 가능합니다. 허용 목록이 심표로 구분된 클래스 이름 목록으로 지정된 경우, 운영 체제나 셸이 이를 처리하지 않는다고 가정할 때(많은 AIX and Linux 셸에서 기본 주석 문자로 사용됨), 이는 일반 문자로 처리됩니다.

허용 목록은 언제 발생합니까?

허용 목록은 애플리케이션이 처음으로 `ObjectMessage setMessage()` 또는 `getMessage()` 메소드를 실행할 때 시작됩니다.

메커니즘이 초기화될 때 시스템 특성이 평가되고, 허용 목록 파일이 열려 있고 ENFORCEMENT 모드에 있고, 허용 목록 클래스의 목록이 로드됩니다. 이 시점에서 항목은 애플리케이션에 대한 IBM MQ JMS 로그 파일에 기록됩니다.

메커니즘이 초기화되면 해당 매개변수가 변경되지 않을 수 있습니다. 초기화 시점은 애플리케이션 동작에 따라 달려 있으므로 쉽게 예측되지 않습니다. 시스템 특성 설정 및 허용 목록 파일 콘텐츠는 그러므로 애플리케이션이 시작된 시점으로부터 수정된 것으로 간주되어야 합니다. 애플리케이션이 실행 중인 동안 결과가 보장되지 않으므로 허용 목록 파일의 특성 또는 콘텐츠를 변경하지 마십시오.

고려할 사항

Java 직렬화 메커니즘에 내재되어 있는 위험성을 완화하는 최적의 접근법은 데이터 전송에 대한 대체 접근법(예: `ObjectMessage` 대신 JSON 사용)을 탐색하는 것입니다. Advanced Message Security(AMS) 메커니즘을 사용할 경우 신뢰할 수 있는 소스로부터 메시지가 제공되도록 보장함으로써 추가적인 보안이 추가될 수 있습니다.

애플리케이션과 함께 Java security manager 메커니즘을 사용할 경우, 다음 권한을 부여해야 합니다.

- 사용하는 모든 허용 목록 파일에 대한 `FilePermission`과 ENFORCEMENT 모드의 읽기 권한 및 DISCOVER 모드의 쓰기 권한.
- **JMS 2.0** `com.ibm.mq.jms.allowlist`, `com.ibm.mq.jms.allowlist.discover` 및 `com.ibm.mq.jms.allowlist.mode` 특성에서의 `PropertyPermission`(읽기).

- **JM 3.0** `com.ibm.mq.jakarta.jms.allowlist`,
`com.ibm.mq.jakarta.jms.allowlist.discover` 및
`com.ibm.mq.jakarta.jms.allowlist.mode` 특성에서의 PropertyPermission(읽기).

자세한 정보

허용 목록에 대한 자세한 정보는 132 페이지의 『허용 Jakarta MessagingJMS 목록 설정 및 사용』 및 134 페이지의 『WebSphere Application Server의 허용 목록』의 내용을 참조하십시오.

관련 개념

106 페이지의 『아래에서 애플리케이션 IBM MQ classes for JMS 실행 Java security manager』

IBM MQ classes for JMS는 Java security manager를 사용한 상태에서 실행할 수 있습니다. Java security manager를 사용한 상태로 애플리케이션을 실행하려면 적절한 정책 구성 파일로 JVM(Java Virtual Machine)을 구성해야 합니다.

허용 Jakarta MessagingJMS 목록 설정 및 사용

이 정보는 허용 목록이 어떻게 작동하는지, 그리고 애플리케이션이 처리할 수 있는 ObjectMessages 유형 목록을 포함하는 허용 목록 파일을 생성하기 위해 IBM MQ classes for JMS IBM MQ classes for Jakarta Messaging 또는 예 포함된 기능을 사용하여 허용 목록을 설정하는 방법을 설명합니다.

시작하기 전에

중요사항:

가능하면 용어 허용 목록은 용어 화이트리스트로 대체되었습니다. 여기에는 이 항목에서 언급된 일부 Java 시스템 속성 이름이 포함됩니다. 기존 구성을 변경할 필요는 없습니다. 이전 시스템 특성 이름도 계속해서 작동합니다.

이 태스크를 시작하기 전에 129 페이지의 『허용 목록 개념』의 내용을 읽고 이해해야 합니다.

이 태스크 정보

와 Jakarta Messaging 는 공통점이 많으므로 JMS , 이 주제에서 예 JMS 대한 언급은 두 가지 모두를 가리키는 것으로 볼 수 있습니다. 필요한 경우 차이점을 강조 표시했습니다.

허용 목록 기능을 사용하는 경우 IBM MQ classes for JMS에서 이 기능을 다음과 같은 방식으로 사용합니다.

- 애플리케이션에서 ObjectMessage를 전송하려는 경우 다음을 호출하여 두 가지 방법 중 하나로 작성할 수 있습니다.
 - Session.createObjectMessage(Serializable) 메소드. 메시지에 포함할 오브젝트를 전달합니다.
 - Session.createObjectMessage() 메소드. 빈 ObjectMessage를 작성한 후 ObjectMessage.setObject(Serializable)를 호출하여 ObjectMessage 내 전송할 오브젝트를 저장합니다.

Session.createObjectMessage(Serializable) 또는 ObjectMessage.setObject(Serializable) 메소드가 호출되면 JMS에 대한 클래스가 허용 목록에 언급된 유형으로 전달되는 오브젝트인지 여부를 확인합니다.

언급된 유형인 경우 오브젝트가 직렬화되고 ObjectMessage에 저장됩니다. 그러나 오브젝트가 허용 목록에 있는 유형이 아니면 IBM MQ classes for JMS에서 다음 메시지를 포함하는 JMSEException이 발생합니다.

```
JMSCC0052: An exception occurred while serializing the object:
'java.io.InvalidClassException: <object class>; The class may not be serialized
or deserialized as it has not been included in the allowlist '<allowlist>'.
```

애플리케이션에서 해당 예외가 다시 발행됩니다.

중요사항: Session.createObjectMessage(Serializable) 메소드에서 예외가 발생하면 ObjectMessage가 작성되지 않습니다. 마찬가지로, ObjectMessage.setObject(Serializable) 메소드에서 JMSEException이 발생하면 오브젝트가 ObjectMessage에 추가되지 않습니다.

- 애플리케이션에서 ObjectMessage를 수신하면 ObjectMessage.getObject() 메소드를 호출하여 여기에 포함된 오브젝트를 가져옵니다. 이 메소드가 호출된 경우 IBM MQ classes for JMS에서는 ObjectMessage에 포함된 오브젝트 유형을 확인하여 허용 목록에 지정된 유형의 오브젝트인지 확인합니다.

이 경우 오브젝트가 직렬화 해제되고 애플리케이션으로 반환됩니다. 그러나 오브젝트가 허용 목록에 있는 유형이 아니면 IBM MQ classes for JMS에서 다음 메시지를 포함하는 JMSEException이 발생합니다.

```
JMSCC0053: An exception occurred while deserializing a message:
'java.io.InvalidClassException: <object class>; The class may not be
serialized or deserialized as it has not been included in the
허용 목록 '<허용 목록>'. '.
```

애플리케이션에서 해당 예외가 다시 발행됩니다.

예를 들어, 애플리케이션이 `java.net.URL` 유형의 오브젝트를 포함하는 `ObjectMessage`를 전송하기 위해 다음 코드를 포함하고 있다고 가정합니다.

```
java.net.URL testURL = new java.net.URL("https://www.ibm.com/");
ObjectMessage msg = session.createObjectMessage(testURL);
sender.send(msg);
```

허용 목록을 사용할 수 없으면 애플리케이션은 필요한 목적지로 메시지를 넣을 수 있습니다.

단일 항목을 포함하는 `C:\allowlist.txt` 라는 파일을 생성하고, 시스템 Java 속성을 설정하여 `java.net.URL` 애플리케이션을 다시 시작하면:

```
-Dcom.ibm.mq.jms.allowlist=file:/C:/allowlist.txt
```

허용 목록 기능이 사용 가능합니다. 해당 유형이 허용 목록에 지정되어 있으므로 애플리케이션에서 여전히 `java.net.URL` 유형의 오브젝트를 포함하는 `ObjectMessage`를 작성하고 전송할 수 있습니다.

그러나 파일을 `allowlist.txt` 수정하여 파일에 단일 항목만 포함되도록 `java.util.Calendar` 하면, 허용 목록 기능이 여전히 활성화되어 있으므로 애플리케이션이 다음을 호출할 때:

```
ObjectMessage msg = session.createObjectMessage(testURL);
```

IBM MQ classes for JMS에서는 허용 목록을 검사하고 `java.net.URL`에 대한 항목을 포함하지 않음을 확인합니다.

결과적으로 JMSCC0052 메시지를 포함하는 JMSEException이 발생합니다.

마찬가지로 이 코드를 사용하는 `ObjectMessages`를 수신하는 다른 애플리케이션이 있다고 가정합니다.

```
ObjectMessage message = (ObjectMessage)receiver.receive(30000);
if (message != null) {
    Object messageBody = objectMessage.getObject();
    if (messageBody instanceof java.net.URL) {
        :      :      :      :      :      :
    }
```

허용 목록을 사용할 수 없으면 애플리케이션은 임의의 유형에 해당하는 오브젝트를 포함하는 `ObjectMessage`를 수신할 수 있습니다. 그런 다음, 적절한 처리를 수행하기 전에 오브젝트 유형이 `java.net.URL`인지 확인합니다.

이제 Java 시스템 특성으로 애플리케이션을 시작하는 경우:

```
-Dcom.ibm.mq.jms.allowlist=java.net.URL
```

허용 목록 기능이 켜집니다. 애플리케이션이 호출되면:

```
Object messageBody = objectMessage.getObject();
```

`ObjectMessage.getObject()` 메소드는 `java.net.URL` 유형의 오브젝트만 반환합니다.

`ObjectMessage`에 포함된 오브젝트가 이 유형이 아니면 `ObjectMessage.getObject()` 메소드에서 JMSCC0053 메시지를 포함하는 JMSEException이 발생합니다. 그런 다음, 애플리케이션은 메시지에서 수행할 작업을 결정해야 합니다. 예를 들어, 메시지를 해당 큐 관리자의 데드-레터 큐로 이동할 수 있습니다.

`ObjectMessage` 내 오브젝트가 `java.net.URL`인 경우에만 보통 애플리케이션에서 리턴합니다.

프로시저

1. 다음 Java 시스템 속성을 지정하여 `ObjectMessages`, 를 처리하는 애플리케이션을 실행하십시오:

```
-Dcom.ibm.mq.jms.allowlist.discover=true
-Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
```

애플리케이션이 실행되면 IBM MQ classes for JMS에서는 애플리케이션이 처리하는 오브젝트 유형이 포함된 파일을 작성합니다.

2. 애플리케이션이 일정 기간에 걸쳐 ObjectMessage의 대표 샘플을 처리한 후 중지하십시오.

이제 허용 목록 파일은 실행 중에 애플리케이션이 처리한 ObjectMessage 내 포함된 오브젝트의 모든 유형을 포함합니다.

충분한 시간 동안 애플리케이션을 실행한 경우 이 목록에는 애플리케이션이 처리할 가능성이 큰 ObjectMessage 내 포함된 모든 가능한 오브젝트 유형이 포함되어 있습니다.

3. 다음 시스템 특성을 설정하고 애플리케이션을 재시작하십시오.

```
-Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
```

이를 통해 허용 목록을 사용할 수 있으며, IBM MQ classes for JMS에서 허용 목록에 없는 유형의 ObjectMessage를 감지하면 JMSCC0052 또는 JMSCC0053 메시지를 포함하는 JMSException이 발생합니다.

WebSphere Application Server의 허용 목록

허용 목록을 WebSphere Application Server IBM MQ classes for JMS 사용하는 방법

중요사항:

가능하면 용어 허용 목록은 용어 화이트리스트로 대체되었습니다. 여기에는 이 항목에서 언급된 일부 Java 시스템 속성 이름이 포함됩니다. 기존 구성을 변경할 필요는 없습니다. 이전 시스템 특성 이름도 계속해서 작동합니다.

WebSphere Application Server 설치에 허용 목록을 지원하는 IBM MQ 자원 어댑터의 버전이 포함되는지 확인해야 합니다.

두 제품 사용에 대한 추가 정보는 [470 페이지의 『IBM MQ 및 WebSphere Application Server 함께 사용』](#)의 내용을 참조하십시오.

IBM MQ 9.0.0 Fix Pack 1 이상은 적절한 기능을 포함합니다.

애플리케이션 서버를 업데이트하면 Java 시스템 특성을 사용할 수 있습니다.

- -Dcom.ibm.mq.jms.allowlist
- -Dcom.ibm.mq.jms.allowlist.discover

[132 페이지의 『허용 Jakarta MessagingJMS 목록 설정 및 사용』](#)에 설명되어 있습니다.

참고: 애플리케이션 서버를 실행하는 데 사용되는 Java Virtual Machine 시스템에 시스템 Java 속성을 일반적인 JVM 인자로 설정해야 하며, 변경 사항이 적용되려면 애플리케이션 서버를 재시작해야 합니다.

자세한 정보는 [Java 가상 머신 설정](#)의 일반 JVM 인수에 관한 절을 참조하십시오.

속성을 설정하려면 프로세스 정의의 창으로 Java Virtual Machine 이동하여 적절한 인수를 입력하십시오.

다음 설정을 사용합니다.

```
-Dcom.ibm.mq.jms.allowlist=<youruserId>_MyObject
```

그러면, 애플리케이션 서버에서 허용 목록 `youruserId_MyObject`를 사용합니다. 해당 유형의 오브젝트만 애플리케이션 서버에서 처리됩니다.

다음 설정을 사용합니다.

```
-Dcom.ibm.mq.jms.allowlist.discover=true
-Dcom.ibm.mq.jms.allowlist=file:C:/allowlist.txt
```

발견 모드를 사용하도록 애플리케이션 서버를 구성하고 애플리케이션 서버가 처리하는 JMS ObjectMessage의 세부사항을 `C:\allowlist.txt` 파일에 기록합니다.

다음 설정을 사용합니다.

```
-Dcom.ibm.mq.jms.allowlist=file:C:/allowlist.txt
```

애플리케이션 서버가 C:/allowlist.txt 파일을 로드하고 해당 파일의 정보를 사용하여 허용 목록을 판별하게 합니다.

관련 개념

106 페이지의 『아래에서 애플리케이션 IBM MQ classes for JMS 실행 Java security manager』

IBM MQ classes for JMS는 Java security manager를 사용한 상태에서 실행할 수 있습니다. Java security manager를 사용한 상태로 애플리케이션을 실행하려면 적절한 정책 구성 파일로 JVM(Java Virtual Machine)을 구성해야 합니다.

IBM MQ classes for JMS에서 문자열 변환

IBM MQ classes for JMS에서는 문자 문자열 변환을 위해 CharsetEncoders 및 CharsetDecoders를 직접 사용합니다. 문자 문자열 변환을 위한 기본 동작은 두 개의 시스템 특성으로 구성할 수 있습니다. 맵핑할 수 없는 문자가 포함되는 메시지 처리는 UnmappableCharacterAction 및 대체 바이트를 설정하기 위한 메시지 특성을 통해 구성할 수 있습니다.

IBM MQ 8.0 이전에는 IBM MQ classes for JMS의 문자열 변환이 `java.nio.charset.Charset.decode(ByteBuffer)` 및 `Charset.encode(CharBuffer)` 메소드를 호출하여 수행되었습니다.

이러한 방법 중 하나를 사용하면 형식이 잘못되거나 변환 불가능한 데이터의 기본 대체(REPLACE)가 발생합니다. 이 작동으로 인해 애플리케이션의 오류가 모호해질 수 있으며, 변환된 데이터에서 예상치 못한 문자(예: ?)가 발생할 수 있습니다.

IBM MQ 8.0부터는, 이러한 문제를 더 빠르고 효율적으로 감지하기 위해 IBM MQ classes for JMS이(가) CharsetEncoders 및 CharsetDecoders를 직접 사용하여 잘못된 형식의 데이터 및 변환 불가능한 데이터의 처리를 명시적으로 구성합니다. 기본 동작은 적합한 MQException을 발생시켜 이러한 문제를 REPORT하는 것입니다.

구성

UTF-16(Java에서 사용되는 문자 표현)에서 고유 문자 세트(예: UTF-8)로의 변환을 *encoding*이라고 하고 반대 방향으로의 변환을 *decoding*이라고 합니다.

코드 번역은 CharsetDecoders의 기본 동작을 수행하며 예외를 발생시켜 오류를 보고합니다.

하나의 설정을 사용하여 인코딩 및 디코딩 모두에서 오류 핸들링을 제어하기 위한 `java.nio.charset.CodingErrorAction`을 지정합니다. 다른 설정은 인코딩 시에 대체 바이트를 제어하는 데 사용됩니다. 기본 Java 대체 문자열은 디코딩 조작에서 사용됩니다.

IBM MQ classes for JMS에서의 UnmappableCharacterAction 및 대체 바이트 설정

IBM MQ 8.0부터는 다음 두 특성을 사용하여 UnmappableCharacterAction 및 대체 바이트를 설정합니다. 해당 상수 정의는 `com.ibm.msg.client.wmq.WMQConstants`에 있습니다.

JMS_IBM_UNMAPPABLE_ACTION

인코딩 또는 디코딩 조작에서 문자가 맵핑될 수 없을 때 적용할 CodingErrorAction을 설정하거나 가져옵니다.

다음과 같이 `CodingErrorAction.{REPLACE|REPORT|IGNORE}.toString()` 로 설정해야 합니다.

```
public static final String JMS_IBM_UNMAPPABLE_ACTION = "JMS_IBM_Unmappable_Action";
```

JMS_IBM_UNMAPPABLE_REPLACEMENT

문자가 인코딩 조작에서 맵핑될 수 없을 때 적용할 대체 바이트를 설정하거나 가져옵니다.

기본 Java 대체 문자열이 디코딩 조작에서 사용됩니다.

```
public static final String JMS_IBM_UNMAPPABLE_REPLACEMENT = "JMS_IBM_Unmappable_Replacement";
```


JMS_IBM_UNMAPPABLE_ACTION 및 JMS_IBM_UNMAPPABLE_REPLACEMENT 특성이 목적지 또는 메시지에서 설정될 수 있습니다. 메시지에 설정된 값은 메시지가 송신되는 목적지에서 설정된 값을 대체합니다.

참고로, JMS_IBM_UNMAPPABLE_REPLACEMENT는 단일 바이트로 설정되어야 합니다.

시스템 기본값 설정을 위한 시스템 특성

IBM MQ 8.0부터 문자열 변환과 관련된 기본 동작을 구성하는 데 다음 두 가지 Java 시스템 특성을 사용할 수 있습니다.

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction

인코딩 및 디코딩에서 변환 불가능한 데이터에 대해 취하는 조치를 지정합니다. 값은 REPORT, REPLACE 또는 IGNORE일 수 있습니다.

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement

문자가 인코딩 조작에서 맵핑될 수 없을 때 적용할 대체 바이트를 설정하거나 가져옵니다. 기본 Java 대체 문자열이 디코딩 조작에서 사용됩니다.

Java 문자 및 고유 바이트 표현 간의 혼란을 피하려면, 고유 문자 세트에서 대체 바이트를 표시하는 10진수로 `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement`를 지정해야 합니다.

예를 들어, 고유 문자 세트가 ASCII 기반(예: ISO-8859-1)인 경우에 고유 바이트로서 ?의 10진수는 63입니다. 한편 고유 문자 세트가 EBCDIC인 경우에 이는 111입니다.

참고: MQMD 또는 MQMessage 오브젝트에서 `unmappableAction` 또는 `unMappableReplacement` 필드가 설정된 경우 이 필드 값이 Java 시스템 특성보다 우선됩니다. 이를 통해 필요한 경우 각 메시지에 대해 Java 시스템 특성에서 지정한 값을 대체할 수 있습니다.

관련 개념

[323 페이지의 『IBM MQ classes for Java에서 문자열 변환』](#)

IBM MQ classes for Java에서는 문자 문자열 변환을 위해 `CharsetEncoders` 및 `CharsetDecoders`를 직접 사용합니다. 문자 문자열 변환을 위한 기본 동작은 두 개의 시스템 특성으로 구성할 수 있습니다. 맵핑할 수 없는 문자가 포함되는 메시지 처리는 `com.ibm.mq.MQMD`를 통해 구성할 수 있습니다.

작성 IBM MQ classes for JMS/Jakarta Messaging 신청서

모델에 JMS 대한 간략한 소개 후, 본 절에서는 및 IBM MQ classes for Jakarta Messaging 응용 프로그램을 IBM MQ classes for JMS 작성하는 방법에 대한 상세한 지침을 제공합니다.

이 태스크 정보

JM 3.0 IBM MQ 9.3.0부터, Jakarta Messaging 3.0 는 새로운 애플리케이션 개발에 지원됩니다. IBM MQ 9.3.0 그리고 향후에도 기존 애플리케이션에 대한 지원을 JMS 2.0 계속할 예정입니다. 동일한 애플리케이션에서 두 JMS 2.0/Jakarta Messaging 3.0 API를 모두 사용하는 것은 지원되지 않습니다. 자세한 내용은 [‘JMS/Jakarta Messaging용 IBM MQ 클래스 사용’](#)을 참조하십시오.

관련 개념

[IBM MQ classes for Jakarta Messaging개요](#)

와 Jakarta MessagingJMS 모델

와 Jakarta MessagingJMS 모델은 애플리케이션이 메시징 작업을 수행하는 Java 데 사용할 수 있는 일련의 인터페이스를 정의합니다. IBM MQ classes for JMS 그리고 둘 다 IBM MQ classes for Jakarta Messaging 메시징 서비스 제공업체입니다. 그들은 객체와 Jakarta MessagingJMS 개념이 IBM MQ 어떻게 관련되는지를 정의합니다. 해당 Jakarta MessagingJMS 사양은 특정 JMS 객체와 Jakarta Messaging 객체가 관리 대상 객체여야 한다고 규정합니다.

JMS 2.0 IBM MQ 8.0 버전의 JMSJMS 2.0 표준에 대한 지원을 추가했으며, 이는 단순화된 API를 도입하는 동시에 기존의 클래식 JMS 1.1API도 유지합니다.

JM 3.0 부터 IBM MQ 9.3.0, Jakarta Messaging 3.0 는 새로운 애플리케이션 개발을 지원합니다. IBM MQ 9.3.0 기존 애플리케이션에 대한 지원을 JMS 2.0 계속하고 있습니다. 동일한 애플리케이션에서 두 Jakarta Messaging 3.0 JMS 2.0 API를 모두 사용하는 것은 지원되지 않습니다.

참고: 이 경우 Jakarta Messaging 3.0, JMS 사양에 대한 제어권이 에서 Oracle 로 넘어갑니다 Java Community Process. 그러나 “javax”라는 이름에 대한 통제권은 여전히 Oracle 유지하고 있으며, 이 이름은 커뮤니티 Java 프로세스로 전환되지 않은 다른 Java 기술들에서도 사용되고 있습니다. 따라서, 는 기능적으로는 Jakarta Messaging 3.0 와 동일하지만 명명 방식에는 JMS 2.0 몇 가지 차이가 있습니다:

- 3.0 버전의 공식 명칭은 이지, Jakarta Messaging 가 Java Message Service아닙니다.
- 패키지 및 상수 이름 앞에는 가 붙으며 jakarta , 는 붙지 javax않습니다. 예를 들어, 메시징 서비스 제공자와의 초기 연결은 JMS 2.0 javax.jms.Connection 객체이고, 에서는 Jakarta Messaging 3.0 jakarta.jms.Connection 객체입니다.

JMS 2.0 javax.jms 패키지는 JMS 인터페이스를 정의하며, 각 JMS 제공자는 특정 메시징 제품에 대해 이러한 인터페이스를 구현합니다. IBM MQ classes for JMS 는 의 JMS 인터페이스를 IBM MQ구현하는 JMS 제공자입니다.

JM 3.0 jakarta.jms 패키지는 Jakarta Messaging 인터페이스를 정의하며, 각 Jakarta Messaging 제공자는 특정 메시징 제품에 대해 이러한 인터페이스를 구현합니다. IBM MQ classes for Jakarta Messaging 는 의 Jakarta Messaging 인터페이스를 IBM MQ구현하는 Jakarta Messaging 제공자입니다.

와 Jakarta Messaging 는 공통점이 많으므로 JMS , 이 주제에서 에 JMS 대한 언급은 두 가지 모두를 가리키는 것으로 볼 수 있습니다. 필요한 경우 차이점을 강조 표시했습니다.

간소화된 API

JMS 2.0 간소화된 API를 도입하면서, 동시에. JMS 1.1에서 도메인 특정 및 도메인 독립 인터페이스를 유지했습니다. 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 콘텐츠인 '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(100000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

JMS 2.0 또한 여러 소비자 간에 메시지를 공유할 수 있는 공유 구독 기능을 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 9에서는 이러한 오브젝트와 해당 관계를 표시합니다.

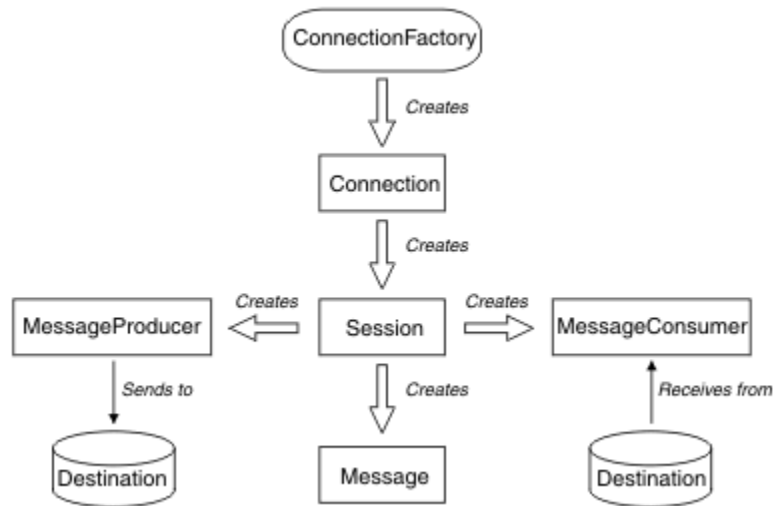


그림 9. JMS 오브젝트 및 해당 관계

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.

메시징 도메인

JMS 두 가지 메시징 방식을 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.

이전 JMS 버전에서는 JMS 1.1, 지점 간 도메인 프로그래밍에는 한 세트의 인터페이스와 메서드가 사용되고, 계시/구독 도메인 프로그래밍에는 다른 세트가 사용됩니다. 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 15에서는 JMS 도메인 독립적 인터페이스와 대응하는 도메인 특정 인터페이스를 나열합니다.

표 15. JMS 도메인 독립적 인터페이스 및 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 기존 JMS 1.1 도메인별 인터페이스와 간소화된 API를 모두 JMS 2.0지원합니다. IBM MQ classes for JMS 2.0 따라서 기존 애플리케이션의 유지보수 및 기존 애플리케이션 내 신규 기능 개발에 활용될 수 있습니다.

JM 3.0 IBM MQ classes for Jakarta Messaging 3.0 동일한 인터페이스의 버전을 Jakarta Messaging 지원하며, 새로운 애플리케이션 개발에 권장됩니다.

와 IBM MQ classes for Jakarta Messaging IBM MQ classes for JMS 에서 JMS 객체들은 다음과 같은 방식으로 개념들과 IBM MQ 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.

또는 를 IBM MQ classes for JMS 사용할 IBM MQ classes for Jakarta Messaging 때, 의 모든 일반 IBM MQ 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.

Destination 인터페이스의 IBM MQ classes for JMSIBM MQ classes for Jakarta Messaging 구현체는 Queue와 Topic의 추상 상위 클래스이므로, Destination의 인스턴스는 Queue 객체이거나 Topic 객체입니다. 도메인 독립적 인터페이스에서는 큐나 토픽을 목적지로 처리합니다. MessageProducer 또는 MessageConsumer 오브젝트의 메시징 도메인은 목적지가 큐인지 아니면 토픽인지에 따라 결정됩니다.

따라서 IBM MQ classes for Jakarta MessagingIBM MQ classes for JMS , 다음과 같은 유형의 객체들은 관리 가능한 객체들입니다:

- ConnectionFactory
- QueueConnectionFactory

- TopicConnectionFactory
- 큐
- 토픽
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

관련 개념

IBM MQ Java 언어 인터페이스

199 페이지의 『연결 팩토리 및 목적지 작성 및 구성』

애플리케이션은 IBM MQ classes for JMS IBM MQ classes for Jakarta Messaging JNDI(Naming and Directory Java Interface) 네임스페이스에서 관리 객체로 검색하거나, JAR 확장 또는 JAR 확장 기능을 IBM JMS IBM MQ JMS 사용하여 연결 팩토리 및 목적지를 생성할 수 있습니다. 또한 애플리케이션은 IBM JMS 확장 또는 IBM MQ JMS 확장을 사용하여 연결 팩토리 및 목적지의 특성을 설정할 수도 있습니다.

JMS 메시지

JMS 메시지는 헤더, 특성 및 본문으로 구성됩니다. JMS는 5가지 유형의 메시지 본문을 정의합니다.

JMS 메시지는 다음 부분으로 구성됩니다.

헤더

모든 메시지는 동일한 헤더 필드 세트를 지원합니다. 헤더 필드는 클라이언트와 제공자가 모두 메시지를 식별하고 라우트하는 데 사용하는 값을 포함합니다.

특성

각 메시지에는 애플리케이션 정의 특성 값을 지원하는 내장된 기능이 있습니다. 특성은 애플리케이션 정의 메시지를 필터하는 데 효율적인 메커니즘을 제공합니다.

본문

JMS는 현재 사용 중인 대부분의 메시징 스타일을 포함하는 5가지 유형의 메시지 본문을 정의합니다.

스트림

Java 기본 노드 값의 스트림입니다. 순차적으로 입력되고 읽습니다.

맵

이름-값 쌍 세트입니다. 여기서 이름은 문자열이고 값은 Java 기본 유형입니다. 이름별로 순차적으로 또는 무작위로 항목에 액세스할 수 있습니다. 항목의 순서는 정의되어 있지 않습니다.

텍스트

java.lang.String을 포함하는 메시지입니다.

오브젝트

순차적 실행 Java 오브젝트를 포함하는 메시지입니다.

바이트

해석되지 않은 바이트 스트림입니다. 이 메시지는 글자 그대로 기존 메시지 형식과 일치하도록 본문을 인코딩하는 데 사용합니다.

JMSCorrelationID 헤더 파일을 사용하여 한 메시지를 다른 메시지와 링크합니다. 일반적으로 응답 메시지를 요청 메시지와 링크합니다. JMSCorrelationID는 제공자 특정 메시지 ID, 애플리케이션 특정 문자열 또는 제공자 고유 byte[] 값을 보유할 수 있습니다.

JMS의 메시지 선택자

메시지에는 애플리케이션 정의 특성 값이 포함될 수 있습니다. 애플리케이션에서 메시지 선택자를 사용하여 JMS 제공자가 메시지를 필터링하게 할 수 있습니다.

메시지에는 애플리케이션 정의 특성 값을 지원하는 내장된 기능이 있습니다. 따라서 이 기능은 애플리케이션 특정 헤더 필드를 메시지에 추가하는 메커니즘을 제공합니다. 특성을 사용하면 애플리케이션이 메시지 선택자를 사용하여 JMS 제공자가 애플리케이션 특정 기준을 이용하여 대신 메시지를 선택하거나 필터링하게 할 수 있습니다. 애플리케이션 정의 특성은 다음 규칙을 따라야 합니다.

- 특성 이름은 메시지 선택자 ID에 대한 규칙을 따라야 합니다.
- 특성 값은 Boolean, byte, short, int, long, float, double 및 String입니다.

- JMSX 및 JMS_ 이름 접두부가 예약됩니다.

메시지를 송신하기 전에 특성 값이 설정됩니다. 클라이언트가 메시지를 수신할 때 메시지 특성은 읽기 전용입니다. 클라이언트가 이 지점에서 특성을 설정하려고 하면 `MessageNotWriteableException`이 처리됩니다. `clearProperties`를 호출하면 이제 특성을 읽고 쓸 수 있습니다.

특성 값이 메시지 본문의 값을 복제할 수 있습니다. JMS에서는 특성으로 작성될 수 있는 정책을 정의하지 않습니다. 그러나 애플리케이션 개발자는 JMS 제공자가 메시지 특성의 데이터보다 메시지 본문의 데이터를 더욱 효율적으로 핸들링할 수 있다는 점을 알아야 합니다. 최상의 성능을 위해 애플리케이션은 메시지 헤더를 사용자 정의해야 하는 경우에만 메시지 특성을 사용해야 합니다. 그 이유는 기본적으로 사용자 정의된 메시지 선택을 지원하기 위해서입니다.

JMS 메시지 선택자를 사용하면 클라이언트가 메시지 헤더를 사용하여 관심있는 메시지를 지정할 수 있습니다. 선택자와 일치하는 헤더가 있는 메시지만 전달됩니다.

메시지 선택자는 메시지 본문 값을 참조할 수 없습니다.

메시지 헤더 필드와 특성 값이 선택자의 해당 ID를 대체할 때 선택자가 `true`로 평가되면 메시지 선택자가 메시지와 일치합니다.

메시지 선택자는 SQL92 조건식 구문의 서브세트를 기반으로 하는 구문을 사용하는 문자열입니다. 메시지 선택자가 평가되는 순서는 우선순위 레벨 내의 왼쪽에서 오른쪽으로 평가됩니다. 괄호를 사용하여 이 순서를 변경할 수 있습니다. 사전 정의된 선택자 리터럴 및 연산자 이름은 여기에 대문자로 작성되지만 대소문자를 구분하지는 않습니다.

메시지 선택자의 콘텐츠

메시지 선택자는 다음을 포함할 수 있습니다.

- 리터럴
 - 문자열 리터럴은 따옴표로 묶습니다. 큰따옴표 표시는 따옴표를 나타냅니다. 예를 들어 'literal' 및 'literal's'입니다. Java 문자열 리터럴처럼 유니코드 문자 인코딩을 사용합니다.
 - 정확한 숫자 리터럴은 소수점을 사용하지 않는 숫자 값입니다(예: 57, -957 및 +62). Java long 범위의 숫자가 지원됩니다.
 - 근사 숫자 리터럴은 7E3 또는 -57.9E2와 같은 과학적 표기법의 숫자 값이거나, 7., -95.7 또는 +6.2와 같은 소수점을 포함한 숫자 값입니다. Java double 범위의 숫자가 지원됩니다.
 - Boolean 리터럴 TRUE 및 FALSE.
- ID:
 - ID는 일련의 Java 문자와 Java 숫자로 구성되며 길이가 무제한입니다. 첫 번째는 Java 문자여야 합니다. 문자는 `Character.isJavaLetter` 메소드에서 `true`를 리턴하는 문자입니다. 여기에는 _ 및 \$가 포함됩니다. 문자 또는 숫자는 `Character.isJavaLetterOrDigit` 메소드가 `true`를 리턴하는 문자입니다.
 - ID의 이름은 NULL, TRUE 또는 FALSE가 될 수 없습니다.
 - ID는 NOT, AND, OR, BETWEEN, LIKE, IN 또는 IS가 될 수 없습니다.
 - ID는 헤더 필드 참조 또는 특성 참조입니다.
 - ID는 대소문자를 구분합니다.
 - 메시지 헤더 필드 참조는 다음으로 제한됩니다.
 - JMSDeliveryMode
 - JMSPriority
 - JMSMessageID
 - JMSTimestamp
 - JMSCorrelationID
 - JMSType
 -  JMS_IBM_MsgToken

JMSMessageID, JMSTimestamp, JMSCorrelationID 및 JMSType 값은 Null일 수 있으며, 이 경우 NULL 값으로 처리됩니다.

- JMSX로 시작하는 이름은 JMS 정의 특성 이름입니다.
- JMS_로 시작하는 이름은 제공자 특정 특성 이름입니다.
- JMS로 시작하지 않는 모든 이름은 애플리케이션 특정 특성 이름입니다. 메시지에 없는 특성에 대한 참조가 있을 경우 값은 NULL입니다. 특성이 있으면 특성의 값은 해당 특성 값이 됩니다.
- 공백은 Java에 정의된 것과 동일합니다. 즉, 공백, 가로 탭, 용지 넘김 및 행 종단자입니다.
- 표현식:
 - 선택자는 조건식입니다. true로 평가되는 선택자는 일치하며, false나 알 수 없음으로 평가되는 선택자는 일치하지 않습니다.
 - 산술 연산식은 산술 연산식, 산술 연산, ID(숫자 리터럴로 처리된 값) 및 숫자 리터럴로 구성됩니다.
 - 조건식은 조건식, 비교 연산 및 논리 연산으로 구성됩니다.
- 표현식을 평가하는 순서를 설정하는 표준 대괄호()가 지원됩니다.
- 우선순위의 논리 연산자: NOT, AND, OR.
- 비교 연산자: =, >, >=, <, <=, <>(같지 않음).
 - 동일한 유형의 값만 비교할 수 있습니다. 한 가지 예외는 정확한 숫자 값과 근사 숫자 값을 비교하는 것이 유효합니다 (필수 유형 변환은 Java 숫자 승격 규칙에 따라 정의됨). 다른 유형을 비교하려고 시도할 경우 선택자는 항상 false입니다.
 - 문자열과 부울 비교는 = 및 <>로 제한됩니다. 두 개의 문자열은 동일한 문자 시퀀스를 포함한 경우에만 동일합니다.
- 우선순위 순서의 산술 연산자:
 - +, - 단항.
 - *, /, 곱셈 및 나눗셈.
 - +, -, 덧셈 및 뺄셈.
 - NULL 값의 산술 연산은 지원되지 않습니다. 이 연산을 시도하는 경우 전체 선택자는 항상 false입니다.
 - 산술 연산에서는 Java 숫자 승격을 사용해야 합니다.
- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 및 arithmetic-expr3 비교 연산자:
 - Age BETWEEN 15 and 19는 age >= 15 AND age <= 19와 동일합니다.
 - Age NOT BETWEEN 15 and 19는 age < 15 OR age > 19와 동일합니다.
 - BETWEEN 연산의 표현식이 NULL이면 연산 값은 false입니다. NOT BETWEEN 연산의 표현식이 NULL이면 연산 값은 true입니다.
- 식별자 [NOT] IN (string-literal1, string-literal2,...) 비교 연산자. 여기서 ID의 값은 문자열 또는 NULL입니다.
 - Country IN ('UK', 'US', 'France')은 'UK'의 경우 true이고 'Peru'의 경우 false입니다. 표현식 (Country = 'UK') OR (Country = 'US') OR (Country = 'France')와 같습니다.
 - Country NOT IN ('UK', 'US', 'France')은 'UK'의 경우 false이고 'Peru'의 경우 true입니다. NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France')) 표현식과 같습니다.
 - IN 또는 NOT IN 연산의 ID가 NULL이면 연산 값은 알 수 없음입니다.
- ID [NOT] LIKE pattern-value [ESCAPE escape-character] 비교 연산자. 여기서 ID의 값은 문자열입니다. pattern-value는 문자열 리터럴입니다. 여기서 _은 단일 문자를 나타내며 %는 일련의 문자(빈 시퀀스 포함)를 나타냅니다. 기타 모든 문자는 해당 문자 자체를 나타냅니다. 선택적 escape-character는 pattern-value에 있는 _ 및 %의 특수 의미를 이스케이프하는 데 사용하는 문자가 포함된 단일 문자열 리터럴입니다.
 - phone LIKE '12%3'은 123과 12993은 true이고 1234는 false입니다.
 - word LIKE 'l_se'는 "lose"는 true이고 "loose"는 false입니다.
 - underscored LIKE '_%' ESCAPE '\'은 "_foo"는 true이고 "bar"는 false입니다.
 - phone NOT LIKE '12%3'은 123과 12993은 false이고 1234는 true입니다.
 - LIKE 또는 NOT LIKE 연산의 ID가 NULL이면 연산 값은 알 수 없음입니다.

z/OS

공유 큐의 경우 적절한 z/OS인덱스 설정이 필요합니다. 이러한 방식으로 접근되는 큐 관리자의 공유 z/OS 큐에는 따라서 적절한 MSGID **INDXTYPE** 또는 CORRELID 가 설정되어야 합니다. 개인 큐의 경우 인덱스를 사용하면 성능이 향상되지만 필수 사항은 아닙니다.

MQMO_MATCH_MSG_ID 및 MQMO_MATCH_CORREL_ID **MatchOptions**에 대한 자세한 정보는 [MatchOptions\(MQLONG\)](#)를 참조하십시오.

JMS_IBM_MsgToken 를 사용한 선택

V1000 ~ V1000

JMSMessageID 속성만으로도 일반적으로 특정 메시지를 선택하는 데 충분하지만, 애플리케이션에서 고유한 MessageID 를 설정하지 않은 경우 여러 메시지가 해당 선택기와 일치할 수 있습니다. IBM MQ MsgToken, 를 사용하여 메시지를 정밀하게 선택하는 기능을 기본적으로 지원하며, 이를 JMS 선택기에 활용할 수 있습니다 (MsgToken, 에 대한 자세한 내용은 [MsgToken \(MQBYTE16\)](#) 을 참조하십시오).

JMS_IBM_MsgToken 읽기 전용 메시지 속성이며, 애플리케이션이 큐나 토픽에서 메시지를 검색하거나 처리할 때 초기화됩니다. 애플리케이션은 JMSMessage의 `getStringProperty` 메서드를 호출하여 토큰을 가져온 다음, 이를 셀렉터에서 사용할 수 있습니다. 경우에 따라 빈 문자열이 반환될 수 있습니다. JMS_IBM_MsgToken 선택자 내에서 단독으로만 사용할 수 있으며, 다른 속성과 함께 사용할 수 없습니다.

JMS_IBM_MsgToken 를 사용하여 메시지를 검색할 때, JMS용 MQ 클래스는 MQMO_MATCH_MSG_TOKEN(MatchOption)을 사용하여 메시지를 가져옵니다. 자세한 내용은 [MQMO_*\(일치 옵션\)](#)을 참조하십시오.

큐를 검색하여 `JMS\_IBM\_MsgToken,` 를 가져온 후, 해당 토큰을 사용하여 메시지를 선택하려고 하면 메시지가 반환되지 않을 수 있습니다. 이는 해당 메시지가 이미 다른 애플리케이션에 의해 처리되었기 때문입니다.

예를 들어, 애플리케이션은 JMSContext 또는 Session에서 `createBrowser()` 메서드를 호출하여 큐 목적지에 대한 `QueueBrowser` 객체를 생성한 다음, 이를 사용하여 해당 목적지의 메시지를 조회할 수 있습니다. 처리할 메시지를 발견하면, 해당 애플리케이션은:

- 메시지에서 메시지 토큰을 추출합니다.
- JMS_IBM_MsgToken='message_token' 형식의 메시지 선택기를 생성합니다.
- 메시지 선택기를 지정하여 해당 큐에 대한 컨슈머를 생성합니다.
- Consumer를 사용하여 메시지를 가져옵니다.

다음 코드는 해당 메커니즘을 설명합니다.

```
.....
// Create the QueueBrowser and browse the queue
QueueBrowser browser = context.createBrowser(queue);
Enumeration<?> browsedMessages = browser.getEnumeration();

while (browsedMessages.hasMoreElements()) {
    Message msg = (Message) browsedMessages.nextElement();

    // Extract the message token from the message
    String msgToken = msg.getStringProperty(JmsConstants.JMS_IBM_MSGTOKEN);

    // Build the message selector
    String selector = JmsConstants.JMS_IBM_MSGTOKEN+"='"+msgToken+"'";

    // Create a consumer, passing in the message selector.
    JMSConsumer consumer = context.createConsumer(queue, selector);

    // Get the message from the queue
    Message message = consumer.receiveNoWait();

    .....
}
```

참고: IBM MQMsgToken 선택자 구문은 지원되는 모든 버전의 에서 사용할 수 있습니다.

JMS_IBM_MsgToken 제한 사항

- 형식: 16진수 소문자 32자로 구성되어야 하며, 이는 `message.getStringProperty("JMS\_IBM\_MsgToken")` 호출 시 반환되는 값과 동일해야 합니다.

- 선택자 조합: 다른 선택자 속성과 함께 사용할 수 없습니다.
- 작동 모드: 반드시 즉시 검색(immediate retrieval)만 사용해야 합니다(예: receiveNoWait()).
- 속성 접근 권한: 읽기 전용 속성이므로 애플리케이션에서 설정할 수 없습니다.

제한사항

SQL에서 고정 10진수 비교 및 산술을 지원하지만 JMS 메시지 선택자는 지원하지 않습니다. 정확한 숫자 리터럴이 소수점이 없는 리터럴로 제한되기 때문입니다. 또한 근사 숫자 값의 대체 표시로 10진수가 있는 숫자를 사용하는 이유이기도 합니다.

SQL 주석은 지원되지 않습니다.

JMS 메시지를 IBM MQ 메시지로 맵핑

IBM MQ 메시지는 메시지 디스크립터, 선택적 MQRFH2 헤더 및 본문으로 구성됩니다. JMS 메시지의 콘텐츠는 IBM MQ 메시지에 일부는 맵핑되고 일부는 복사됩니다.

이 주제에서는 이 섹션의 첫 부분에 설명된 JMS 메시지 구조가 IBM MQ 메시지에 맵핑되는 방식을 설명합니다. JMS와 기존 IBM MQ 애플리케이션 사이에 메시지를 전송하려는 프로그래머에게 흥미로운 내용입니다. 예를 들어, IBM Integration Bus 구현에서 두 JMS 애플리케이션 간에 전송되는 메시지를 조작하려는 사용자에게도 중요합니다.

애플리케이션이 브로커에 대한 실시간 연결을 사용하는 경우 이 절은 적용되지 않습니다. 애플리케이션에서 실시간 연결을 사용할 때 모든 통신은 TCP/IP를 통해 직접 수행되며, IBM MQ 큐나 메시지는 관련되지 않습니다.

IBM MQ 메시지는 다음 세 개의 컴포넌트로 구성됩니다.

- MQMD(IBM MQ Message Descriptor)
- IBM MQ MQRFH2 헤더
- 메시지 본문.

MQRFH2는 선택사항이며, 송신 메시지에 포함되는 항목은 JMS Destination 클래스의 TARGCLIENT 플래그에 의해 제어됩니다. IBM MQ JMS 관리 도구를 사용하여 이 플래그를 설정할 수 있습니다. MQRFH2에는 JMS 특정 정보가 있으므로, 수신 목적지가 JMS 애플리케이션임을 송신자가 아는 경우 MQRFH2를 항상 메시지에 포함하십시오. 일반적으로 비JMS 애플리케이션에 메시지를 직접 보낼 때는 MQRFH2를 생략하십시오. 왜냐하면 애플리케이션에서는 IBM MQ 메시지에 MQRFH2가 있을 것으로 기대하지 않기 때문입니다.

수신되는 메시지에 MQRFH2 헤더가 없으면 메시지의 JMSReplyTo 헤더 필드에서 도출된 큐 또는 토픽 오브젝트에는 기본적으로 이 플래그가 설정되므로, 큐나 토픽에 전송된 응답 메시지에 MQRFH2 헤더가 없습니다. 원래 메시지에 MQRFH2 헤더가 있는 경우에만 연결 팩토리의 TARGCLIENTMATCHING 특성을 NO로 설정하여 응답 메시지에 MQRFH2 헤더를 포함하는 이 작동을 오프 상태로 전환할 수 있습니다.

145 페이지의 그림 10에서는 JMS 메시지 구조가 IBM MQ 메시지로 변환되고 다시 원래대로 변환되는 방식을 보여줍니다.

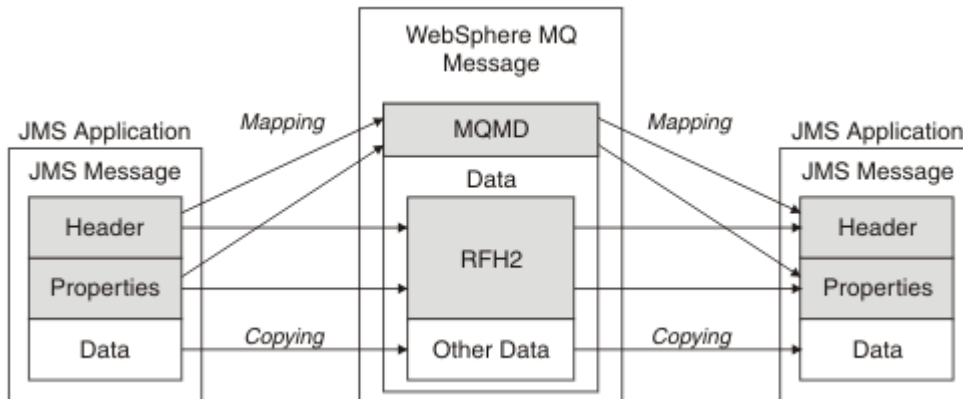


그림 10. MQRFH2 헤더를 사용하여 JMS와 IBM MQ 간에 메시지가 변환되는 방식

구조는 다음 두 가지 방법으로 변환됩니다.

맵핑

MQMD에 JMS 필드와 동등한 필드가 포함되면 JMS 필드가 MQMD 필드에 맵핑됩니다. 비JMS 애플리케이션과 통신할 때 JMS 애플리케이션에서 이러한 필드를 가져오거나 설정해야 할 수도 있으므로 추가 MQMD 필드는 JMS 특성으로 공개됩니다.

복사

MQMD 대응 필드가 없으면 JMS 헤더나 특성이 MQRFH2 내부의 필드로 변환되어 전달됩니다.

MQRFH2 헤더 및 JMS

이 주제 컬렉션에서는 메시지 콘텐츠와 연관된 JMS별 데이터를 수반하는 MQRFH 버전 2 헤더를 설명합니다. MQRFH 버전 2 헤더는 확장 가능하며, JMS와 직접 연결되지 않은 추가 정보도 수반할 수 있습니다. 그러나 이 섹션에서는 JMS에서의 사용만 다룹니다. 전체 설명은 [MQRFH2 - 규칙 및 헤더 2 형식화](#)를 참조하십시오.

헤더는 두 부분, 고정 부분 및 가변 부분으로 구성되어 있습니다.

고정 부분

고정 부분은 표준 IBM MQ 헤더 패턴에 따라 모델링되었으며 다음 필드로 구성됩니다.

StrucId (MQCHAR4)

구조 ID입니다.

MQRFH_STRUC_ID(값: "RFH ")(초기값)여야 합니다.

MQRFH_STRUC_ID_ARRAY(값: "R", "F", "H", " ")도 정의됩니다.

Version(MQLONG)

구조 버전 번호입니다.

MQRFH_VERSION_2(값: 2)(초기값)여야 합니다.

StrucLength(MQLONG)

NameValueData 필드를 포함하여 MQRFH2의 총 길이입니다.

StrucLength로 설정된 값은 4의 배수여야 합니다(이를 위해 NameValueData 필드의 데이터는 공백 문자로 채움).

Encoding(MQLONG)

데이터 인코딩입니다.

MQRFH2 다음에 오는 메시지 부분에 있는 숫자 데이터의 인코딩입니다(다음 헤더 또는 이 헤더 다음에 오는 메시지 데이터).

CodedCharSetId(MQLONG)

코드화 문자 세트 ID.

MQRFH2 다음에 오는 메시지 부분에 있는 문자 데이터 표시입니다(다음 헤더 또는 이 헤더 다음에 오는 메시지 데이터).

Format(MQCHAR8)

형식 이름입니다.

MQRFH2 다음에 오는 메시지 부분의 형식 이름입니다.

플래그 (MQLONG)

플래그입니다.

MQRFH_NO_FLAGS = 0. 플래그가 설정되지 않습니다.

NameValueCCSID(MQLONG)

이 헤더에 포함된 NameValueData 문자열의 코드화 문자 세트 ID(CCSID)입니다. NameValueData는 헤더 (StrucID 및 Format)에 포함된 다른 문자열과 다른 문자 세트로 코드화할 수 있습니다.

NameValueCCSID가 2바이트 유니코드 CCSID(1200, 13488 또는 17584)이면 유니코드의 바이트 순서가 MQRFH2에 있는 숫자 필드의 바이트 순서와 동일합니다 (예: Version, StrucLength 및 NameValueCCSID 자체).

표 16. NameValueCCSID 필드의 가능한 값	
CCSID	의미
1200	UTF-16, 가장 최근 지원된 유니코드 버전
13488	UTF-16, 유니코드 버전 2.0 서브세트
17584	UTF-16, 유니코드 버전 3.0 서브세트(유로 기호 포함)
1208	UTF-8, 가장 최근 지원된 유니코드 버전

가변 부분

가변 부분은 고정 부분 다음에 옵니다. 가변 부분에는 MQRFH2 폴더의 가변 번호가 포함됩니다. 각 폴더에는 가변 요소 또는 특성 수가 포함됩니다. 폴더는 관련 특성을 그룹화합니다. JMS에서 작성한 MQRFH2에는 다음 폴더가 포함될 수 있습니다.

mcd 폴더

mcd에는 메시지의 형식을 설명하는 특성이 포함되어 있습니다. 예를 들어, 메시지 서비스 도메인 Msd 특성은 JMSTextMessage, JMSBytesMessage, JMSStreamMessage, JMSMapMessage, JMSObjectMessage 또는 널로 JMS 메시지를 식별합니다.

mcd 폴더는 항상 MQRFH2을(를) 포함하는 JMS 메시지에 있습니다.

이 메시지는 항상 IBM Integration Bus에서 전송된 MQRFH2을(를) 포함하는 메시지에 표시됩니다. 이 폴더는 메시지의 도메인, 형식, 유형 및 메시지 세트를 설명합니다.

표 17. mcd 특성 이름, 동의어, 데이터 유형 및 폴더			
특성 동의어	특성 이름	데이터 유형	폴더
	mcd.Msd	string	<mcd><Msd>messageDomain</Msd></mcd>
	mcd.Set	string	<mcd><Set>messageDomain</Set></mcd>
	mcd.Type	string	<mcd><Type>messageDomain</Type></mcd>
	mcd.Fmt	string	<mcd><Fmt>messageDomain</Fmt></mcd>

mcd 폴더에 사용자 고유의 특성을 추가하지 마십시오.

jms 폴더

jms에는 JMS 헤더 필드와 MQMD에서 완전히 표현할 수 없는 JMSX 속성이 포함되어 있습니다..jms 폴더는 항상 JMS MQRFH2에 있습니다.

usr 폴더

usr에는 메시지와 연관된 애플리케이션 정의 JMS 특성이 포함되어 있습니다.usr 폴더는 애플리케이션이 애플리케이션 정의 특성을 설정한 경우에만 표시됩니다.

mqext 폴더

mqext에는 다음 유형의 특성이 포함되어 있습니다.

- WebSphere Application Server에서만 사용되는 특성
- 메시지 지연 전달과 관련된 특성

IBM 정의 특성 또는 사용된 전달 지연 중 적어도 하나가 애플리케이션에 설정된 경우에만 이 폴더가 있습니다.

표 18. <i>mqext</i> 특성 이름, 동의어, 데이터 유형 및 폴더			
특성 동의어	특성 이름	데이터 유형	폴더
JMSArmCorrelator	mqext.Arm	string	<mqext><Arm>armCorrelator</Arm></mqext>
JMSRMCorrelator	mqext.Wrm	string	<mqext><Wrm>wrmCorrelator</Wrm></mqext>
JMSDeliveryTime	mqext.Dlt	i8	<mqext><Dlt>DeliveryTime</Dlt></mqext>
JMSDeliveryDelay	mqext.Dly	i8	<mqext><Dly>DeliveryTime</Dly></mqext>

mqext 폴더에 사용자 고유의 특성을 추가하지 마십시오.

mqps 폴더

mqps에는 IBM MQ 발행/구독에서만 사용되는 특성이 포함되어 있습니다. 통합 발행/구독 특성 중 적어도 하나가 애플리케이션에 설정된 경우에만 이 폴더가 있습니다.

표 19. <i>mqps</i> 특성 이름, 동의어, 데이터 유형 및 폴더			
특성 동의어	특성 이름	데이터 유형	폴더
MQTopicString	mqps.Top	string	<mqps><Top>topicString</Top></mqps>
MQSubUserData	mqps.Sud	string	<mqps><Sud>subscriberUserData...</Sud></mqps>
MQIsRetained	mqps.Ret	boolean	<mqps><Ret>isRetained</Ret></mqps>
MQPubOptions	mqps.Pub	i8	<mqps><Pub>publicationOptions</Pub></mqps>
MQPubLevel	mqps.Pbl	i8	<mqps><Pbl>publicationLevel</Pbl></mqps>
MQPubTime	mqpse.Pts	string	<mqps><Pts>publicationTime</Pts></mqps>
MQPubSeqNum	mqpse.Seq	i8	<mqps><Seq>publicationSequenceNumber</Seq></mqps>
MQPubStrIntData	mqpse.Sid	string	<mqps><Sid>publicationData</Sid></mqps>
MQPubFormat	mqpse.Pfmt	i8	<mqps><Pfmt>messageFormat</Pfmt></mqps>

mqps 폴더에 사용자 고유의 특성을 추가하지 마십시오.

148 페이지의 표 20에서는 특성 이름의 전체 목록을 표시합니다.

표 20. JMS에서 사용한 <i>MQRFH2</i> 폴더 및 특성				
JMS 필드 이름	Java 유형	MQRFH2 폴더 이름	특성 이름	유형/값
JMSDestination	목적지	jms	Dst	문자열

표 20. JMS에서 사용한 MQRFH2 폴더 및 특성 (계속)

JMS 필드 이름	Java 유형	MQRFH2 폴더 이름	특성 이름	유형/값
JMSExpiration	long	jms	Exp	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	문자열	jms	Cid	문자열
JMSReplyTo	목적지	jms	Rto	문자열
JMSTimestamp	long	jms	Tms	i8
JMSType	문자열	mcd	Type, Set, Fmt	문자열
JMSXGroupID	문자열	jms	Gid	문자열
JMSXGroupSeq	int	jms	순서	i4
xxx(사용자 정의)	임의	usr	xxx	any
		mcd	Msd	jms_none jms_text jms_bytes jms_map jms_stream jms_object

NameValueLength(MQLONG)

이 길이 필드 바로 뒤에 오는 NameValueData 문자열의 바이트 길이(자신의 길이는 포함하지 않음)입니다.

NameValueData(MQCHARn)

이전 NameValueLength 필드에 길이(바이트)가 제공되는 단일 문자열입니다. 특성의 순서를 보유하는 폴더가 포함되어 있습니다. 각 특성은 다음과 같이 이름/유형/값 삼중항이며 이름이 폴더 이름인 XML 요소에 포함되어 있습니다.

```
<foldername>
triplet1 triplet2 ..... tripletn </foldername>
```

닫기 </foldername> 태그 뒤에는 채움 문자로 공백이 올 수 있습니다. 각 삼중항은 XML과 같은 구문을 사용하여 인코딩됩니다.

```
<name dt='datatype'>value</name>
```

데이터 유형은 사전 정의되므로 dt='datatype' 요소는 선택적이며 많은 특성에서 생략됩니다. 포함된 경우 dt= 태그 앞에 하나 이상의 공백 문자가 포함되어야 합니다.

name

특성의 이름입니다. [148 페이지의 표 20](#)의 내용을 참조하십시오.

datatype

중첩 후에 [150 페이지의 표 21](#)에 나열된 데이터 유형 중 하나와 일치해야 합니다.

value

[150 페이지의 표 21](#)의 정의를 사용하여 변환할 값의 문자열 표시입니다.

널값은 다음 구문을 사용하여 인코딩됩니다.

```
<name dt='datatype' xsi:nil='true'></name>
```

xsi:nil='false'을(를) 사용하지 마십시오.

표 21. 특성 데이터 유형	
데이터 유형	정의
문자열	< 및 &을(를) 제외한 문자 시퀀스
부울	문자 0 또는 1(0 = false, 1 = true)
bin.hex	옥텟을 표시하는 16진수
i1	0..9 숫자를 사용하여 표시되며 선택적으로 부호가 있는 숫자(분수 또는 지수가 아님). -128 ~ 127 범위(포함)에 속해야 합니다.
i2	0..9 숫자를 사용하여 표시되며 선택적으로 부호가 있는 숫자(분수 또는 지수가 아님). -32768 ~ 32767 범위(포함)에 속해야 합니다.
i4	0..9 숫자를 사용하여 표시되며 선택적으로 부호가 있는 숫자(분수 또는 지수가 아님). -2147483648 ~ 2147483647 범위(포함)에 속해야 합니다.
i8	0..9 숫자를 사용하여 표시되며 선택적으로 부호가 있는 숫자(분수 또는 지수가 아님). -9223372036854775808 ~ 9223372036854775807 범위(포함)에 속해야 합니다.
int	0..9 숫자를 사용하여 표시되며 선택적으로 부호가 있는 숫자(분수 또는 지수가 아님). i8과 동일한 범위에 속해야 합니다. 송신자가 특정 정밀도를 특성과 연관시키지 않으려는 경우 i* 유형 중 하나 대신 사용할 수 있습니다.
r4	0..9 숫자, 선택적 부호, 선택적 분수 숫자, 선택적 지수를 사용하여 표시되는 부동 소수점 숫자, 크기 <= 3.40282347E+38, >= 1.175E-37
r8	0..9 숫자, 선택적 부호, 선택적 분수 숫자, 선택적 지수를 사용하여 표시되는 부동 소수점 숫자, 크기 <= 1.7976931348623E+308, >= 2.225E-307

문자열 값에는 공백이 포함될 수 있습니다. 문자열 값에서 다음 이스케이프 순서를 사용해야 합니다.

- & 문자의 경우 &
- < 문자의 경우 <

다음 이스케이프 순서는 사용할 수 있지만 필수는 아닙니다.

- > 문자의 경우 >
- ' 문자의 경우 '
- " 문자의 경우 "

JMS 필드와 대응하는 MQMD 필드가 있는 특성

이러한 표에는 JMS 헤더 필드에 해당하는 MQMD 필드, JMS 특성 및 JMS 제공자 특정 특성이 표시됩니다.

150 페이지의 표 22에는 JMS 헤더 필드가 나열되고 151 페이지의 표 23에는 MQMD 필드에 직접 맵핑되는 JMS 특성이 나열됩니다. 151 페이지의 표 24에는 제공자 특정 특성과 해당 특성이 맵핑된 MQMD 필드가 나열됩니다.

표 22. JMS 헤더 필드와 MQMD 필드 맵핑			
JMS 헤더 필드	Java 유형	MQMD 필드	C 유형
JMSDeliveryMode	int	Persistence	MQLONG
JMSExpiration	long	만기	MQLONG
JMSPriority	int	Priority	MQLONG
JMSMessageID	문자열	MsgID	MQBYTE24

표 22. JMS 헤더 필드와 MQMD 필드 매핑 (계속)			
JMS 헤더 필드	Java 유형	MQMD 필드	C 유형
JMSTimestamp	long	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	문자열	CorrelId	MQBYTE24

표 23. JMS 특성과 MQMD 필드 매핑			
JMS 특성	Java 유형	MQMD 필드	C 유형
JMSXUserID	문자열	UserIdentifier	MQCHAR12
JMSXAppID	문자열	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MLONG
JMSXGroupID	문자열	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MLONG

표 24. JMS 제공자별 특성과 MQMD 필드 매핑			
JMS 제공업체별 특성	Java 유형	MQMD 필드	C 유형
JMS_IBM_Report_Exception	int	보고서	MLONG
JMS_IBM_Report_Expiration	int	보고서	MLONG
JMS_IBM_Report_COA	int	보고서	MLONG
JMS_IBM_Report_COD	int	보고서	MLONG
JMS_IBM_Report_PAN	int	보고서	MLONG
JMS_IBM_Report_NAN	int	보고서	MLONG
JMS_IBM_Report_Pass_Msg_ID	int	보고서	MLONG
JMS_IBM_Report_Pass_Correl_ID	int	보고서	MLONG
JMS_IBM_Report_Discard_Msg	int	보고서	MLONG
JMS_IBM_MsgType	int	MsgType	MLONG
JMS_IBM_Feedback	int	Feedback	MLONG
JMS_IBM_Format	문자열	Format 152 페이지의 『1』	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MLONG
JMS_IBM_Encoding	int	Encoding	MLONG
JMS_IBM_Character_Set	문자열	CodedCharacterSetId 152 페이지의 『2』	MLONG
JMS_IBM_PutDate	문자열	PutDate	MQCHAR8
JMS_IBM_PutTime	문자열	PutTime	MQCHAR8
JMS_IBM_Last_Msg_In_Group	부울	MsgFlags	MLONG

참고:

1. JMS_IBM_Format은 메시지 본문의 형식을 나타냅니다. 이는 메시지의 JMS_IBM_Format 특성을 설정하는 애플리케이션에 의해 정의되거나 (8자제한이 있음) JMS 메시지 유형에 적합한 메시지 본문의 IBM MQ 형식으로 기본 설정될 수 있습니다. 메시지에 RFH나 RFH2 섹션이 없는 경우에만 JMS_IBM_Format이 MQMD 형식 필드에 매핑됩니다. 일반적인 메시지에서는 메시지 본문 바로 앞에 있는 RFH2의 형식 필드에 매핑됩니다.
2. JMS_IBM_Character_Set 특성 값은 숫자 CodedCharacterSetId 값에 대해 동등한 Java 문자 세트가 포함된 문자열 값입니다. MQMD 필드 CodedCharacterSetId는 JMS_IBM_Character_Set 특성에 지정된 Java 문자 세트 문자열에 해당하는 값을 포함하는 숫자 값입니다.

JMS fields onto IBM MQ 필드 매핑(보내는 메시지)

이 표에는 JMS 헤더와 특성 필드가 send() 또는 publish() 중에 MQMD와 MQRFH2 필드에 매핑되는 방식을 보여줍니다.

152 페이지의 표 25에서는 JMS 헤더가 send() 또는 publish() 중에 MQMD/RFH2 필드에 매핑되는 방식을 보여줍니다. 152 페이지의 표 26에서는 JMS 특성이 send() 또는 publish() 중에 MQMD/RFH2 필드에 매핑되는 방식을 보여줍니다. 153 페이지의 표 27에서는 JMS 제공자 특정 특성이 send() 또는 publish() 중에 MQMD 필드에 매핑되는 방식을 보여줍니다.

메시지 오브젝트를 통해 설정으로 표시된 필드에서 전송된 값은 send() 또는 publish() 조작 바로 전에 JMS 메시지에 보류된 값입니다. JMS 메시지 값은 조작이 변경하지 않은 상태로 둡니다.

보내기 메소드를 통해 설정으로 표시된 필드에서 값은 send() 또는 publish()를 수행할 때 지정됩니다(JMS 메시지에 보류된 값은 무시됨). 사용된 값을 표시하기 위해 JMS 메시지의 값이 업데이트됩니다.

수신 전용으로 표시된 필드는 전송되지 않으며 send() 또는 publish()를 통해 메시지에서 변경되지 않습니다.

표 25. 보내는 메시지 필드 매핑			
JMS 헤더 필드 이름	전송에 사용되는 MQMD 필드	헤더	설정 기준
JMSDestination		MQRFH2	송신 메소드
JMSDeliveryMode	Persistence	MQRFH2	송신 메소드
JMSExpiration	만기	MQRFH2	송신 메소드
JMSPriority	Priority	MQRFH2	송신 메소드
JMSMessageID	MsgID		송신 메소드
JMSTimestamp	PutDate/PutTime		송신 메소드
JMSCorrelationID	CorrelId	MQRFH2	메시지 오브젝트
JMSReplyTo	ReplyToQ/ReplyToQMgr	MQRFH2	메시지 오브젝트
JMSType		MQRFH2	메시지 오브젝트
JMSRedelivered			수신 전용

참고:

1. MQMD 필드 CodedCharacterSetId는 JMS_IBM_Character_Set 특성에 지정된 Java 문자 세트 문자열에 해당하는 값을 포함하는 숫자 값입니다.

표 26. 보내는 메시지 JMS 특성 매핑			
JMS 특성 이름	전송에 사용되는 MQMD 필드	헤더	설정 기준
JMSXUserID	UserIdentifier		송신 메소드
JMSXAppID	PutApplName		송신 메소드
JMSXDeliveryCount			수신 전용
JMSXGroupID	GroupId	MQRFH2	메시지 오브젝트

표 26. 보내는 메시지 JMS 특성 �핑 (계속)			
JMS 특성 이름	전송에 사용되는 MQMD 필드	헤더	설정 기준
JMSXGroupSeq	MsgSeqNumber	MQRFH2	메시지 오브젝트

참고:

이러한 특성은 JMS 스펙에 의해 읽기 전용으로 정의되며, JMS 제공자에 의해 설정(어떤 경우에는 선택적으로) 됩니다.

IBM MQ classes for JMS에서는 애플리케이션이 이들 중 두 특성을 대체할 수 있습니다. 이를 수행하려면 목적이 다음 특성을 설정하여 적절하게 구성되었는지 확인하십시오.

1. 특성 WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT를 WMQConstants.WMQ_MDCTX_SET_ALL_CONTEXT로 설정하십시오.
2. 특성 WMQConstants.WMQ_MQMD_WRITE_ENABLED를 true로 설정하십시오.

애플리케이션이 대체할 수 있는 특성은 다음과 같습니다.

JMSXAppID

이 속성은 메시지에 WMQConstants.JMS_IBM_MQMD_PUTAPPLNAME 속성을 설정하여 재정의할 수 있습니다. 값은 Java 문자열이어야 합니다.

JMSXGroupID

이 특성은 메시지에 특성 WMQConstants.JMS_IBM_MQMD_GROUPID를 설정하여 대체할 수 있으며, 값은 바이트 배열이어야 합니다.

표 27. 보내는 메시지 JMS 제공자 특정 특성 �핑			
JMS 제공자 특정 특성 이름	전송에 사용되는 MQMD 필드	헤더	설정 기준
JMS_IBM_Report_Exception	보고서		메시지 오브젝트
JMS_IBM_Report_Expiration	보고서		메시지 오브젝트
JMS_IBM_Report_COA/COD	보고서		메시지 오브젝트
JMS_IBM_Report_NAN/PAN	보고서		메시지 오브젝트
JMS_IBM_Report_Pass_Msg_ID	보고서		메시지 오브젝트
JMS_IBM_Report_Pass_Correl_ID	보고서		메시지 오브젝트
JMS_IBM_Report_Discard_Msg	보고서		메시지 오브젝트
JMS_IBM_MsgType	MsgType		메시지 오브젝트
JMS_IBM_Feedback	Feedback		메시지 오브젝트
JMS_IBM_Format	형식		메시지 오브젝트
JMS_IBM_PutApplType	PutApplType		송신 메소드
JMS_IBM_Encoding	Encoding		메시지 오브젝트
JMS_IBM_Character_Set	CodedCharacterSetId		메시지 오브젝트
JMS_IBM_PutDate	PutDate		송신 메소드
JMS_IBM_PutTime	PutTime		송신 메소드
JMS_IBM_Last_Msg_In_Group	MsgFlags		메시지 오브젝트

send() 또는 *publish()*에서 JMS 헤더 필드 �핑

이 참고사항은 *send()* 또는 *publish()*에서 JMS 필드의 �핑과 관련됩니다.

JMSDestination에서 MQRFH2

수신 JMS이(가) 동등한 목적지 오브젝트를 재구성할 수 있도록 목적지 오브젝트의 핵심 특성을 직렬화하는 문자열로 저장됩니다. MQRFH2 필드는 URI로 인코딩됩니다(URI 표기법 세부사항은 [214 페이지의 『URI\(Uniform Resource Identifier\)』](#) 참조).

JMSReplyTo에서 MQMD.ReplyToQ, ReplyToQMGr, MQRFH2

큐 이름이 MQMD.ReplyToQ 필드에 복사되며, 큐 관리자 이름이 ReplyToQMGr 필드에 복사됩니다. 목적지 확장 정보(목적지 오브젝트에서 유지되는 기타 유용한 세부사항)는 MQRFH2 필드에 복사됩니다. MQRFH2 필드는 URI로서 인코딩됩니다(URI 표기법의 세부사항은 [214 페이지의 『URI\(Uniform Resource Identifier\)』](#) 참조).

JMSDeliveryMode에서 MQMD.Persistence

목적지 오브젝트가 대체하지 않는 한 JMSDeliveryMode 값은 send() 또는 publish() 메소드 또는 MessageProducer에 의해 설정됩니다. JMSDeliveryMode 값은 다음과 같이 MQMD.Persistence 필드에 맵핑됩니다.

- JMS 값 PERSISTENT는 MQPER_PERSISTENT와 동등함
- JMS 값 NON_PERSISTENT는 MQPER_NOT_PERSISTENT와 동등함

MQQueue 지속성 특성이 WMQConstants.WMQ_PER_QDEF로 설정되지 않은 경우에는 전달 모드 값도 MQRFH2에서 인코딩됩니다.

JMSExpiration 대 MQMD.Expiry, MQRFH2(양방향)

JMSExpiration은 만료 시간(현재 시간 + 잔존 시간)을 저장하는 반면, MQMD는 잔존 시간을 저장합니다. 또한 JMSExpiration은 밀리초 단위이지만, MQMD.Expiry는 1/10초 단위입니다.

- send() 메소드가 무제한 잔존 시간으로 설정된 경우, MQMD.Expiry는 MQEI_UNLIMITED로 설정되며 JMSExpiration은 MQRFH2에서 인코딩되지 않습니다.
- send() 메소드가 214748364.7초(약 7년) 미만인 잔존 시간을 설정하는 경우, 잔존 시간은 MQMD.Expiry에 저장되며 만료 시간(밀리초)은 MQRFH2에서 i8 값으로 인코딩됩니다.
- send() 메소드가 214748364.7초를 초과하는 잔존 시간을 설정하는 경우, MQMD.Expiry는 MQEI_UNLIMITED로 설정됩니다. 실제 만료 시간(밀리초)은 MQRFH2에서 i8 값으로 인코딩됩니다.

JMSPriority에서 MQMD.Priority

JMSPriority 값(0-9)을 MQMD 우선순위 값(0-9)으로 직접 맵핑합니다. JMSPriority가 비기본 값으로 설정된 경우, 우선순위 레벨도 MQRFH2에서 인코딩됩니다.

MQMD.MessageID에서 JMSMessageID

JMS에서 송신된 모든 메시지에는 IBM MQ에서 지정한 고유 메시지 ID가 있습니다. 지정된 값은 MQPUT 호출 이후에 MQMD.MessageId 필드에서 리턴되며, JMSMessageID 필드의 애플리케이션으로 다시 전달됩니다. IBM MQ messageId는 24바이트 2진 값인 반면, JMSMessageID는 문자열입니다. JMSMessageID는 문자 ID가 접두부로 지정된 48개 16진 문자의 시퀀스로 변환된 2진 messageId 값으로 구성되어 있습니다. JMS는 메시지 ID의 프로덕션을 사용 안하도록 설정될 수 있는 힌트를 제공합니다. 이 힌트는 무시되며, 고유 ID가 모든 케이스에서 지정됩니다. send() 이전에 JMSMessageID 필드에 설정된 모든 값은 대체됩니다.

MQMD.MessageID, 을 지정하는 기능이 필요한 경우 [234 페이지의 『IBM MQ classes for JMS 애플리케이션에서 메시지 디스크립터 읽기 및 쓰기』](#)에 설명된 IBM MQ JMS 확장자 중 하나를 사용하여 이 작업을 수행할 수 있습니다.

JMSTimestamp에서 MQRFH2

송신 중에 JMSTimestamp 필드는 JVM의 클럭에 따라 설정됩니다. 이 값은 MQRFH2로 설정됩니다. send() 이전에 JMSTimestamp 필드에 설정된 모든 값은 대체됩니다. JMS_IBM_PutDate 및 JMS_IBM_PutTime 특성도 참조하십시오.

JMSType에서 MQRFH2

이 문자열은 MQRFH2 mcd.Type 필드로 설정됩니다. URI 형식인 경우, 이는 mcd.Set 및 mcd.Fmt 필드에도 영향을 줍니다.

JMSCorrelationID에서 MQMD.CorrelId, MQRFH2

JMSCorrelationID는 다음 중 하나를 보유할 수 있습니다.

제공자 특정 메시지 ID

이는 이전에 송신 또는 수신된 메시지의 메시지 ID이므로 ID:가 접두부로 지정된 48자 소문자의 16진수 문자열이어야 합니다. 접두부가 제거되고 나머지 문자는 2진으로 변환된 후 MQMD.CorrelId 필드에 설정됩니다.

제공자 고유 byte[] 값

값은 MQMD.CorrelId 필드로 복사되며, 널로 채워지거나 24바이트로 잘립니다(필요한 경우). CorrelId 값은 MQRFH2에서 인코딩되지 않습니다.

애플리케이션 특정 문자열

값이 MQRFH2로 설정됩니다. UTF8 형식인 문자열의 첫 24바이트는 MQMD.CorrelID에 쓰여집니다.

JMS 특성 필드 매핑

이 참고사항은 IBM MQ 메시지에서 JMS 특성 필드의 매핑을 참조합니다.

MQMD UserIdentifier에서 JMSXUserID

JMSXUserID가 send 호출에서 리턴 시에 설정됩니다.

MQMD PutApplName에서 JMSXAppID

JSMXAppID가 send 호출에서 리턴 시에 설정됩니다.

JMSXGroupID에서 MQRFH2(포인트-투-포인트)

포인트-투-포인트 메시지의 경우, JMSXGroupID가 MQMD GroupID 필드에 복사됩니다. JMSXGroupID가 접두부 ID:로 시작하는 경우, 이는 2진으로 변환됩니다. 그렇지 않으면, 이는 UTF8 문자열로 인코딩됩니다. 값은 필요하면 24바이트 길이로 채워지거나 잘립니다. MQMF_MSG_IN_GROUP 플래그가 설정됩니다.

JMSXGroupID에서 MQRFH2(발행/구독)

발행/구독 메시지의 경우, JMSXGroupID가 문자열로서 MQRFH2에 복사됩니다.

JMSXGroupSeq MQMD MsgSeqNumber(포인트-투-포인트)

포인트-투-포인트 메시지의 경우, JMSXGroupSeq가 MQMD MsgSeqNumber 필드에 복사됩니다. MQMF_MSG_IN_GROUP 플래그가 설정됩니다.

JMSXGroupSeq MQMD MsgSeqNumber(발행/구독)

발행/구독 메시지의 경우, JMSXGroupSeq가 i4로서 MQRFH2에 복사됩니다.

JMS 제공자 특정 필드 매핑

다음 노트는 공급자별 JMS 필드를 메시지로 IBM MQ 매핑하는 것을 참조합니다.

MQMD 보고서에 대한 JMS_IBM_Report_XXX

JMS 애플리케이션은 다음 JMS_IBM_Report_XXX 특성을 사용하여 MQMD 보고서 옵션을 설정할 수 있습니다. 단일 MQMD는 여러 JMS_IBM_Report_XXX 특성에 매핑됩니다.

JMS_IBM_Report_XXX 상수는 또는 com.ibm.msg.client.jakarta.wmq.WMQConstants 에 있습니다 com.ibm.msg.client.wmq.WMQConstants.

JMS_IBM_Report_Exception

MQRO_EXCEPTION 또는
MQRO_EXCEPTION_WITH_DATA 또는
MQRO_EXCEPTION_WITH_FULL_DATA

JMS_IBM_Report_Expiration

MQRO_EXPIRATION 또는
MQRO_EXPIRATION_WITH_DATA 또는
MQRO_EXPIRATION_WITH_FULL_DATA

JMS_IBM_Report_COA

MQRO_COA 또는
MQRO_COA_WITH_DATA 또는
MQRO_COA_WITH_FULL_DATA

JMS_IBM_Report_COD

MQRO_COD 또는
MQRO_COD_WITH_DATA 또는
MQRO_COD_WITH_FULL_DATA

JMS_IBM_Report_PAN

MQRO_PAN

JMS_IBM_Report_NAN

MQRO_NAN

JMS_IBM_Report_Pass_Msg_ID

MQRO_PASS_MSG_ID

JMS_IBM_Report_Pass_Correl_ID

MQRO_PASS_CORREL_ID

JMS_IBM_Report_Discard_Msg

MQRO_DISCARD_MSG

MQRO 값은 단위입니다 com.ibm.mq.constants.CMQC.

JMS_IBM_MsgType에서 MQMD MessageType

값이 MQMD MessageType으로 직접 매핑됩니다. 애플리케이션이 JMS_IBM_MsgType의 명시적 값을 설정하지 않으면 기본값이 사용됩니다. 이 기본값은 다음과 같이 판별됩니다.

- JMSReplyTo가 IBM MQ 큐 목적지로 설정된 경우, MessageType은 MQMT_REQUEST 값으로 설정됨
- JMSReplyTo가 설정되지 않거나 IBM MQ 큐 목적지 이외의 값으로 설정된 경우, MessageType은 MQMT_DATAGRAM 값으로 설정됨

JMS_IBM_Feedback에서 MQMD 피드백

값이 MQMD 피드백으로 직접 매핑됩니다.

JMS_IBM_Format에서 MQMD 형식

값이 MQMD 형식으로 직접 매핑됩니다.

JMS_IBM_Encoding에서 MQMD 인코딩

설정된 경우, 이 특성은 목적지 큐 또는 토픽의 숫자 인코딩을 대체합니다.

JMS_IBM_Character_Set에서 MQMD CodedCharacterSetId

설정된 경우, 이 특성은 목적지 큐 또는 토픽의 코드화 문자 세트 특성을 대체합니다.

MQMD PutDate에서 JMS_IBM_PutDate

이 특성 값은 송신 중에 MQMD의 PutDate 필드에서 직접 설정됩니다. 송신 전에 JMS_IBM_PutDate 특성으로 설정된 값은 대체됩니다. 이 필드는 YYYYMMDD의 IBM MQ 날짜 형식인 8자 문자열입니다. 이 특성을 JMS_IBM_PutTime 특성과 함께 사용하여 큐 관리자에 따라 메시지를 넣은 시간을 판별할 수 있습니다.

MQMD PutTime에서 JMS_IBM_PutTime

이 특성 값은 송신 중에 MQMD의 PutTime 필드에서 직접 설정됩니다. 송신 전에 JMS_IBM_PutTime 특성으로 설정된 값은 대체됩니다. 이 필드는 HHMMSSSTH의 IBM MQ 시간 형식인 8자 문자열입니다. 이 특성을 JMS_IBM_PutDate 특성과 함께 사용하여 큐 관리자에 따라 메시지를 넣은 시간을 판별할 수 있습니다.

JMS_IBM_Last_Msg_In_Group에서 MQMD MsgFlags

포인트-투-포인트 메시징의 경우, 이 Boolean 값은 MQMD MsgFlags 필드의 MQMF_LAST_MSG_IN_GROUP 플래그에 매핑됩니다. 이는 일반적으로 JMSXGroupID 및 JMSXGroupSeq 특성과 함께 사용되어 이 메시지가 그룹에서 마지막임을 레거시 IBM MQ 애플리케이션에 표시합니다. 이 특성은 발행/구독 메시징의 경우 무시됩니다.

JMS_IBM_MsgToken MQGMO MsgToken

이 속성은 MQGMO(MQ 메시지 옵션 가져오기) 구조체의 'MsgToken' 필드와 대응됩니다. 메시지 토큰은 메시지가 큐에 삽입될 때 MQPUT 호출에 의해 반환되는 고유 식별자입니다. 이 토큰은 이후의 MQGET 작업에서 MQMO_MATCH_MSG_TOKEN 일치 옵션을 지정하여 특정 메시지를 검색하는 데 사용할 수 있습니다.

IBM MQ 필드를 JMS 필드에 매핑(수신 메시지)

다음 표에서는 JMS 헤더 및 특성 필드가 get() 또는 receive() 시간에 MQMD 및 MQRFH2 필드로 매핑되는 방법을 표시합니다.

157 페이지의 표 28에서는 JMS 헤더 필드가 get() 또는 receive() 시에 MQMD/MQRFH2 필드로 매핑되는 방법을 표시합니다. 157 페이지의 표 29에서는 JMS 특성 필드가 get() 또는 receive() 시에 MQMD/MQRFH2 필드로 매핑되는 방법을 표시합니다. 158 페이지의 표 30에서는 JMS 제공자 특정 특성이 매핑되는 방법을 표시합니다.

표 28. 수신 메시지 JMS 헤더 필드 매핑		
JMS 헤더 필드 이름	MQMD 필드가 검색된 소스	MQRFH2 필드가 검색된 소스
JMSDestination		jms.Dst 또는 mqps.Top 157 페이지의 『1』
JMSDeliveryMode	Persistence 157 페이지의 『2』	jms.Dlv 157 페이지의 『2』
JMSExpiration		jms.Exp
JMSPriority	Priority	
JMSMessageID	MsgID	
JMSTimestamp	PutDate 157 페이지의 『2』 PutTime 157 페이지의 『2』	jms.Tms 157 페이지의 『2』
JMSCorrelationID	CorrelId 157 페이지의 『2』	jms.Cid 157 페이지의 『2』
JMSReplyTo	ReplyToQ 157 페이지의 『2』 ReplyToQMGr 157 페이지의 『2』	jms.Rto 157 페이지의 『2』
JMSType		mcd.Type, mcd.Set, mcd.Fmt
JMSRedelivered	BackoutCount	

참고:

1. jms.Dst 및 mqps.Top이 모두 설정된 경우에는 jms.Dst의 값이 사용됩니다.
2. MQRFH2 또는 MQMD에서 검색된 값을 보유할 수 있는 특성의 경우, 둘 모두가 사용 가능하면 MQRFH2의 설정이 사용됩니다.
3. JMS_IBM_Character_Set 특성 값은 숫자 CodedCharacterSetId 값에 대해 동등한 Java 문자 세트가 포함된 문자열 값입니다.

표 29. 수신 메시지 특성 매핑		
JMS 특성 이름	MQMD 필드가 검색된 소스	MQRFH2 필드가 검색된 소스
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId 157 페이지의 『1』	jms.Gid 157 페이지의 『1』
JMSXGroupSeq	MsgSeqNumber 157 페이지의 『1』	jms.Seq 157 페이지의 『1』

참고:

1. MQRFH2 또는 MQMD에서 검색된 값을 보유할 수 있는 특성의 경우, 둘 모두가 사용 가능하면 MQRFH2의 설정이 사용됩니다. MQMF_MSG_IN_GROUP 또는 MQMF_LAST_MSG_IN_GROUP 메시지 플래그가 설정된 경우에만 특성이 MQMD 값에서 설정됩니다.

표 30. 수신 메시지 제공자 특정 JMS 특성 맵핑

JMS 특성 이름	MQMD 필드가 검색된 소스	MQRFH2 필드가 검색된 소스
JMS_IBM_Report_Exception	보고서	
JMS_IBM_Report_Expiration	보고서	
JMS_IBM_Report_COA	보고서	
JMS_IBM_Report_COD	보고서	
JMS_IBM_Report_PAN	보고서	
JMS_IBM_Report_NAN	보고서	
JMS_IBM_Report_Pass_Msg_ID	보고서	
JMS_IBM_Report_Pass_Correl_ID	보고서	
JMS_IBM_Report_Discard_Msg	보고서	
JMS_IBM_MsgType	MsgType	
JMS_IBM_Feedback	Feedback	
JMS_IBM_Format	형식	
JMS_IBM_PutApplType	PutApplType	
JMS_IBM_Encoding 158 페이지의 『1』	Encoding	
JMS_IBM_Character_Set 158 페이지의 『1』	CodedCharacterSetId	
JMS_IBM_PutDate	PutDate	
JMS_IBM_PutTime	PutTime	
JMS_IBM_Last_Msg_In_Group	MsgFlags	

1. 수신 메시지가 바이트 메시지인 경우에만 설정됩니다.

JMS 애플리케이션 및 기존 IBM MQ 애플리케이션 간의 메시지 교환

이 주제에서는 JMS 애플리케이션이 MQRFH2 헤더를 처리할 수 없는 기존 IBM MQ 애플리케이션과 메시지를 교환할 때 발생하는 상황을 설명합니다.

[159 페이지의 그림 11](#)에서는 맵핑을 표시합니다.

관리자는 목적지의 TARGCLIENT 특성을 MQ로 설정하여 JMS 애플리케이션이 기존 IBM MQ 애플리케이션과 통신 중임을 표시합니다. 이는 MQRFH2 헤더가 생성되지 않음을 표시합니다. 이를 수행하지 않은 경우, 수신 애플리케이션은 MQRFH2 헤더를 핸들링할 수 있어야 합니다.

JMS에서 기존 IBM MQ 애플리케이션을 대상으로 한 MQMD로의 맵핑은 JMS에서 JMS 애플리케이션을 대상으로 한 MQMD로의 맵핑과 동일합니다. IBM MQ classes for JMS 가 MQMD *Format* 필드가 MQFMT_RFH2이외의 값으로 설정된 IBM MQ 메시지를 수신하는 경우, 데이터는 비JMS 애플리케이션에서 수신됩니다. 형식이 MQFMT_STRING인 경우에는 메시지가 JMS 텍스트 메시지로 수신됩니다. 그렇지 않으면, 이는 JMS 바이트 메시지로 수신됩니다. MQRFH2가 없으므로, MQMD에서 전송된 해당 JMS 특성만 복원이 가능합니다.

IBM MQ classes for JMS에서 MQRFH2 헤더가 없는 메시지를 수신하는 경우, 메시지의 JMSReplyTo 헤더 필드에서 도출된 큐 및 토픽 오브젝트의 TARGCLIENT 특성이 기본적으로 MQ로 설정됩니다. 이는 큐 또는 토픽에 송신된 응답 메시지에도 MQRFH2 헤더가 없음을 의미합니다. 원래 메시지에 MQRFH2 헤더가 있는 경우에만 연결 팩토리의 TARGCLIENTMATCHING 특성을 NO로 설정하여 응답 메시지에 MQRFH2 헤더를 포함하는 이 작동을 오프 상태로 전환할 수 있습니다.

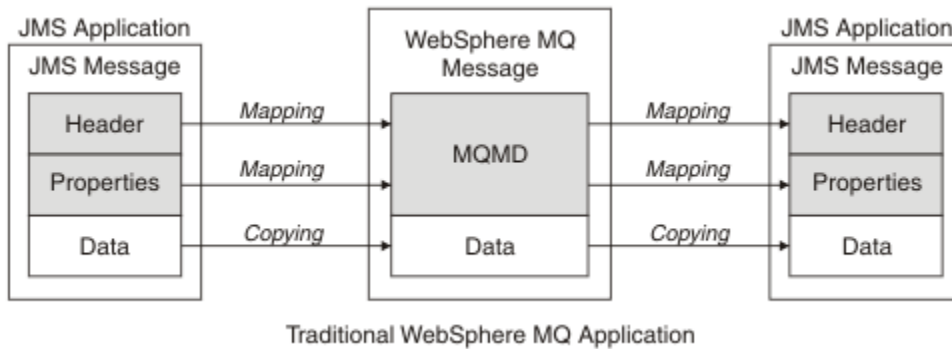


그림 11. JMS 메시지를 MQRFH2 헤더가 없는 IBM MQ 메시지로 변환하는 방법

JMS 메시지 본문

이 주제에는 메시지 본문 자체의 인코딩에 대한 정보가 포함되어 있습니다. 인코딩은 JMS 메시지의 유형에 따라 다릅니다.

ObjectMessage

ObjectMessage는 일반적인 방식으로 Java 런타임에 의해 직렬화됩니다.

TextMessage

TextMessage는 인코딩된 문자열입니다. 발신 메시지의 경우, 문자열은 목적지 오브젝트에서 제공하는 문자 세트로 인코딩됩니다. 이의 기본값은 UTF8 인코딩입니다(UTF8 인코딩은 메시지의 첫 문자로 시작되며, 시작 시에 길이 필드가 없음). 그러나 IBM MQ classes for JMS에서 지원하는 기타 문자 세트를 지정할 수도 있습니다. 해당 문자 세트는 메시지를 비-JMS 애플리케이션으로 송신할 때 주로 사용됩니다.

문자 세트가 2바이트 세트(UTF16 포함)인 경우, 목적지 오브젝트의 정수 인코딩 스펙은 바이트의 순서를 판별합니다.

수신 메시지는 메시지 자체에 지정된 문자 세트 및 인코딩을 사용하여 해석됩니다. 이러한 스펙은 마지막 IBM MQ 헤더(또는 헤더가 없으면 MQMD)에 있습니다. JMS 메시지의 경우, 마지막 헤더는 일반적으로 MQRFH2입니다.

BytesMessage

BytesMessage는 기본적으로 JMS 1.0.2 스펙 및 연관된 Java 문서에서 정의하는 바이트 시퀀스입니다.

애플리케이션 자체에 의해 어셈블된 발신 메시지의 경우에는 목적지 오브젝트의 인코딩 특성을 사용하여 메시지에 포함된 정수 및 부동 소수점 필드의 인코딩을 대체할 수 있습니다. 예를 들어, 부동 소수점 값이 IEEE 형식 대신 S/390으로 저장되도록 요청할 수 있습니다.

수신 메시지는 메시지 자체에 지정된 숫자 인코딩을 사용하여 해석됩니다. 이 스펙은 마지막 IBM MQ 헤더(또는 헤더가 없으면 MQMD)에 있습니다. JMS 메시지의 경우, 마지막 헤더는 일반적으로 MQRFH2입니다.

BytesMessage가 수신되고 수정 없이 재송신되는 경우, 해당 본문은 수신된 그대로 바이트 단위로 전송됩니다. 목적지 오브젝트의 인코딩 특성은 본문에 영향을 주지 않습니다. BytesMessage에서 명시적으로 송신될 수 있는 유일한 문자열 같은 엔티티는 UTF8 문자열입니다. 이는 Java UTF8 형식으로 인코딩되며, 2바이트 길이 필드로 시작됩니다. 목적지 오브젝트의 문자 세트 특성은 발신 BytesMessage의 인코딩에는 영향을 주지 않습니다. 수신 IBM MQ 메시지의 문자 세트 값은 JMS BytesMessage로서 해당 메시지의 해석에 영향을 주지 않습니다.

비-Java 애플리케이션은 Java UTF8 인코딩을 인식할 가능성이 거의 없습니다. 따라서 JMS 애플리케이션이 텍스트 데이터가 포함된 BytesMessage를 송신하려면, 애플리케이션이 직접 해당 문자열을 바이트 배열로 변환해야 하며 이러한 바이트 배열을 BytesMessage에 써야 합니다.

MapMessage

MapMessage는 다음으로 인코딩된 XML 이름/유형/값 쌍이 포함된 문자열입니다.

```
<map>
  <elt name="elementname1" dt="datatype1">value1</elt>
  <elt name="elementname2" dt="datatype2">value2</elt>
  ...
</map>
```

여기서 datatype은 150 페이지의 표 21에 나열된 데이터 유형 중 하나입니다. 기본 데이터 유형이 string이므로, 속성 dt="string"은 문자열 요소에 대해 생략됩니다.

맵 메시지의 본문을 구성하는 XML 문자열을 인코딩하거나 해석하는 데 사용되는 문자 세트는 텍스트 메시지에 적용되는 규칙에 따라 판별됩니다.

5.3 이전의 IBM MQ classes for JMS 버전은 맵 메시지의 본문을 다음 형식으로 인코딩했습니다.

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
  ...
</map>
```

IBM MQ classes for JMS 5.3 이상은 두 형식을 모두 해석할 수 있지만, 5.3 미만의 IBM MQ classes for JMS 버전은 현재 형식을 해석할 수 없습니다.

애플리케이션이 5.3 미만의 IBM MQ classes for JMS 버전을 사용 중인 다른 애플리케이션에 맵 메시지를 송신해야 하는 경우, 송신 애플리케이션은 연결 팩토리 메소드 `setMapNameStyle(WMQConstants.WMQ_MAP_NAME_STYLE_COMPATIBLE)`을 사용하여 맵 메시지가 이전 형식으로 송신됨을 지정해야 합니다. 기본적으로, 모든 맵 메시지는 현재 형식으로 송신됩니다.

StreamMessage

StreamMessage는 맵 메시지와 유사하지만 요소 이름이 없습니다.

```
<stream>
  <elt dt="datatype1">value1</elt>
  <elt dt="datatype2">value2</elt>
  ...
</stream>
```

여기서 datatype은 150 페이지의 표 21에 나열된 데이터 유형 중 하나입니다. 기본 데이터 유형이 string이므로, 속성 dt="string"은 문자열 요소에 대해 생략됩니다.

StreamMessage 본문을 구성하는 XML 문자열을 인코딩하거나 해석하는 데 사용되는 문자 세트는 TextMessage에 적용되는 규칙에 따라 판별됩니다.

MQRFH2.format 필드는 다음과 같이 설정됩니다.

MQFMT_NONE

ObjectMessage, BytesMessage 또는 본문 없는 메시지의 경우.

MQFMT_STRING

TextMessage, StreamMessage 또는 MapMessage의 경우.

JMS 메시지 변환

JMS의 메시지 데이터 변환은 메시지를 송신하고 수신할 때 수행됩니다. IBM MQ는 대부분의 데이터 변환을 자동으로 수행합니다. 이는 JMS 애플리케이션 간에 메시지를 전송할 때 텍스트 및 숫자 데이터를 변환합니다. 텍스트는 JMS 애플리케이션과 IBM MQ 애플리케이션 간에 JMSTextMessage를 교환할 때 변환됩니다.

보다 복잡한 메시지 교환을 수행하려고 계획 중이면 다음 주제를 참조하십시오. 복잡한 메시지 교환에는 다음이 포함됩니다.

- IBM MQ 애플리케이션 및 JMS 애플리케이션 간의 비-텍스트 메시지 전송.
- 바이트 형식으로 텍스트 데이터 교환.
- 애플리케이션에서 텍스트 변환.

JMS 메시지 데이터

데이터 변환은 애플리케이션 간에는 물론 두 JMS 애플리케이션 간에 텍스트 및 숫자 데이터를 교환하는 데 필요합니다. 메시지에서 전송될 수 있도록 텍스트 및 숫자의 내부 표현은 인코딩되어야 합니다. 인코딩은 숫자 및 텍스트가 표현되는 방법에 대한 의사결정을 강제합니다. IBM MQ는 JMSObjectMessage를 제외하고 JMS 메시지에서 텍스트 및 숫자의 인코딩을 관리합니다. 167 페이지의 『JMSObjectMessage』의 내용을 참조하십시오. 이는 세 개의 메시지 속성을 사용합니다. 세 개의 속성은 CodedCharacterSetId, Encoding 및 Format입니다.

이 세 개의 메시지 속성은 일반적으로 JMS 헤더, MQRFH2, JMS 메시지의 필드에 저장되어 있습니다. 메시지 유형이 메시지의 JMS 유형이 아닌 MQ인 경우, 속성은 메시지 디스크립터 MQMD에 저장됩니다. 속성은 JMS 메시지 데이터를 변환하는 데 사용됩니다. JMS 메시지 데이터는 IBM MQ 메시지의 메시지 데이터 파트에서 전송됩니다.

JMS 메시지 특성

JMS 메시지 특성 (예: JMS_IBM_CHARACTER_SET) 은 MQRFH2없이 메시지가 전송되지 않은 경우 JMS 메시지의 MQRFH2 헤더 파트에서 교환됩니다. JMS `TextMessage` 및 JMS `BytesMessage`만 MQRFH2 없이 송신이 가능합니다. JMS 특성이 메시지 디스크립터 MQMD에 IBM MQ 메시지 특성으로 저장되면 MQMD 변환의 일부로 변환됩니다. JMS 특성이 MQRFH2에 저장된 경우, 이는 MQRFH2.NameValueCCSID에서 지정한 문자 세트에 저장됩니다. 메시지를 송신하거나 수신할 때 메시지 특성은 JVM의 자체 내부 표현에 대해 양방향으로 변환됩니다. 변환은 MQRFH2.NameValueCCSID 또는 메시지 디스크립터의 문자 세트에 대해 양방향으로 수행됩니다. 숫자 데이터는 텍스트로 변환됩니다.

JMS 메시지 변환

다음 주제에는 변환이 필요한 보다 복잡한 메시지를 교환하고자 할 때 유용한 예제와 태스크가 포함되어 있습니다.

JMS 메시지 변환 접근 방법

다수의 데이터 변환 접근 방법이 JMS 애플리케이션 디자이너에게 열려 있습니다. 이러한 접근 방법은 배타적이지 않습니다. 일부 애플리케이션은 이러한 접근 방법을 조합하여 사용할 수 있습니다. 애플리케이션이 텍스트만 교환 중이거나 메시지를 다른 JMS 애플리케이션과만 교환 중인 경우에는 일반적으로 데이터 변환을 고려하지 않습니다. 데이터 변환은 사용자를 위해 IBM MQ에서 자동으로 수행합니다.

메시지 변환 접근 방법에 대해 몇 가지 질문을 할 수 있습니다.

메시지 변환에 대해 생각해야 할 필요가 있습니까?

일부 경우(예: JMS 대 JMS 메시지 전송, IBM MQ 프로그램과 텍스트 메시지 교환), IBM MQ는 사용자에게 필요한 변환을 자동으로 수행합니다. 사용자가 성능상 이유로 데이터 변환을 제어하고자 하거나, 사전정의된 형식을 지닌 복잡한 메시지를 교환 중일 수 있습니다. 이와 같은 경우에는 메시지 변환을 이해해야 하며 다음 주제를 읽어야 합니다.

어떤 종류의 변환이 있습니까?

네 가지의 기본 변환 유형이 있으며, 이는 다음 절에 설명되어 있습니다.

1. [161 페이지의 『JMS 클라이언트 데이터 변환』](#)
2. [162 페이지의 『애플리케이션 데이터 변환』](#)
3. [163 페이지의 『큐 관리자 데이터 변환』](#)
4. [163 페이지의 『메시지 채널 데이터 변환』](#)

변환은 어디에서 수행되어야 합니까?

[163 페이지의 『메시지 변환에 대한 접근 방법 선택: 수신자 성공』](#) 절에서는 "수신자 성공"의 일반적인 접근 방법을 설명합니다. "수신자 성공"은 JMS 데이터 변환에도 적용됩니다.

JMS 클라이언트 데이터 변환

JMS 클라이언트¹데이터 변환은 대상으로 전송될 때 Java 기본요소 및 오브젝트를 JMS 메시지의 바이트로 변환하고 수신될 때 다시 변환하는 것입니다. JMS 클라이언트 데이터 변환은 JMS `Message` 클래스의 메소드를 사용합니다. 메소드는 [164 페이지의 표 31](#)의 JMS `Message` 클래스 유형으로 나열됩니다.

숫자 및 텍스트의 내부 JVM 표현에 대한 양방향 변환은 `read`, `get`, `set` 및 `write` 메소드에 대해 수행됩니다. 변환은 메시지가 송신될 때와 수신된 메시지에서 `read` 또는 `get` 메소드가 호출될 때 수행됩니다.

¹ "JMS 클라이언트"는 클라이언트 또는 바인딩 모드에서 실행하는 JMS 인터페이스를 구현하는 IBM MQ classes for JMS(를) 나타냅니다.

메시지 콘텐츠의 쓰기 또는 설정에 사용되는 코드 페이지 및 숫자 인코딩은 목적지의 속성으로 정의되어 있습니다. 목적지 코드 페이지 및 숫자 인코딩은 관리 측면에서 변경이 가능합니다. 애플리케이션은 메시지 콘텐츠의 쓰기나 설정을 제어하는 메시지 특성을 설정하여 목적지 코드 페이지 및 인코딩을 대체할 수도 있습니다.

Native 인코딩으로 정의되지 않은 목적지로 JMSBytesMessage 메시지가 송신될 때 숫자 인코딩을 변환하고자 하는 경우에는 메시지를 송신하기 전에 메시지 특성 JMS_IBM_ENCODING을 설정해야 합니다. "수신자 성공" 패턴을 따르거나 JMS 애플리케이션 간의 메시지를 교환 중인 경우, 애플리케이션은 JMS_IBM_ENCODING을 설정할 필요가 없습니다. 대부분의 경우 Encoding 특성을 Native로 둘 수 있습니다.

JMSStreamMessage, JMSMapMessage 및 JMSTextMessage 메시지의 경우에는 목적지의 문자 세트 ID 특성이 사용됩니다. 숫자가 텍스트 형식으로 쓰여지므로 인코딩은 송신 시에 무시됩니다. 적용할 목적지 문자 세트 특성이 있는 경우, JMS 클라이언트 애플리케이션 프로그램은 메시지를 송신하기 전에 JMS_IBM_CHARACTER_SET를 설정할 필요가 없습니다.

메시지의 데이터를 가져오기 위해 애플리케이션은 JMS 메시지 read 또는 get 메소드를 호출합니다. 이 메소드는 이전 메시지 헤더에 정의된 코드 페이지 및 인코딩을 참조하여 Java 기본요소 및 오브젝트를 올바르게 작성합니다.

JMS 클라이언트 데이터 변환은 하나의 JMS 클라이언트와 다른 클라이언트 간에 메시지를 교환하는 대부분의 JMS 애플리케이션의 요구사항을 충족합니다. 사용자는 명시적 데이터 변환을 코드화하지 않습니다. 또한 파일에 텍스트를 쓸 때 일반적으로 사용되는 java.nio.charset.Charset 클래스를 사용하지 않습니다.

writeString 및 setString 메소드가 사용자 대신 변환을 수행합니다.

JMS 클라이언트 데이터 변환에 대한 자세한 정보는 [174 페이지의 『JMS 클라이언트 메시지 변환 및 인코딩』](#)의 내용을 참조하십시오.

애플리케이션 데이터 변환

JMS 클라이언트 애플리케이션은 java.nio.charset.Charset 클래스를 사용하여 명시적 문자 데이터 변환을 수행할 수 있습니다. [166 페이지의 그림 14](#) 및 [166 페이지의 그림 15](#)의 예를 참조하십시오. 문자열 데이터는 getBytes 메소드를 사용하여 바이트로 변환되며 바이트로 송신됩니다. 바이트 배열 및 Charset를 사용하는 String 생성자를 사용하여 바이트를 다시 텍스트로 변환합니다. 문자 데이터는 encode 및 decode Charset 메소드를 사용하여 변환됩니다. 일반적으로 메시지는 JMSBytesMessage로 송신 또는 수신됩니다. JMSBytesMessage의 메시지 파트에는 애플리케이션이 쓴 데이터 이외의 데이터가 포함되어 있지 않기 때문입니다.² JMSStreamMessage, JMSMapMessage 또는 JMSObjectMessage를 사용하여 바이트를 보내고 받을 수도 있습니다.

서로 다른 인코딩 형식으로 표현된 숫자 데이터가 포함된 바이트를 인코딩하고 디코딩하는 Java 메소드는 없습니다. 숫자 데이터는 숫자 JMSMessage read 및 write 메소드를 사용하여 자동으로 인코딩되고 디코딩됩니다. read 및 write 메소드는 메시지 데이터의 JMS_IBM_ENCODING 속성 값을 사용합니다.

애플리케이션 데이터 변환의 일반적인 사용은 JMS 클라이언트가 비-JMS 애플리케이션에서 형식화된 메시지를 송신하거나 수신하는 경우입니다. 형식화된 메시지에는 데이터 필드의 길이로 구성된 텍스트, 숫자 및 바이트 데이터가 포함되어 있습니다. JMS 이외의 애플리케이션이 메시지 형식을 "MQSTR"로 지정한 경우가 아니면 메시지는 JMSBytesMessage로 생성됩니다. JMSBytesMessage의 형식화된 메시지 데이터를 수신하려면 메소드의 시퀀스를 호출해야 합니다. 메소드는 필드가 메시지에 쓰여진 순서와 동일하게 호출되어야 합니다. 필드가 숫자인 경우에는 숫자 데이터의 인코딩 및 길이를 알고 있어야 합니다. 임의의 필드에 바이트 또는 텍스트 데이터가 포함된 경우에는 메시지에서 바이트 데이터의 길이를 알고 있어야 합니다. 형식화된 메시지를 사용이 용이한 Java 오브젝트로 변환하는 두 가지 방법이 있습니다.

1. 레코드에 해당되는 Java 클래스를 구성하여 메시지 읽기 및 쓰기를 캡슐화합니다. 레코드의 데이터에 대한 액세스는 클래스의 get 및 set 메소드로 이루어집니다.
2. com.ibm.mq.headers 클래스를 확장하여 레코드에 해당하는 Java 클래스를 생성하십시오. 클래스에 있는 데이터에 대한 액세스는 getStringValue(fieldName); 양식의 유형별 액세스서를 사용합니다.

[181 페이지의 『비-JMS 애플리케이션에서 형식화된 레코드 교환』](#)의 내용을 참조하십시오.

² 한 가지 예외: writeUTF를 사용하여 작성된 데이터는 2바이트 길이 필드로 시작합니다

큐 관리자 데이터 변환

JMS 클라이언트 프로그램이 메시지를 가져올 때 큐 관리자가 코드 페이지 변환을 수행할 수 있습니다. 변환은 C 프로그램에 대해 수행된 변환과 동일합니다. C 프로그램은 MQGMO_CONVERT를 MQGET GetMsgOpts 매개변수 옵션으로 설정합니다. [166 페이지의 그림 13](#)의 내용을 참조하십시오. WMQ_RECEIVE_CONVERSION 목적지 특성이 WMQ_RECEIVE_CONVERSION_QMGR로 설정된 경우 큐 관리자가 메시지를 수신하는 JMS 클라이언트 프로그램에 대한 변환을 수행합니다. JMS 클라이언트 프로그램도 목적지 특성을 설정할 수 있습니다. [163 페이지의 그림 12](#)의 내용을 참조하십시오.

```
((MQDestination)destination).setIntProperty(
    WMQConstants.WMQ_RECEIVE_CONVERSION,
    WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

또는

```
((MQDestination)destination).setReceiveConversion(
    WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

그림 12. 큐 관리자 데이터 변환 사용

큐 관리자 변환의 주요 장점은 비-JMS 애플리케이션과 메시지를 교환할 때 나타납니다. 메시지의 Format 필드가 정의되어 있으며 대상 문자 세트 또는 인코딩이 메시지와 다른 경우, 애플리케이션이 요청하면 큐 관리자가 대상 애플리케이션의 데이터 변환을 수행합니다. 큐 관리자는 사전 정의된 IBM MQ 메시지 유형 (예: CICS bridge 헤더 (MQCIH)) 중 하나에 따라 형식화된 메시지 데이터를 변환합니다. Format 필드가 사용자 정의된 경우 큐 관리자는 Format 필드에 제공된 이름을 사용하여 데이터 변환 엑시트를 찾습니다.

큐 관리자 데이터 변환은 "수신자 성공" 디자인 패턴의 최상의 효과에 사용됩니다. 송신 JMS 클라이언트는 변환을 수행할 필요가 없습니다. 비-JMS 수신 프로그램은 변환 엑시트에 의존하여 메시지가 필수 코드 페이지 및 인코딩으로 전달되는지 확인합니다. 송신 JMS 클라이언트 및 비JMS 수신자를 사용하는 경우 예제는 IBM MQ에 적용됩니다.

큐 관리자가 자체 레코드 형식화 데이터를 변환할 수 있도록 사용자는 데이터 변환 엑시트 유틸리티, **crtmqcvx**를 사용하여 데이터 변환 엑시트를 작성할 수 있습니다. 사용자 고유의 레코드 형식을 빌드하고, **com.ibm.mq.headers**을(를) 사용하여 Java 클래스로 액세스하며, 자체 변환 엑시트를 사용하여 이를 변환할 수 있습니다. z/OS에서는 유틸리티를 **CSQUCVX**라고 하며, IBM i에서는 **CVTMQMDTA**라고 합니다. [181 페이지의 『비-JMS 애플리케이션에서 형식화된 레코드 교환』](#)의 내용을 참조하십시오.

메시지 채널 데이터 변환

IBM MQ 송신자, 서버, 클러스터-수신자 및 클러스터-송신자 채널에는 메시지 변환 옵션, CONVERT가 있습니다. 메시지의 콘텐츠는 메시지가 송신될 때 선택적으로 변환될 수 있습니다. 변환은 채널의 송신 측에서 발생합니다. 클러스터-수신자 정의는 해당되는 클러스터-송신자 채널을 자동 정의하는 데 사용됩니다.

메시지 채널에 의한 데이터 변환은 일반적으로 기타 양식의 변환을 사용할 수 없을 때 사용됩니다.

메시지 변환에 대한 접근 방법 선택: "수신자 성공"

코드 변환을 위한 IBM MQ 애플리케이션 디자인의 일반적인 접근 방법은 "수신자 성공"입니다. "수신자 성공"은 메시지 변환의 수를 줄여줍니다. 또한 이는 메시지 전송 중에 일부 중개 큐 관리자에서 메시지 변환이 실패하는 경우에 예상치 못한 채널 오류의 문제점도 피합니다. "수신자 성공" 규칙은 수신자가 성공할 수 없는 일부 이유가 있는 경우에만 위반됩니다. 예를 들어, 수신 플랫폼에 올바른 문자 세트가 없을 수 있습니다.

"수신자 성공"은 JMS 클라이언트 애플리케이션에 대한 바람직한 일반 지침이기도 합니다. 그러나 특정 경우에는 소스에서 올바른 문자 세트로의 변환이 보다 효율적일 수 있습니다. JVM 내부 표현으로부터의 변환은 텍스트 또는 숫자 유형이 포함된 메시지가 송신될 때 발생해야 합니다. 수신자가 JMS 클라이언트가 아닌 경우, 수신자에게 필요한 문자 세트로 변환하면 비-JMS 수신인이 변환을 수행해야 할 필요성이 제거될 수 있습니다. 수신인이 JMS 클라이언트인 경우, 이는 메시지 데이터를 디코딩하고 Java 기본요소 및 오브젝트를 작성하기 위해 어떤 방식으로든 변환합니다.

JMS 클라이언트 애플리케이션 및 C 등의 언어로 작성된 애플리케이션 간의 차이점은 Java가 데이터 변환을 수행해야 하는지 여부입니다. Java 애플리케이션은 숫자 및 텍스트를 자체 내부 표현에서 메시지에서 사용되는 인코딩 형식으로 변환해야 합니다.

목적지 또는 메시지 특성을 설정함으로써, 사용자는 메시지의 숫자 및 텍스트를 인코딩하기 위해 IBM MQ에서 사용하는 문자 세트 및 인코딩을 설정할 수 있습니다. 일반적으로, 사용자는 문자 세트를 1208으로 두고 인코딩을 Native로 둡니다.

IBM MQ는 바이트 배열을 변환하지 않습니다. 문자열 및 문자 배열을 바이트 배열로 인코딩하려면 `java.nio.charset` 패키지를 사용하십시오. `Charset`는 문자열 또는 문자 배열을 바이트 배열로 변환하는데 사용되는 문자 세트를 지정합니다. `Charset`를 사용하여 바이트 배열을 문자열 또는 문자 배열로 디코딩할 수도 있습니다. 문자열 및 문자 배열 인코딩 시 `java.nio.charset.Charset.defaultCodePage`에 의존하는 것은 좋지 않습니다. 기본 `Charset`는 일반적으로 `windows-1252`(Windows) 및 `UTF-8`(AIX and Linux)입니다. `windows-1252`는 1바이트 문자 세트이며, `UTF-8`은 다중 바이트 문자 세트입니다.

일반적으로 다른 JMS 애플리케이션과 메시지를 교환할 때 `UTF-8` 및 `Native`의 기본값으로 대상 문자 세트 및 인코딩 특성을 그대로 두십시오. 숫자 또는 텍스트가 포함된 메시지를 JMS 애플리케이션과 교환 중인 경우에는 사용자 용도에 맞는 `JMSTextMessage`, `JMSStreamMessage`, `JMSMapMessage` 또는 `JMSObjectMessage` 메시지 유형 중 하나를 선택하십시오. 수행할 기타 변환 태스크는 없습니다.

레코드 형식을 사용하는 비-JMS 애플리케이션과 교환 중인 경우에는 보다 복잡합니다. 전체 레코드에 텍스트가 포함되며 `JMSTextMessage`로서 전송 가능하지 않는 한, 사용자는 애플리케이션에서 텍스트를 인코딩하고 디코딩해야 합니다. 목적지 메시지 유형을 MQ로 설정하고 `JMSBytesMessage`를 사용하여 IBM MQ classes for JMS가 추가 헤더 및 태그 지정 정보를 메시지 데이터에 추가하는 것을 피할 수 있습니다. `JMSBytesMessage` 메소드를 사용하여 숫자 및 바이트를 쓸 수 있으며, `Charset` 클래스는 텍스트를 바이트 배열로 명시적으로 변환합니다. 다수의 요인이 문자 세트의 선택에 영향을 줄 수 있습니다.

- 성능: 가장 다수의 서버에서 사용되는 문자 세트로 텍스트를 변환하여 변환 횟수를 줄일 수 있습니까?
- 일관성: 동일한 문자 세트의 모든 메시지를 전송합니다.
- 풍부함: 어떤 문자 세트가 애플리케이션이 사용해야 하는 모든 코드 포인트를 보유합니까?
- 단순성: 1바이트 문자 세트는 가변 길이 및 다중 바이트 문자 세트보다 사용하기가 보다 단순합니다.

181 페이지의 『비-JMS 애플리케이션에서 형식화된 레코드 교환』의 내용을 참조하십시오. 비-JMS 애플리케이션과 교환된 메시지 변환의 예를 확인할 수 있습니다.

예

메시지 유형 및 변환 유형에 대한 표

표 31. 메시지 유형 및 변환 유형				
	변환 유형			
메시지 유형	텍스트	숫자	기타	없음
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			

표 31. 메시지 유형 및 변환 유형 (계속)

메시지 유형	변환 유형			
	텍스트	숫자	기타	없음
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

```
gmo.Options = MQGMO_WAIT          /* wait for new messages          */
              | MQGMO_NO_SYNCPOINT /* no transaction              */
              | MQGMO_CONVERT;    /* convert if necessary        */

while (CompCode != MQCC_FAILED) {
    buflen = sizeof(buffer) - 1; /* buffer size available for GET */
    memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
    memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
    md.Encoding = MQENC_NATIVE;
    md.CodedCharSetId = MQCCSI_Q_MGR;

    MQGET(Hcon,          /* connection handle          */
          Hobj,          /* object handle              */
          &md,            /* message descriptor          */
          &gmo,          /* get message options         */
          buflen,        /* buffer length               */
          buffer,        /* message buffer              */
          &messlen,      /* message length              */
          &CompCode,     /* completion code             */
          &Reason);      /* reason code                  */
}
```

그림 13. *amqsget0.c*의 코드 스니펫

JMSBytesMessage의 텍스트 송신 및 수신

166 페이지의 그림 14의 코드는 BytesMessage에서 문자열을 송신합니다. 단순화를 위해, 예제에서는 단일 문자열을 송신하며 이에 대해 JMSTextMessage가 보다 적절합니다. 혼합 유형이 포함된 바이트 메시지의 텍스트 문자열을 수신하려면, 166 페이지의 그림 15에서 TEXT_LENGTH라고 하는 문자열의 길이(바이트)를 알고 있어야 합니다. 문자 수가 고정된 문자열인 경우에도 바이트 표현의 길이는 보다 길 수 있습니다.

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
    .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

그림 14. JMSBytesMessage 에서 String 전송

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength()).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

그림 15. JMSBytesMessage 에서 String 수신

관련 개념

[JMS 클라이언트 메시지 변환 및 인코딩](#)

JMS 클라이언트 메시지 변환 및 인코딩을 수행하는 데 사용되는 메소드가 각 변환 유형의 코드 예제와 함께 나열되어 있습니다.

[큐 관리자 데이터 변환](#)

JMS 클라이언트에서 메시지를 수신하는 비-JMS 애플리케이션은 큐 관리자 데이터 변환을 항상 사용할 수 있습니다. 메시지를 수신하는 JMS 클라이언트도 선택적인 큐 관리자 데이터 변환을 사용합니다.

관련 태스크

[비-JMS 애플리케이션에서 형식화된 레코드 교환](#)

이 작업에서 제안된 단계를 따라 데이터 변환 출구(exit)를 설계 및 구축하고, 를 사용하여 비JMS 애플리케이션과 메시지를 교환할 JMSBytesMessage 수 있는 클라이언트 JMS 애플리케이션을 개발하십시오. 비-JMS 애플리케이션과 형식화된 메시지의 교환은 데이터 변환 엑시트를 호출하거나 호출하지 않고도 발생할 수 있습니다.

관련 참조

JMS 메시지 유형 및 변환

메시지 유형의 선택은 메시지 변환에 대한 접근 방법에 영향을 줍니다. 메시지 변환 및 메시지 유형의 상호작용이 JMS 메시지 유형, JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessage 및 JMSBytesMessage에 대해 설명되어 있습니다.

JMS 메시지 유형 및 변환

메시지 유형의 선택은 메시지 변환에 대한 접근 방법에 영향을 줍니다. 메시지 변환 및 메시지 유형의 상호작용이 JMS 메시지 유형, JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessage 및 JMSBytesMessage에 대해 설명되어 있습니다.

JMSObjectMessage

JMSObjectMessage에는 하나의 오브젝트와 이 오브젝트가 참조하는 오브젝트가 JVM에 의해 바이트 스트림으로 직렬화되어 포함되어 있습니다. 텍스트는 UTF-8로 직렬화되며, 최대 65534바이트의 문자열 또는 문자 배열로 제한됩니다. JMSObjectMessage의 장점은 오브젝트의 메소드 및 속성만 사용하는 한 애플리케이션이 데이터 변환 문제에 관련되지 않는다는 점입니다. JMSObjectMessage는 애플리케이션 프로그래머가 메시지의 오브젝트를 인코딩하는 방법을 고려하지 않아도 복합 오브젝트에 대한 데이터 변환을 제공합니다. JMSObjectMessage 사용의 단점은 기타 JMS 애플리케이션과의 교환만 가능하다는 점입니다. 기타 JMS 메시지 유형 중 하나를 선택함으로써, JMS 메시지를 비-JMS 애플리케이션과 교환하는 것이 가능합니다.

170 페이지의 『JMSObjectMessage 송신 및 수신』 메시지 내에서 교환되는 객체를 String 보여줍니다.

클라이언트 JMS 애플리케이션은 -스타일 JMS 본문을 가진 메시지에서만 JMSObjectMessage 를 수신할 수 있습니다. 목적지는 JMS 스타일 본문을 지정해야 합니다.

JMSTextMessage

JMSTextMessage에는 단일 텍스트 문자열이 포함되어 있습니다. 문자 메시지가 전송되면, 텍스트는 로 Format 설정됩니다 "MQSTR ". WMQConstants.MQFMT_STRING 텍스트의 CodedCharacterSetId는 해당 목적지에 대해 정의된 코드화 문자 세트 ID로 설정됩니다. 텍스트는 로 CodedCharacterSetId IBM MQ 인코딩됩니다. 및 FormatCodedCharacterSetId 필드는 메시지 디스크립터(메시지)에 설정되거나 MQMD, 메시지 MQRFH2(메시지)의 필드에 JMS 설정됩니다. 메시지가 WMQ_MESSAGE_BODY_MQ 메시지 본문 스타일을 지닌 것으로 정의되거나 본문 스타일이 지정되지 않았지만 대상 목적지가 WMQ_TARGET_DEST_MQ인 경우에는 메시지 디스크립터 필드가 설정됩니다. 그렇지 않으면, 메시지가 JMS RFH2를 보유하며 필드가 MQRFH2의 고정 파트에서 설정됩니다.

애플리케이션은 목적지에 대해 정의된 코드화 문자 세트 ID를 대체할 수 있습니다. 이는 메시지 특성 JMS_IBM_CHARACTER_SET를 코드화 문자 세트 ID로 설정해야 합니다. 170 페이지의 『JMSTextmessage 송신 및 수신』의 예제를 참조하십시오.

JMS 클라이언트가 consumer.receive 메소드를 호출할 때 큐 관리자 변환은 선택사항입니다. 큐 관리자 변환은 목적지 특성 WMQ_RECEIVE_CONVERSION을 WMQ_RECEIVE_CONVERSION_QMGR로 설정하여 사용됩니다. 큐 관리자는 메시지를 클라이언트로 JMS 전송하기 전에 메시지에 대해 지정된 JMS_IBM_CHARACTER_SET 형식으로 텍스트 메시지를 변환합니다. 목적지에 다른 WMQ_RECEIVE_CCSID가 없는 한 변환된 메시지의 문자 세트는 1208, UTF-8입니다. JMSTextMessage를 참조하는 메시지의 CodedCharacterSetId는 대상 문자 세트 ID로 업데이트됩니다. 텍스트는 getText 메소드를 사용하여 대상 문자 세트에서 유니코드로 디코딩됩니다. 170 페이지의 『JMSTextmessage 송신 및 수신』의 예제를 참조하십시오.

JMSTextMessage는 JMS MQRFH2 헤더 없이 MQ 스타일 메시지 본문으로 송신될 수 있습니다. 애플리케이션에 의해 대체되지 않는 한 목적지 속성, WMQ_MESSAGE_BODY 및 WMQ_TARGET_DEST의 값은 메시지 본문 스타일을 판별합니다. 애플리케이션은 destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) 또는

`destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ)`를 호출하여 목적지에서 설정된 값을 대체할 수 있습니다.

`WMQ_MESSAGE_BODY`가 `WMQ_MESSAGE_BODY_MQ`로 설정된 목적지에 송신하여 MQ 스타일 본문의 `JMSTextMessage`를 송신하는 경우에는 동일 목적지에서 `JMSTextMessage`로서 이를 수신할 수 없습니다. `WMQ_MESSAGE_BODY`가 `WMQ_MESSAGE_BODY_MQ`로 설정된 목적지에서 수신된 모든 메시지는 `JMSBytesMessage`로서 수신됩니다. 메시지를 로 수신하려고 하면 `JMSTextMessage` 예외가 발생합니다. `ClassCastException: com.ibm.jms.JMSBytesMessage cannot be cast to jakarta (or javax).jms.TextMessage`

참고: `JMSBytesMessage`의 텍스트는 JMS 클라이언트에 의해 변환되지 않습니다. 클라이언트는 메시지의 텍스트를 바이트 배열로서만 수신할 수 있습니다. 큐 관리자 변환을 사용하는 경우, 텍스트는 큐 관리자에 의해 변환되지만 JMS 클라이언트는 이를 `JMSBytesMessage`의 바이트 배열로서 계속 수신해야 합니다.

`WMQ_TARGET_DEST` 특성을 사용하여 `JMSTextMessage`가 MQ 또는 JMS 본문 스타일로 송신되는지 여부를 제어하는 것이 일반적으로 보다 바람직합니다. 그리고 사용자는 `WMQ_TARGET_DEST`가 `WMQ_TARGET_DEST_MQ` 또는 `WMQ_TARGET_DEST_JMS`로 설정된 목적지에서 메시지를 수신할 수 있습니다. `WMQ_TARGET_DEST`는 수신자에 영향을 주지 않습니다.

JMSMapMessage 및 JMSStreamMessage

이 두 가지 JMS 메시지 유형은 유사합니다. `DataInputStream` 및 `DataOutputStream` 인터페이스 기반의 메소드를 사용하여 메시지에 대해 기본 유형을 읽고 쓸 수 있습니다. 172 페이지의 『메시지 유형 및 변환 유형에 대한 표』의 내용을 참조하십시오. 세부사항은 174 페이지의 『JMS 클라이언트 메시지 변환 및 인코딩』에 설명되어 있습니다. 각 기본요소에는 태그가 지정되어 있습니다. 159 페이지의 『JMS 메시지 본문』의 내용을 참조하십시오.

숫자 데이터는 XML 텍스트로서 인코딩된 메시지에 대해 읽히고 쓰여집니다. 목적지 특성 `JMS_IBM_ENCODING`에 대한 참조는 작성되지 않습니다. 텍스트 데이터는 `JMSTextMessage`의 텍스트와 동일한 방법으로 처리됩니다. 171 페이지의 그림 20의 예제에서 작성한 메시지 콘텐츠를 보는 경우, 모든 메시지 데이터는 37의 문자 세트 값으로 송신되었으므로 EBCDIC입니다.

`JMSMapMessage` 또는 `JMSStreamMessage`에서 다중 항목을 송신할 수 있습니다.

`JMSMapMessage`에서 이름별로 또는 `JMSStreamMessage`에서 위치별로 데이터의 개별 항목을 검색할 수 있습니다. 각 항목은 메시지에 저장된 `CodedCharacterSetId` 값을 사용하여 `get` 또는 `read` 메소드가 호출될 때 디코딩됩니다. 항목 검색에 사용된 메소드가 송신된 유형과는 다른 유형을 리턴하는 경우에는 해당 유형이 변환됩니다. 유형을 변환할 수 없는 경우에는 예외가 전달됩니다. 세부사항은 클래스 `JMSStreamMessage`를 참조하십시오. 171 페이지의 『JMSStreamMessage 및 JMSMapMessage에서 데이터 송신』의 예제에서는 유형 변환 및 순서가 뒤바뀐 `JMSMapMessage` 콘텐츠 가져오기를 설명합니다.

와 `JMSMapMessage`의 필드는 `MQRFH2.formatJMSStreamMessage`로 설정됩니다 "MQSTR". 목적지 특성 `WMQ_RECEIVE_CONVERSION`이 `WMQ_RECEIVE_CONVERSION_QMGR`로 설정된 경우, 메시지 데이터는 JMS 클라이언트에 송신되기 전에 큐 관리자에 의해 변환됩니다. 메시지의 `MQRFH2.CodedCharacterSetId`는 목적지의 `WMQ_RECEIVE_CCSID`입니다. `MQRFH2.Encoding`은 Native입니다.

`WMQ_RECEIVE_CONVERSION`이 `WMQ_RECEIVE_CONVERSION_CLIENT_MSG`인 경우, `MQRFH2.CodedCharacterSetId` 및 `Encoding`은 송신자가 설정한 값입니다.

클라이언트 JMS 애플리케이션은 -스타일 JMS 본문을 가진 메시지에서만 또는 `JMSStreamMessage` `JMSMapMessage`을 수신할 수 있으며, 이는 MQ 스타일 본문을 지정하지 않은 목적지에서만 가능합니다.

JMSBytesMessage

`JMSBytesMessage`에는 다수의 기본 유형이 포함될 수 있습니다. `DataInputStream` 및 `DataOutputStream` 인터페이스 기반의 메소드를 사용하여 메시지에 대해 기본 유형을 읽고 쓸 수 있습니다. 172 페이지의 『메시지 유형 및 변환 유형에 대한 표』의 내용을 참조하십시오. 세부사항은 167 페이지의 『JMS 메시지 유형 및 변환』에 설명되어 있습니다.

메시지에서 숫자 데이터의 인코딩은 숫자 데이터를 `JMSBytesMessage`에 쓰기 전에 설정된 `JMS_IBM_ENCODING` 값에 의해 제어됩니다. 애플리케이션은 메시지 특성 `JMS_IBM_ENCODING`을 설정하여 `JMSBytesMessage`에 대해 정의된 기본 Native 인코딩을 대체할 수 있습니다.

텍스트 데이터는 `readUTF` 및 `writeUTF`를 사용하여 UTF-8로, 그리고 `readChar` 및 `writeChar` 메소드를 사용하여 유니코드로 읽고 쓸 수 있습니다. `CodedCharacterSetId`를 사용하는 메소드는 없습니다. 또는 JMS 클라이언트는 `Charset` 클래스를 사용하여 텍스트를 바이트로 인코딩하고 디코딩할 수 있습니다. 이는 변환을 수행하는 IBM MQ classes for JMS 없이 JVM 및 메시지 간에 바이트를 전송합니다. [171 페이지의 『JMSBytesMessage의 텍스트 송신 및 수신』](#)의 내용을 참조하십시오.

MQ 애플리케이션에 송신된 `JMSBytesMessage`는 일반적으로 JMS MQRFH2 헤더 없이 MQ 스타일 메시지 본문에서 송신됩니다. JMS 애플리케이션에 송신된 경우, 메시지 본문 스타일은 일반적으로 JMS입니다. 애플리케이션에 의해 대체되지 않는 한 목적지 속성, `WMQ_MESSAGE_BODY` 및 `WMQ_TARGET_DEST`의 값은 메시지 본문 스타일을 판별합니다. 애플리케이션은 `destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ)` 또는 `destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ)`를 호출하여 목적지에서 설정된 값을 대체할 수 있습니다.

MQ 스타일 본문이 있는 `JMSBytesMessage`를 송신하는 경우에는 MQ 또는 JMS 메시지 본문 스타일을 정의하는 목적지에서 메시지를 수신할 수 있습니다. JMS 스타일 본문이 있는 `JMSBytesMessage`를 송신하는 경우에는 JMS 메시지 본문 스타일을 정의하는 목적지에서 메시지를 수신해야 합니다. 그렇지 않으면, MQRFH2는 사용자 메시지 데이터의 일부로서 처리되며 이는 예상과 다를 수 있습니다.

메시지에 MQ 또는 JMS 본문 스타일이 있는지와 무관하게, 수신되는 방법은 `WMQ_TARGET_DEST` 설정의 영향을 받지 않습니다.

`Format`이 메시지 데이터에 대해 제공되며 큐 관리자 데이터 변환이 사용되는 경우, 메시지는 큐 관리자에 의해 나중에 변환될 수 있습니다. 메시지 데이터의 형식 지정 외에는 어떤 경우에도 형식 필드를 사용하지 않거나 이를 공백으로 두십시오(`MQConstants.MQFMT_NONE`).

`JMSBytesMessage`에서 다중 항목을 송신할 수 있습니다. 각각의 숫자 항목은 메시지에 대해 정의된 인코딩을 사용하여 메시지가 송신될 때 변환됩니다.

`JMSBytesMessage`에서 데이터의 개별 항목을 검색할 수 있습니다. 메시지를 작성하기 위해 `write` 메소드가 호출된 순서와 동일하게 `read` 메소드를 호출하십시오. 각각의 숫자 항목은 메시지에 저장된 `Encoding` 값을 사용하여 메시지가 호출될 때 변환됩니다.

`JMSMapMessage` 및 `JMSStreamMessage`와는 달리, `JMSBytesMessage`에는 애플리케이션이 쓴 데이터만 포함됩니다. 추가 데이터가 메시지 데이터에 저장되지 않습니다(예: `JMSMapMessage` 및 `JMSStreamMessage`에서 항목을 정의하는 데 사용된 XML 태그). 이러한 이유 때문에, `JMSBytesMessage`를 사용하여 기타 애플리케이션에 대해 형식화된 메시지를 전송하십시오.

`JMSBytesMessage` 및 `DataInputStream` 및 `DataOutputStream` 간의 변환은 일부 애플리케이션에서 유용합니다. 예를 기본으로 한 코드는 `com.ibm.mq.header` 패키지를 와 JMS 함께 [171 페이지의 『DataInputStream 및 DataOutputStream를 사용하여 메시지 읽기 및 쓰기』](#) 사용하기 위해 필요합니다.

예

JMSObjectMessage 송신 및 수신

```
ObjectMessage omo = session.createObjectMessage();
omo.setObject(new String("A string"));
producer.send(omo);
...
ObjectMessage omi = (ObjectMessage)consumer.receive();
System.out.println((String)omi.getObject());
...
A string
```

그림 16. JMSObjectMessage 송신 및 수신

JMSTextmessage 송신 및 수신

텍스트 메시지에는 서로 다른 문자 세트의 텍스트가 포함될 수 없습니다. 예제에서는 두 개의 서로 다른 메시지에
서 송신되는 서로 다른 문자 세트의 텍스트를 표시합니다.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

그림 17. 목적지에서 정의한 문자 세트의 텍스트 메시지 송신

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

그림 18. ccsid 37의 텍스트 메시지 송신

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

그림 19. 텍스트 메시지 수신

JMSStreamMessage 및 JMSMapMessage에서 데이터 송신

```
StreamMessage smo = session.createStreamMessage();
smo.writeString("256");
smo.writeInt(512);
smo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(smo);
...
MapMessage mmo = session.createMapMessage();
mmo.setString("First", "256");
mmo.setInt("Second", 512);
mmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(mmo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println("Stream: First as float " + smi.readFloat() +
    " Second as String " + smi.readString());
...
Stream: First as float: 256.0, Second as String: 512
...
MapMessage mmi = (MapMessage)consumer.receive();
System.out.println("Map: Second as String " + mmi.getString("Second") +
    " First as double " + mmi.getDouble("First"));
...
Map: Second as String: 512, First as double: 256.0
```

그림 20. JMSStreamMessage 및 JMSMapMessage에서 데이터 송신

JMSBytesMessage의 텍스트 송신 및 수신

171 페이지의 그림 21의 코드는 BytesMessage에서 문자열을 송신합니다. 단순화를 위해, 예제에서는 단일 문자열을 송신하며 이에 대해 JMSTextMessage가 보다 적절합니다. 혼합 유형이 포함된 바이트 메시지의 텍스트 문자열을 수신하려면, 171 페이지의 그림 22에서 TEXT_LENGTH라고 하는 문자열의 길이(바이트)를 알고 있어야 합니다. 문자 수가 고정된 문자열인 경우에도 바이트 표현의 길이는 보다 길 수 있습니다.

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
    .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

그림 21. JMSBytesMessage 에서 String 전송

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength()).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

그림 22. JMSBytesMessage 에서 String 수신

DataInputStream 및 DataOutputStream를 사용하여 메시지 읽기 및 쓰기

172 페이지의 그림 23의 코드는 DataOutputStream을 사용하여 JMSBytesMessage를 작성합니다.

```

ByteArrayOutputStream bout = new ByteArrayOutputStream();
DataOutputStream dout = new DataOutputStream(bout);
BytesMessage messageOut = prod.session.createBytesMessage();
// messageOut.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
//                             ((MQDestination) prod.destination).getIntProperty
//                             (WMQConstants.WMQ_ENCODING));
int ccsidOut = (((MQDestination) prod.destination).getIntProperty(WMQConstants.WMQ_CCSID));
String codePageOut = CCSID.getCodepage(ccsidOut);
dout.writeInt(ccsidOut);
dout.write(codePageOut.getBytes(codePageOut));
messageOut.writeBytes(bout.toByteArray());
producer.send(messageOut);

```

그림 23. *DataOutputStream*을 사용하여 *JMSBytesMessage* 송신

JMS_IBM_ENCODING 특성을 설정하는 명령문은 주석 처리되어 있습니다. JMSBytesMessage에 직접 쓰는 경우에는 명령문이 유효하지만, *DataOutputStream*에 쓰는 경우에는 효과가 없습니다. *DataOutputStream*에 쓰여진 숫자는 Native 인코딩으로 인코딩됩니다. JMS_IBM_ENCODING 설정은 효과가 없습니다.

172 페이지의 그림 24의 코드는 *DataInputStream*을 사용하여 *JMSBytesMessage*를 수신합니다.

```

static final int ccsidIn_SIZE = (Integer.SIZE)/8;
...
connection.start();
BytesMessage messageIn = (BytesMessage) consumer.receive();
int messageLength = new Long(messageIn.getBodyLength()).intValue();
byte [] bin = new byte[messageLength];
messageIn.readBytes(bin, messageLength);
DataInputStream din = new DataInputStream(new ByteArrayInputStream(bin));
int ccsidIn = din.readInt();
byte [] codePageByte = new byte[messageLength - ccsidIn_SIZE];
din.read(codePageByte, 0, codePageByte.length);
System.out.println("CCSID " + ccsidIn + " code page " + new String(codePageByte,
    messageIn.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET)));

```

그림 24. *DataInputStream*을 사용하여 *JMSBytesMessage* 수신

코드 페이지는 입력 메시지 데이터의 코드 페이지 특성, JMS_IBM_CHARACTER_SET를 사용하여 인쇄됩니다. 입력에서 JMS_IBM_CHARACTER_SET는 Java 코드 페이지이며 숫자 코드화 문자 세트 ID가 아닙니다.

메시지 유형 및 변환 유형에 대한 표

표 32. 메시지 유형 및 변환 유형				
	변환 유형			
메시지 유형	텍스트	숫자	기타	없음
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			

표 32. 메시지 유형 및 변환 유형 (계속)				
	변환 유형			
메시지 유형	텍스트	숫자	기타	없음
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

관련 개념

JMS 메시지 변환 접근 방법

다수의 데이터 변환 접근 방법이 JMS 애플리케이션 디자이너에게 열려 있습니다. 이러한 접근 방법은 배타적이지 않습니다. 일부 애플리케이션은 이러한 접근 방법을 조합하여 사용할 수 있습니다. 애플리케이션이 텍스트만 교환 중이거나 메시지를 다른 JMS 애플리케이션과만 교환 중인 경우에는 일반적으로 데이터 변환을 고려하지 않습니다. 데이터 변환은 사용자를 위해 IBM MQ에서 자동으로 수행합니다.

JMS 클라이언트 메시지 변환 및 인코딩

JMS 클라이언트 메시지 변환 및 인코딩을 수행하는 데 사용되는 메소드가 각 변환 유형의 코드 예제와 함께 나열되어 있습니다.

큐 관리자 데이터 변환

JMS 클라이언트에서 메시지를 수신하는 비-JMS 애플리케이션은 큐 관리자 데이터 변환을 항상 사용할 수 있습니다. 메시지를 수신하는 JMS 클라이언트도 선택적인 큐 관리자 데이터 변환을 사용합니다.

관련 태스크

비-JMS 애플리케이션에서 형식화된 레코드 교환

이 작업에서 제안된 단계를 따라 데이터 변환 출구(exit)를 설계 및 구축하고, 를 사용하여 비JMS 애플리케이션과 메시지를 교환할 JMSBytesMessage수 있는 클라이언트 JMS 애플리케이션을 개발하십시오. 비-JMS 애플리케이션과 형식화된 메시지의 교환은 데이터 변환 엑시트를 호출하거나 호출하지 않고도 발생할 수 있습니다.

JMS 클라이언트 메시지 변환 및 인코딩

JMS 클라이언트 메시지 변환 및 인코딩을 수행하는 데 사용되는 메소드가 각 변환 유형의 코드 예제와 함께 나열되어 있습니다.

변환 및 인코딩은 Java 기본요소 또는 오브젝트가 JMS 메시지에 대해 양방향으로 읽혀지거나 쓰여질 때 발생합니다. 큐 관리자 데이터 변환 및 애플리케이션 데이터 변환과 구분하기 위해, 변환을 JMS 클라이언트 데이터 변환이라고 합니다. 데이터를 JMS 메시지에서 읽거나 쓸 때는 변환이 엄격하게 이루어집니다. 텍스트는 내부 16비트 유니코드 표시로(부터) 변환됨³에 대해 양방향으로 메시지의 텍스트에 사용되는 문자 세트에 대해 변환됩니다. 숫자 데이터가 변환되며 Java 기본 숫자 유형이 메시지에 대해 정의된 인코딩으로 변환됩니다. 변환이 수행되는지 여부와 수행되는 변환의 유형은 JMS 메시지 유형 및 읽기/쓰기 조작에 달려 있습니다.

174 페이지의 표 33에는 수행되는 변환의 유형별로 서로 다른 JMS 메시지 유형에 대해 읽기/쓰기 메소드가 분류되어 있습니다. 변환 유형은 표 이후의 텍스트에 설명되어 있습니다.

표 33. 메시지 유형 및 변환 유형				
	변환 유형			
메시지 유형	텍스트	숫자	기타	없음
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar

³ 일부 유니코드 표시는 16비트 이상을 필요로 합니다. Java SE 참조를 참조하십시오.

표 33. 메시지 유형 및 변환 유형 (계속)

메시지 유형	변환 유형			
	텍스트	숫자	기타	없음
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

텍스트

목적지의 기본 CodedCharacterSetId는 1208, UTF-8입니다. 기본적으로, 텍스트는 유니코드에서 변환되며 UTF-8 텍스트 문자열로 송신됩니다. 수신 시에 텍스트는 클라이언트가 수신한 메시지의 코드화 문자 세트에서 유니코드로 변환됩니다.

setText 및 writeString 메소드는 유니코드의 텍스트를 목적지에 대해 정의된 문자 세트로 변환합니다. 애플리케이션은 메시지 특성 JMS_IBM_CHARACTER_SET를 설정하여 목적지 문자 세트를 대체할 수 있습니다. JMS_IBM_CHARACTER_SET는 메시지를 전송할 때 숫자로 코드화된 문자 세트 ID여야 합니다⁴.

177 페이지의 『JMSTextmessage 송신 및 수신』의 코드 스니펫은 두 개의 메시지를 송신합니다. 하나는 목적지에 대해 정의된 문자 세트에서 송신되며, 다른 하나는 애플리케이션이 정의한 문자 세트 37에서 송신됩니다.

getText 및 readString 메소드는 메시지의 텍스트를 메시지에 정의된 문자 세트에서 유니코드로 변환합니다. 메소드는 메시지 특성, JMS_IBM_CHARACTER_SET에 정의된 코드 페이지를 사용합니다. 메시지가 MQ 유형 메시지이며 MQRFH2를 보유하지 않는 경우가 아니면 코드 페이지는 MQRFH2.CodedCharacterSetId에서 맵핑됩니다. 메시지가 MQ 유형 메시지이며 MQRFH2를 보유하지 않는 경우, 코드 페이지는 MQMD.CodedCharacterSetId에서 맵핑됩니다.

178 페이지의 그림 29의 코드 스니펫은 목적지에 송신된 메시지를 수신합니다. 메시지의 텍스트는 코드 페이지 IBM037에서 다시 유니코드로 변환됩니다.

참고: 텍스트가 코드화 문자 세트 37로 변환되었는지 확인하는 간단한 방법은 IBM MQ Explorer를 사용하는 것입니다. 큐를 찾아보고 검색 전에 메시지의 특성을 표시하십시오.

⁴ 메시지를 수신할 때 JMS_IBM_CHARACTER_SET는 Java Charset 코드 페이지 이름입니다.

178 페이지의 그림 28의 코드 스니펫을 176 페이지의 그림 25의 올바르게 않은 코드 스니펫과 비교하십시오. 올바르게 않은 스니펫에서 텍스트 문자열은 두 번 변환됩니다. 한 번은 애플리케이션에 의해, 그리고 이는 다시 IBM MQ에 의해 변환됩니다.

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText(new String("Sent in EBCDIC character set 37".getBytes(CCSID.getCodepage(37))));
producer.send(tmo);
```

그림 25. 올바르게 않은 코드 페이지 변환

writeUTF 메소드는 유니코드에서 1208, UTF-8로 텍스트를 변환합니다. 텍스트 문자열은 2바이트 길이로 시작됩니다. 텍스트 문자열의 최대 길이는 65534바이트입니다. readUTF 메소드는 writeUTF 메소드에 의해 쓰여진 메시지의 항목을 읽습니다. 이는 정확히 writeUTF 메소드에 의해 쓰여진 바이트 수를 읽습니다.

숫자

목적지의 기본 숫자 인코딩은 Native입니다. Java 에 대한 Native 인코딩 상수의 값은 273, x'00000111'입니다. 이 값은 모든 플랫폼에서 동일합니다. 수신 시에 메시지의 숫자는 정확히 숫자 Java 기본요소로 변환됩니다. 변환은 메시지에 정의된 인코딩 및 read 메소드에 의해 리턴된 유형을 사용합니다.

send 메소드는 set 및 write에 의해 메시지에 추가된 숫자를 목적지에 대해 정의된 숫자 인코딩으로 변환합니다. 목적지 인코딩은 메시지 특성, JMS_IBM_ENCODING을 설정하는 애플리케이션에 의해 메시지에 대해 대체될 수 있습니다. 예를 들면, 다음과 같습니다.

```
message.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
WMQConstants.WMQ_ENCODING_INTEGER_REVERSED);
```

get 및 read 숫자 메소드는 메시지에 정의된 숫자 인코딩에서 메시지의 숫자를 변환합니다. 이는 숫자를 read 또는 get 메소드에 의해 지정된 유형으로 변환합니다. ENCODING 특성을 참조하십시오. 메소드는 JMS_IBM_ENCODING에 정의된 인코딩을 사용합니다. 메시지가 MQ 유형 메시지이며 MQRFH2를 보유하지 않는 경우가 아니면 인코딩은 MQRFH2.Encoding에서 맵핑됩니다. 메시지가 MQ 유형 메시지이며 MQRFH2를 보유하지 않는 경우, 메소드는 MQMD.Encoding에 정의된 인코딩을 사용합니다.

178 페이지의 그림 30의 예제는 목적지 형식의 숫자를 인코딩하고 JMSStreamMessage에서 이를 송신하는 애플리케이션을 표시합니다. 178 페이지의 그림 30의 예제를 178 페이지의 그림 31의 예제와 비교하십시오. 차이점은 JMS_IBM_ENCODING이 JMSBytesMessage에 설정되어야 한다는 점입니다.

참고: 숫자가 올바르게 인코딩되었는지 확인하는 간단한 방법은 IBM MQ Explorer를 사용하는 것입니다. 큐를 찾아보고 이용 전에 메시지의 특성을 표시하십시오.

기타

JMSByteMessage, JMSStreamMessage 및 JMSMapMessage에서 boolean 메소드는 true 및 false를 x'01' 및 x'00'으로 인코딩합니다.

UTF 메소드는 유니코드를 UTF-8 텍스트 문자열로 인코딩하고 디코딩합니다. 문자열은 65536자 미만으로 제한되며, 2바이트 길이 필드로 시작됩니다.

오브젝트 메소드는 기본 유형을 오브젝트로 래핑합니다. 숫자 및 텍스트 유형은 기본 유형이 숫자 및 텍스트 메소드를 사용하여 읽혀지거나 쓰여진 것처럼 인코딩되거나 변환됩니다.

없음

readByte, readBytes, readUnsignedByte, writeByte 및 writeBytes 메소드는 변환 없이 애플리케이션 및 메시지 간에 단일 바이트 또는 바이트 배열을 가져오거나 넣습니다. readChar 및 writeChar 메소드는 변환 없이 애플리케이션 및 메시지 간에 2바이트 유니코드 문자를 가져오고 넣습니다.

readBytes 및 writeBytes 메소드를 사용하여, 애플리케이션은 178 페이지의 『JMSBytesMessage의 텍스트 송신 및 수신』에서와 같이 자체 코드 포인트 변환을 수행할 수 있습니다.

메시지가 `JMSBytesMessage`이고 `readBytes` 및 `writeBytes` 메소드가 사용되므로 IBM MQ 는 클라이언트에서 코드 페이지 변환을 수행하지 않습니다. 그럼에도 불구하고, 바이트가 텍스트를 표시하는 경우에는 애플리케이션이 사용하는 코드 페이지가 목적지의 코드화 문자 세트와 일치하는지 확인하십시오. 메시지는 큐 관리자 변환 엑시트에 의해 다시 변환될 수 있습니다. 다른 가능성은 수신 JMS 클라이언트 프로그램이 메시지의 텍스트를 표시하는 바이트 배열을 메시지의 `JMS_IBM_CHARACTER_SET` 특성을 사용하는 문자열 또는 문자로 변환하는 규약을 따르는 것입니다.

이 예제에서 클라이언트는 해당 규약에 대해 목적지 코드화 문자 세트를 사용합니다.

```
bytes.writeBytes("In the destination code page".getBytes(
CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

또는 클라이언트가 코드 페이지를 선택한 후에 메시지의 `JMS_IBM_CHARACTER_SET` 특성에서 해당되는 코드화 문자 세트를 설정했을 수 있습니다. IBM MQ classes for Java 는 `JMS_IBM_CHARACTER_SET` 를 사용하여 `MQRFH2` 또는 메시지 디스크립터 `MQMD`의 JMS 특성에서 `CodedCharacterSetId` 필드를 설정합니다.

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage);
```

바이트 배열이 `JMSStringMessage` 또는 `JMSMapMessage`에 기록되는 경우, 바이트가 `JMSStringMessage` 및 `JMSMapMessage`의 텍스트가 아닌 16진데이터로 입력되므로 IBM MQ classes for JMS 는 데이터 변환을 수행하지 않습니다.

바이트가 애플리케이션에서 문자를 표시하는 경우에는 메시지에 대해 읽고 쓸 코드 포인트를 고려해야 합니다. 177 페이지의 그림 26의 코드는 목적지 코드화 문자 세트 사용 규약을 따릅니다. JVM에 대한 기본 문자 세트를 사용하는 문자열을 작성하는 경우, 바이트 콘텐츠는 플랫폼에 따라 다릅니다.

UTF-8/windows-1252/JVM 의 기본값은 Windows 일반적으로이며, 의 기본값은 AIX and LinuxCharset 입니다. Windows 및 AIX and Linux 사이의 교환인 경우, 바이트로 텍스트를 교환하기 위해서는 명시적 코드 페이지를 선택해야 합니다.

```
StreamMessage smo = producer.session.createStreamMessage();
smo.writeBytes("123".getBytes(CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

그림 26. 목적지 문자 세트를 사용하여 `JMSStreamMessage`의 문자열을 표시하는 바이트 쓰기

예

JMSTextmessage 송신 및 수신

텍스트 메시지에는 서로 다른 문자 세트의 텍스트가 포함될 수 없습니다. 예제에서는 두 개의 서로 다른 메시지에서 송신되는 서로 다른 문자 세트의 텍스트를 표시합니다.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

그림 27. 목적지에서 정의한 문자 세트의 텍스트 메시지 송신

⁵ `SetStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage)` currently accepts only numeric character set identifiers.

```

    TextMessage tmo = session.createTextMessage();
    tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
    tmo.setText("Sent in EBCDIC character set 37");
    producer.send(tmo);

```

그림 28. *ccsid* 37의 텍스트 메시지 송신

```

    TextMessage tmi = (TextMessage)consumer.receive();
    System.out.println(tmi.getText());
    ...
    Sent in the character set defined for the destination

```

그림 29. 텍스트 메시지 수신

인코딩 예제

인코딩에서 송신 중인 숫자를 표시하는 예제는 목적지에 대해 정의합니다. 사용자는 `JMSBytesMessage`의 `JMS_IBM_ENCODING` 특성을 목적지에 대해 지정된 값으로 설정해야 함을 유념하십시오.

```

    StreamMessage smo = session.createStreamMessage();
    smo.writeInt(256);
    producer.send(smo);
    ...
    StreamMessage smi = (StreamMessage)consumer.receive();
    System.out.println(smi.readInt());
    ...
    256

```

그림 30. `JMSStreamMessage`의 목적지 인코딩을 사용하여 숫자 송신

```

    BytesMessage bmo = session.createBytesMessage();
    bmo.writeInt(256);
    int encoding = ((MQDestination) (destination)).getIntProperty(
        WMQConstants.WMQ_ENCODING);
    bmo.setIntProperty(WMQConstants.JMS_IBM_ENCODING, encoding);
    producer.send(bmo);
    ...
    BytesMessage bmi = (BytesMessage)consumer.receive();
    System.out.println(bmi.readInt());
    ...
    256

```

그림 31. `JMSBytesMessage`의 목적지 인코딩을 사용하여 숫자 송신

`JMSBytesMessage`의 텍스트 송신 및 수신

179 페이지의 그림 32의 코드는 `BytesMessage`에서 문자열을 송신합니다. 단순화를 위해, 예제에서는 단일 문자열을 송신하며 이에 대해 `JMSTextMessage`가 보다 적절합니다. 혼합 유형이 포함된 바이트 메시지의 텍스트 문자열을 수신하려면, 179 페이지의 그림 33에서 `TEXT_LENGTH`라고 하는 문자열의 길이(바이트)를 알고 있어야 합니다. 문자 수가 고정된 문자열인 경우에도 바이트 표현의 길이는 보다 길 수 있습니다.


```

BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
    .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);

```

그림 32. *JMSBytesMessage* 에서 *String* 전송

```

BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength()).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);

```

그림 33. *JMSBytesMessage* 에서 *String* 수신

관련 개념

JMS 메시지 변환 접근 방법

다수의 데이터 변환 접근 방법이 JMS 애플리케이션 디자이너에게 열려 있습니다. 이러한 접근 방법은 배타적이지 않습니다. 일부 애플리케이션은 이러한 접근 방법을 조합하여 사용할 수 있습니다. 애플리케이션이 텍스트만 교환 중이거나 메시지를 다른 JMS 애플리케이션과만 교환 중인 경우에는 일반적으로 데이터 변환을 고려하지 않습니다. 데이터 변환은 사용자를 위해 IBM MQ에서 자동으로 수행합니다.

큐 관리자 데이터 변환

JMS 클라이언트에서 메시지를 수신하는 비-JMS 애플리케이션은 큐 관리자 데이터 변환을 항상 사용할 수 있습니다. 메시지를 수신하는 JMS 클라이언트도 선택적인 큐 관리자 데이터 변환을 사용합니다.

관련 태스크

비-JMS 애플리케이션에서 형식화된 레코드 교환

이 작업에서 제안된 단계를 따라 데이터 변환 출구(exit)를 설계 및 구축하고, 를 사용하여 비JMS 애플리케이션과 메시지를 교환할 JMSBytesMessage 수 있는 클라이언트 JMS 애플리케이션을 개발하십시오. 비-JMS 애플리케이션과 형식화된 메시지의 교환은 데이터 변환 엑시트를 호출하거나 호출하지 않고도 발생할 수 있습니다.

관련 참조

JMS 메시지 유형 및 변환

메시지 유형의 선택은 메시지 변환에 대한 접근 방법에 영향을 줍니다. 메시지 변환 및 메시지 유형의 상호작용이 JMS 메시지 유형, JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessage 및 JMSBytesMessage에 대해 설명되어 있습니다.

큐 관리자 데이터 변환

JMS 클라이언트에서 메시지를 수신하는 비-JMS 애플리케이션은 큐 관리자 데이터 변환을 항상 사용할 수 있습니다. 메시지를 수신하는 JMS 클라이언트도 선택적인 큐 관리자 데이터 변환을 사용합니다.

큐 관리자는 메시지 데이터에 대해 설정된 CodedCharacterSetId, Encoding 및 Format의 값을 사용하여 메시지 데이터의 문자 및 숫자 데이터를 변환할 수 있습니다. 비-JMS 애플리케이션의 경우에는 GetMessageOption, GMO_CONVERT를 설정하여 변환 기능을 항상 사용할 수 있었습니다.

큐 관리자는 JMS 클라이언트로 송신되는 메시지를 변환할 수 있습니다. 큐 관리자 변환은 목적지 특성 WMQ_RECEIVE_CONVERSION을 WMQ_RECEIVE_CONVERSION_QMGR또는 WMQ_RECEIVE_CONVERSION_CLIENT_MSG로 설정하여 제어됩니다. 애플리케이션은 목적지 설정을 변경할 수 있습니다.

```
((MQDestination)destination).setIntProperty(
    WMQConstants.WMQ_RECEIVE_CONVERSION,
    WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

또는

```
((MQDestination)destination).setReceiveConversion
    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

그림 34. 큐 관리자 데이터 변환 사용

JMS 클라이언트에 대한 큐 관리자 데이터 변환은 클라이언트가 `consumer.receive` 메소드를 호출할 때 발생합니다. 텍스트 데이터는 기본적으로 UTF-8(1208)로 변환됩니다. 후속 `read` 및 `get` 메소드는 UTF-8에서 수신된 데이터의 텍스트를 디코딩하며, 자체 내부 유니코드 인코딩에서 Java 텍스트 기본요소를 작성합니다. UTF-8은 큐 관리자 데이터 변환의 유일한 대상 문자 세트가 아닙니다. `WMQ_RECEIVE_CCSID` 목적지 특성을 설정하여 다른 CCSID를 선택할 수 있습니다.

애플리케이션은 목적지 설정을 변경할 수도 있습니다(예: 이를 437, DOS-US로 설정함).

```
((MQDestination)destination).setIntProperty
    (WMQConstants.WMQ_RECEIVE_CCSID, 437);
```

또는

```
((MQDestination)destination).setReceiveCCSID(437);
```

그림 35. 큐 관리자 변환을 위한 대상 코드화 문자 세트 설정

`WMQ_RECEIVE_CCSID` 변경의 이유는 특수합니다. 선택된 CCSID는 JVM에서 작성된 텍스트 오브젝트에 대해 차이가 없습니다. 그러나 일부 플랫폼의 일부 JVM은 메시지의 텍스트의 CCSID에서 유니코드로의 변환을 핸들링할 수 없습니다. 이 옵션을 사용하면 메시지에서 클라이언트에 전달된 텍스트에 대한 CCSID를 선택할 수 있습니다. 일부 JMS 클라이언트 플랫폼은 UTF-8에서 전달되는 메시지 텍스트에 문제점이 있습니다.

JMS 코드는 180 페이지의 그림 36에서 C 코드의 굵은체 텍스트와 동일합니다.

```
gmo.Options = MQGMO_WAIT           /* wait for new messages          */
              | MQGMO_NO_SYNCPOINT /* no transaction                 */
              | MQGMO_CONVERT;    /* convert if necessary         */

while (CompCode != MQCC_FAILED) {
    buflen = sizeof(buffer) - 1; /* buffer size available for GET */
    memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
    memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
    md.Encoding = MQENC_NATIVE;
    md.CodedCharSetId = MQCCSI_Q_MGR;

    MQGET(Hcon, /* connection handle */
          Hobj, /* object handle */
          &md, /* message descriptor */
          &gmo, /* get message options */
          buflen, /* buffer length */
          buffer, /* message buffer */
          &messlen, /* message length */
          &CompCode, /* completion code */
          &Reason); /* reason code */
```

그림 36. *amqsget0.c*의 코드 스니펫

참고:

큐 관리자 변환은 알려진 IBM MQ 형식의 메시지 데이터에서만 수행됩니다. MQSTR, 또는 MQCIH 는 사전 정의된 알려진 형식의 예입니다. 데이터-변환 엑시트를 제공하는 한, 알려진 형식은 사용자 정의된 형식일 수도 있습니다.

JMSTextMessage, JMSMapMessage 및 JMSStreamMessage로서 구성된 메시지의 형식은 MQSTR이며, 큐 관리자에 의해 변환될 수 있습니다.

관련 개념

JMS 메시지 변환 접근 방법

다수의 데이터 변환 접근 방법이 JMS 애플리케이션 디자이너에게 열려 있습니다. 이러한 접근 방법은 배타적이지 않습니다. 일부 애플리케이션은 이러한 접근 방법을 조합하여 사용할 수 있습니다. 애플리케이션이 텍스트만 교환 중이거나 메시지를 다른 JMS 애플리케이션과만 교환 중인 경우에는 일반적으로 데이터 변환을 고려하지 않습니다. 데이터 변환은 사용자를 위해 IBM MQ에서 자동으로 수행합니다.

JMS 클라이언트 메시지 변환 및 인코딩

JMS 클라이언트 메시지 변환 및 인코딩을 수행하는 데 사용되는 메소드가 각 변환 유형의 코드 예제와 함께 나열되어 있습니다.

730 페이지의 『데이터 변환 엑시트 호출』

데이터 변환 엑시트는 MQGET 호출을 처리하는 동안 제어를 수신하는 사용자 작성 엑시트입니다.

관련 태스크

비-JMS 애플리케이션에서 형식화된 레코드 교환

이 작업에서 제안된 단계를 따라 데이터 변환 출구(exit)를 설계 및 구축하고, 를 사용하여 비JMS 애플리케이션과 메시지를 교환할 JMSBytesMessage수 있는 클라이언트 JMS 애플리케이션을 개발하십시오. 비-JMS 애플리케이션과 형식화된 메시지의 교환은 데이터 변환 엑시트를 호출하거나 호출하지 않고도 발생할 수 있습니다.

관련 참조

JMS 메시지 유형 및 변환

메시지 유형의 선택은 메시지 변환에 대한 접근 방법에 영향을 줍니다. 메시지 변환 및 메시지 유형의 상호작용이 JMS 메시지 유형, JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessage 및 JMSBytesMessage에 대해 설명되어 있습니다.

비-JMS 애플리케이션에서 형식화된 레코드 교환

이 작업에서 제안된 단계를 따라 데이터 변환 출구(exit)를 설계 및 구축하고, 를 사용하여 비JMS 애플리케이션과 메시지를 교환할 JMSBytesMessage수 있는 클라이언트 JMS 애플리케이션을 개발하십시오. 비-JMS 애플리케이션과 형식화된 메시지의 교환은 데이터 변환 엑시트를 호출하거나 호출하지 않고도 발생할 수 있습니다.

시작하기 전에

비JMS 애플리케이션과 메시지를 교환하기 위한 더 간단한 솔루션을 JMSTextMessage 설계할 수 있을지도 모릅니다. 이 태스크의 단계를 따르기 전에 해당 가능성을 제거하십시오.

이 태스크 정보

다른 JMS 클라이언트와 교환된 JMS 메시지 형식화의 세부사항에 포함되지 않은 경우, JMS 클라이언트는 쓰기에 더 용이합니다. 메시지 유형이 JMSTextMessage, JMSMapMessage, JMSStreamMessage 또는 JMSObjectMessage인 경우, IBM MQ는 메시지 형식화의 세부사항을 관리합니다. IBM MQ는 다양한 플랫폼에서 숫자 인코딩 및 코드 페이지의 차이점을 처리합니다.

이러한 메시지 유형을 사용하여 비-JMS 애플리케이션과 메시지를 교환할 수 있습니다. 이를 수행하려면 IBM MQ classes for JMS에 의해 이러한 메시지가 구성되는 방법을 이해해야 합니다. 메시지를 해석할 수 있도록 비-JMS 애플리케이션을 수정할 수 있습니다. 145 페이지의 『JMS 메시지를 IBM MQ 메시지로 매핑』의 내용을 참조하십시오.

이러한 메시지 유형 중 하나를 사용하는 장점은 JMS 클라이언트 프로그래밍이 메시지가 교환되는 애플리케이션의 유형에 의존하지 않는다는 점입니다. 단점은 다른 프로그램을 수정해야 할 수 있으며 다른 프로그램의 변경이 불가능할 수 있다는 점입니다.

대체 접근 방법은 기존 메시지 형식을 처리할 수 있는 JMS 클라이언트 애플리케이션을 작성하는 것입니다. 종종 기존 메시지는 고정된 형식이며, 이에는 형식화되지 않은 데이터, 텍스트 및 숫자가 혼합되어 포함될 수 있습니다. 이 작업의 단계와 예제 JMS 클라이언트를 출발점으로 185 페이지의 『JMSBytesMessage에서 레코드 레이

아웃을 캡슐화하는 클래스 작성』 삼아, 비-JMS 애플리케이션과 형식화된 레코드를 교환할 수 있는 클라이언트를 JMS 구축하십시오.

프로시저

1. 레코드 레이아웃을 정의하거나 사전정의된 IBM MQ 헤더 클래스 중 하나를 사용하십시오.

사전정의된 IBM MQ 헤더의 핸들링은 [IBM MQ 메시지 헤더 핸들링](#)을 참조하십시오.

[183 페이지의 그림 37](#)는 데이터 변환 유틸리티에 의해 처리될 수 있는 사용자 정의된 고정 길이 레코드 레이아웃의 예제입니다.

2. 데이터 변환 엑시트를 작성하십시오.

[데이터 변환 엑시트 프로그램 작성](#)의 지시사항에 따라 데이터 변환 엑시트를 작성하십시오.

[185 페이지의 『JMSBytesMessage에서 레코드 레이아웃을 캡슐화하는 클래스 작성』](#)의 예제를 시도하려면 데이터 변환 엑시트의 이름을 MYRECORD로 지정하십시오.

3. 레코드 레이아웃 및 송신/수신 레코드를 캡슐화하는 Java 클래스를 작성하십시오. 두 가지 접근 방법을 수행할 수 있습니다.

- 레코드가 포함된 JMSBytesMessage를 읽고 쓰는 클래스를 작성하십시오. [185 페이지의 『JMSBytesMessage에서 레코드 레이아웃을 캡슐화하는 클래스 작성』](#)의 내용을 참조하십시오.
- 레코드의 데이터 구조를 정의하기 위해 `com.ibm.mq.header.Header`을(를) 확장하는 클래스를 쓰십시오. 새 헤더 유형에 대한 클래스 작성을 참조하십시오.

4. 메시지가 교환되는 코드화 문자 세트를 결정하십시오.

[메시지 변환에 대한 접근 방법 선택: 수신자 성공](#)을 참조하십시오.

5. JMS MQRFH2 헤더 없이 MQ-유형 메시지를 교환하기 위한 목적지를 구성하십시오.

송신 및 수신 목적지는 모두 MQ-유형 메시지를 교환하도록 구성되어야 합니다. 송신 및 수신 모두에 대해 동일한 목적지를 사용할 수 있습니다.

애플리케이션은 목적지 메시지 본문 특성을 대체할 수 있습니다.

```
((MQDestination)destination).setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

[185 페이지의 『JMSBytesMessage에서 레코드 레이아웃을 캡슐화하는 클래스 작성』](#)의 예제는 목적지 메시지 본문 특성을 대체하며, MQ-스타일 메시지의 송신을 보장합니다.

6. JMS 및 비-JMS 애플리케이션에서 솔루션을 테스트하십시오.

데이터 변환 엑시트를 테스트하는 유용한 도구는 다음과 같습니다.

- `amqsgetc0.c` 샘플 프로그램은 JMS 클라이언트가 송신한 메시지의 수신을 테스트하는 데 유용합니다. [184 페이지의 그림 38](#)에서 예제 헤더, `RECORD.h`을(를) 사용하기 위한 제안된 수정사항을 참조하십시오. 수정을 통해 `amqsgetc0.c`은(는) 예제 JMS 클라이언트가 전송한 메시지를 수신합니다 (`TryMyRecord.java`). [185 페이지의 『JMSBytesMessage에서 레코드 레이아웃을 캡슐화하는 클래스 작성』](#)의 내용을 참조하십시오.
- 샘플 IBM MQ 찾아보기 프로그램, `amqsbcg0.c`는 메시지 헤더, JMS 헤더, MQRFH2의 콘텐츠 및 메시지 컨테이너를 검사하는 데 유용합니다.
- `rfhutil` 프로그램(이전에 SupportPac IH03에서 사용 가능함)을 통해 테스트 메시지를 캡처하여 파일에 저장하고, 메시지 플로우를 구동하는 데 사용할 수 있습니다. 또한 출력 메시지를 다양한 형식으로 읽고 표시할 수 있습니다. 이때 형식으로는, 두 가지 유형의 XML과 COBOL 카피북에 일치하는 형식이 포함됩니다. 데이터는 EBCDIC 또는 ASCII일 수 있습니다. 메시지를 전송하기 전에 메시지에 RFH2 헤더를 추가할 수 있습니다.

수정된 `amqsgetc0.c` 샘플 프로그램을 사용하여 메시지 수신을 시도하며 이유 코드 2080으로 오류를 수신하는 경우에는 메시지에 MQRFH2가 있는지 여부를 확인하십시오. 수정사항은 MQRFH2가 지정되지 않은 목적으로 메시지가 송신되었다고 가정합니다.

```
struct RECORD { MQCHAR StrucID[4];  
                MQLONG Version;  
                MQLONG StructLength;  
                MQLONG Encoding;  
                MQLONG CodeCharSetId;  
                MQCHAR Format[8];  
                MQLONG Flags;  
                MQCHAR RecordData[32];  
            };
```

그림 37. *RECORD.h*

- RECORD.h 데이터 구조 선언

```
struct tagRECORD {
    MQCHAR4      StrucId;
    MQLONG       Version;
    MQLONG       StrucLength;
    MQLONG       Encoding;
    MQLONG       CCSID;
    MQCHAR8      Format;
    MQLONG       Flags;
    MQCHAR32     RecordData;
};
typedef struct tagRECORD RECORD;
typedef RECORD MQPOINTER PRECORD;
RECORD record;
PRECORD pRecord = &(record);
```

- 호출을 MQGET 수정하여 사용하십시오 RECORD,

1. 수정 전:

```
MQGET(Hcon,          /* connection handle */
      Hobj,          /* object handle */
      &md,           /* message descriptor */
      &gmo,          /* get message options */
      buflen,        /* buffer length */
      buffer,        /* message buffer */
      &messlen,       /* message length */
      &CompCode,      /* completion code */
      &Reason);      /* reason code */
```

2. 수정 후:

```
MQGET(Hcon,          /* connection handle */
      Hobj,          /* object handle */
      &md,           /* message descriptor */
      &gmo,          /* get message options */
      sizeof(RECORD), /* buffer length */
      pRecord,       /* message buffer */
      &messlen,       /* message length */
      &CompCode,      /* completion code */
      &Reason);      /* reason code */
```

- 인쇄문 변경

1. 다음을

```
buffer[messlen] = '\0';          /* add terminator */
printf("message <%s>\n", buffer);
```

2. 이를 다음 값으로 변경하십시오.

```
/* buffer[messlen] = '\0';          add terminator */
printf("ccsid <%d>, flags <%d>, message <%32.32s>\n \0",
      md.CodedCharSetId, record.Flags, record.RecordData);
```

그림 38. amqsget0.c 수정

관련 개념

JMS 메시지 변환 접근 방법

다수의 데이터 변환 접근 방법이 JMS 애플리케이션 디자이너에게 열려 있습니다. 이러한 접근 방법은 배타적이지 않습니다. 일부 애플리케이션은 이러한 접근 방법을 조합하여 사용할 수 있습니다. 애플리케이션이 텍스트만 교환 중이거나 메시지를 다른 JMS 애플리케이션과만 교환 중인 경우에는 일반적으로 데이터 변환을 고려하지 않습니다. 데이터 변환은 사용자를 위해 IBM MQ에서 자동으로 수행합니다.

JMS 클라이언트 메시지 변환 및 인코딩

JMS 클라이언트 메시지 변환 및 인코딩을 수행하는 데 사용되는 메소드가 각 변환 유형의 코드 예제와 함께 나열되어 있습니다.

큐 관리자 데이터 변환

JMS 클라이언트에서 메시지를 수신하는 비-JMS 애플리케이션은 큐 관리자 데이터 변환을 항상 사용할 수 있습니다. 메시지를 수신하는 JMS 클라이언트도 선택적인 큐 관리자 데이터 변환을 사용합니다.

변환-엑시트 코드 작성을 위한 유틸리티

관련 참조

JMS 메시지 유형 및 변환

메시지 유형의 선택은 메시지 변환에 대한 접근 방법에 영향을 줍니다. 메시지 변환 및 메시지 유형의 상호작용이 JMS 메시지 유형, JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessage 및 JMSBytesMessage에 대해 설명되어 있습니다.

*JMSBytesMessage*에서 레코드 레이아웃을 캡슐화하는 클래스 작성

이 태스크의 용도는 예를 들어 *JMSBytesMessage*의 고정 레코드 레이아웃 및 데이터 변환을 결합하는 방법을 살펴보는 것입니다. 작업에서, 예시 레코드 구조를 교환하기 위한 몇 가지 *JavaJMSBytesMessage* 클래스를 생성합니다. 예제를 수정하여 다른 레코드 구조를 교환하기 위한 클래스를 작성할 수 있습니다.

*JMSBytesMessage*는 비-JMS 프로그램과 혼합 데이터 유형 레코드를 교환하기 위한 최상의 JMS 메시지 유형 선택사항입니다. JMS 제공자에 의해 메시지 본문에 삽입된 추가 데이터는 없습니다. 따라서 이는 JMS 클라이언트 프로그램이 기존 IBM MQ 프로그램과 상호운영하는 경우에 사용할 최상의 메시지 유형 선택사항입니다. *JMSBytesMessage*의 사용에서 기본 걸림돌은 기타 프로그램에서 예상하는 인코딩 및 문자 세트의 일치입니다. 솔루션은 레코드를 캡슐화하는 클래스를 작성하는 것입니다. 특정 레코드 유형에 대해 *JMSBytesMessage* 읽기/쓰기를 캡슐화하는 클래스는 JMS 프로그램에서 고정 형식 레코드의 송신/수신을 보다 용이하게 합니다. 추상 클래스의 인터페이스의 일반 측면을 캡처하면 많은 솔루션이 서로 다른 레코드 형식에 대해 재사용될 수 있습니다. 서로 다른 레코드 형식은 추상 일반 클래스를 확장하는 클래스에서 구현될 수 있습니다.

다른 방법은 `com.ibm.mq.headers.Header` 클래스를 확장하는 것입니다. 이 `Header` 클래스는 기록 형식을 보다 선언적인 방식으로 구축하기 위한 메서드(예: `addMQLONG`)를 제공합니다. 클래스 `Header` 사용의 단점은 속성 획득 및 설정에 더 복잡한 해석적 인터페이스가 사용된다는 점이다. 두 접근 방법은 결과적으로 거의 동일한 양의 애플리케이션 코드를 생성합니다.

각 레코드가 동일한 형식, 코드화 문자 세트 및 인코딩을 사용하지 않는 한, *JMSBytesMessage*는 하나의 메시지에서 MQRFH2와 함께 단일 형식만 캡슐화할 수 있습니다. *JMSBytesMessage*의 형식, 인코딩 및 문자 세트는 MQRFH2 이후의 모든 메시지의 특성입니다. 예제는 *JMSBytesMessage*에 하나의 사용자 레코드만 포함된다는 가정 하에 작성되었습니다.

시작하기 전에

1. 스킬 레벨: 사용자는 Java 프로그래밍과 JMS에 대해 잘 알고 있어야 합니다. Java 개발 환경의 설정에 대한 지시사항은 제공되지 않습니다. *JMSTextMessage*, *JMSStreamMessage* 또는 *JMSMapMessage*를 교환하는 프로그램을 작성해 두는 것이 좋습니다. 그러면 *JMSBytesMessage*를 사용한 메시지 교환의 차이점을 파악할 수 있습니다.
2. 예제에서는 IBM WebSphere MQ 7.0이 필요합니다.
3. 예제는 Eclipse 워크bench의 Java 퍼스펙티브를 사용하여 작성되었습니다. 6.0 이상이 JRE 필요합니다. 탐색기에서 IBM MQ 제공하는 Java 환경을 활용하여 클래스를 Java 개발하고 실행할 수 있습니다. 또는 자체 Java 개발 환경을 사용합니다.
4. IBM MQ Explorer를 사용하면 명령행 유틸리티를 사용할 때보다 테스트 환경 설정 및 디버깅이 보다 단순해 집니다.

이 태스크 정보

사용자는 두 개의 클래스(RECORD 및 MyRecord)를 작성하는 방법에 대해 안내를 받습니다. 이 두 클래스는 함께 고정 형식 레코드를 캡슐화합니다. 여기에는 속성을 가져오고 설정하는 메소드가 있습니다. Get 메소드는 *JMSBytesMessage*에서 레코드를 읽으며, Put 메소드는 레코드를 *JMSBytesMessage*에 씁니다.

이 태스크의 용도는 재사용 가능한 프로덕션 품질 클래스를 작성하는 것이 아닙니다. 자체 클래스에서 시작하도록 태스크의 예제 사용을 선택할 수 있습니다. 이 태스크의 용도는 *JMSBytesMessage*를 사용할 때 사용자에게 안내 참고사항(주로 문자 세트, 형식 및 인코딩의 사용에 대한)을 제공하는 것입니다. 클래스 작성의 각 단계가 기술되며, 종종 간과되는 *JMSBytesMessage* 사용의 측면이 설명됩니다.

RECORD 클래스는 추상 클래스이며, 사용자 레코드에 대한 일부 공용 필드를 정의합니다. 공용 필드는 아이 캐처, 버전 및 길이 필드를 지닌 표준 IBM MQ 헤더 레이아웃에서 모델화됩니다. 많은 IBM MQ 헤더에서 발견되는 인코딩, 문자 세트 및 형식 필드는 생략됩니다. 다른 헤더는 사용자 정의된 형식을 따를 수 없습니다. RECORD 클래스를 확장하는 MyRecord 클래스는 추가 사용자 필드로 레코드를 실제로 확장함으로써 이를 수행합니다. 클래스에 의해 작성된 JMSBytesMessage는 큐 관리자 데이터 변환 엑시트에 의해 처리될 수 있습니다.

191 페이지의 『예제 실행에 사용된 클래스』에는 RECORD 및 MyRecord의 전체 목록이 포함되어 있습니다. 여기에는 RECORD and MyRecord를 테스트하기 위한 추가 "scaffolding" 클래스의 목록도 포함됩니다. 추가 클래스는 다음과 같습니다.

TryMyRecord

RECORD 및 MyRecord를 테스트하는 기본 프로그램입니다.

EndPoint

단일 클래스에 JMS 연결, 목적지 및 세션을 캡슐화하는 추상 클래스입니다. 해당 인터페이스는 단지 RECORD 및 MyRecord 클래스의 테스트 요구사항을 충족합니다. 이는 JMS 애플리케이션의 작성을 위해 설정된 디자인 패턴이 아닙니다.

참고: 클래스는 Endpoint 대상 생성 후 다음 코드 줄을 포함합니다:

```
((MQDestination)destination).setReceiveConversion
(WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

V7.0의 V7.0.1.5 이상에서는 큐 관리자 변환을 켜야 합니다. 이는 기본적으로 사용되지 않습니다. V7.0의 V7.0.1.4까지에서는 큐 관리자 변환이 기본적으로 사용되며, 다음의 코드 행은 오류의 원인이 됩니다.

MyProducer 및 MyConsumer

EndPoint를 확장하고 MessageConsumer 및 MessageProducer(연결되어 요청을 수락할 준비가 된)를 작성하는 클래스입니다.

모든 클래스는 함께 JMSBytesMessage에서 데이터 변환을 사용하는 방법을 이해하기 위해 빌드하고 시험할 수 있는 전체 애플리케이션을 구성합니다.

프로시저

1. 기본 구성자로 IBM MQ 헤더에서 표준 필드를 캡슐화하는 추상 클래스를 작성하십시오. 나중에 사용자는 클래스를 확장하여 요구사항에 맞게 헤더를 조정합니다.

```
public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }
}
```

참고:

- a. structID에서 nextFormat의 속성은 표준 IBM MQ 메시지 헤더에 레이아웃되는 순서로 나열됩니다.
- b. format, messageEncoding 및 messageCharset 속성은 헤더 자체를 설명하며 헤더의 일부가 아닙니다.
- c. 사용자는 레코드의 문자 세트 또는 코드화 문자 세트 ID를 저장하는지 여부를 결정해야 합니다. Java는 문자 세트를 사용하며, IBM MQ 메시지는 코드화 문자 세트 ID를 사용합니다. 예제 코드는 문자 세트를 사용합니다.

- d. int 에 MQLONG 의해 IBM MQ시리얼화됩니다. MQLONG은 4바이트입니다.
2. 개인용 속성에 대해 Getter 및 Setter를 작성하십시오.
- a) Getter 작성 또는 생성:

```
public String getHeaderFormat() { return headerFormat; }
public int getHeaderEncoding() { return headerEncoding; }
public String getMessageCharset() { return headerCharset; }
public int getMessageEncoding() { return headerEncoding; }
public String getStructID() { return structID; }
public int getStructLength() { return structLength; }
public int getVersion() { return version; }
```

- b) Setter 작성 또는 생성:

```
public void setHeaderCharset(String charset) {
    this.headerCharset = charset; }
public void setHeaderEncoding(int encoding) {
    this.headerEncoding = encoding; }
public void setHeaderFormat(String headerFormat) {
    this.headerFormat = headerFormat; }
public void setStructID(String structID) {
    this.structID = structID; }
public void setStructLength(int structLength) {
    this.structLength = structLength; }
public void setVersion(int version) {
    this.version = version; }
}
```

3. JMSBytesMessage에서 RECORD 인스턴스를 작성하기 위한 구성자를 작성하십시오.

```
public RECORD(BytesMessage message) throws JMSException, IOException,
    MQDataException {
    super();
    setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
    setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
    byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
    message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
    setStructID(new String(structID, getMessageCharset()));
    setVersion(message.readInt());
    setStructLength(message.readInt());
}
```

참고:

- 목적지에 대해 설정된 값을 대체하므로, messageCharset 및 messageEncoding은 메시지 특성에서 캡처됩니다. format은 업데이트되지 않습니다. 예제에서는 오류 검사를 수행하지 않습니다. Record(BytesMessage) 구성자가 호출되는 경우, JMSBytesMessage가 RECORD 유형 메시지라고 가정됩니다. "setStructID(new String(structID, getMessageCharset()))" 행은 아이 캐처를 설정합니다.
 - 메소드를 완료하는 코드 행은 RECORD 인스턴스에 설정된 기본값을 업데이트하여 메시지의 필드를 순서대로 직렬화 해제합니다.
4. 헤더 필드를 JMSBytesMessage에 쓰는 Put 메소드를 작성하십시오.

```
protected BytesMessage put(MyProducer myProducer) throws IOException,
    JMSException, UnsupportedEncodingException {
    setHeaderEncoding(myProducer.getEncoding());
    setHeaderCharset(myProducer.getCharset());
    myProducer.setMQClient(true);
    BytesMessage bytes = myProducer.session.createBytesMessage();
    bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
    bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
    bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
        myProducer.getCCSID());
    bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + " "
        + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
        .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
    bytes.writeInt(getVersion());
    bytes.writeInt(getStructLength());
    return bytes;
}
```

```
}
```

참고:

- a. MyProducer는 JMS Connection, Destination, Session 및 MessageProducer를 단일 클래스에 캡슐화합니다. 나중에 사용되는 MyConsumer는 JMS Connection, Destination, Session 및 MessageConsumer를 단일 클래스에 캡슐화합니다.
- b. JMSBytesMessage의 경우, 인코딩이 Native가 아니면 인코딩을 메시지에 설정해야 합니다. 목적지 인코딩은 메시지 인코딩 속성, JMS_IBM_CHARACTER_SET에 복사되며 RECORD 클래스의 속성으로 저장됩니다.

i) "setMessageEncoding(myProducer.getEncoding());"은 "((MQDestination) destination).getIntProperty(WMQConstants.WMQ_ENCODING));"을 호출하여 목적지 인코딩을 가져옵니다.

ii) "Bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getMessageEncoding());"은 메시지 인코딩을 설정합니다.

- c. 텍스트를 바이트로 변환하는 데 사용되는 문자 세트는 목적지로부터 확보되며, RECORD 클래스의 속성으로 저장됩니다. 메시지에 설정되어 있지 않습니다. 왜냐하면 작성 시 IBM MQ classes for JMS 사용되지 JMSBytesMessage 않기 때문입니다.

"messageCharset = myProducer.getCharset();"는 다음을 호출합니다.

```
public String getCharset() throws UnsupportedOperationException,
    JMSEException {
    return CCSID.getCodepage(getCCSID());
}
```

이는 코드화 문자 세트 ID에서 Java 문자 세트를 가져옵니다.

"CCSID.getCodepage(ccsid)"는 패키지 com.ibm.mq.headers에 있습니다. ccsid는 목적지를 조회하는 MyProducer의 다른 메소드에서 확보됩니다.

```
public int getCCSID() throws JMSEException {
    return (((MQDestination) destination)
        .getIntProperty(WMQConstants.WMQ_CCID));
}
```

- d. "myProducer.setMQClient(true);"는 클라이언트 유형의 목적지 설정을 대체하며, 이를 IBM MQ MQI client로 강제 실행합니다. 관리 구성 오류가 모호해지므로, 이 코드 행을 생략하고자 할 수 있습니다.

"myProducer.setMQClient(true);"는 다음을 호출합니다.

```
((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ); }
if (!getMQDest()) setMQBody();
```

해당 코드는 설정을 재정의해야 하는 경우 본문 IBM MQ 스타일을 지정되지 JMS 애플리케이션으로 설정하는 부작용이 있습니다.

참고:

IBM MQ classes for JMS는 메시지의 형식, 인코딩 및 문자 세트 ID를 메시지 디스크립터, MQMD에 또는 JMS 헤더, MQRFH2에 씁니다. 이는 메시지에 IBM MQ 스타일 본문이 있는지 여부에 달려 있습니다. MQMD 필드를 수동으로 설정하지 마십시오.

메소드는 메시지 디스크립터 메시지 특성을 수동으로 설정하기 위해 존재합니다. 이는 JMS_IBM_MQMD_* 특성을 사용합니다. JMS_IBM_MQMD_* 특성을 설정하려면 목적지 특성, WMQ_MQMD_WRITE_ENABLED를 설정해야 합니다.

```
((MQDestination)destination).setMQMDWriteEnabled(true);
```

특성을 읽으려면 목적지 특성, WMQ_MQMD_READ_ENABLED를 설정해야 합니다.

전체 메시지 페이로드에 대한 전체 제어가 가능한 경우에만 JMS_IBM_MQMD_*를 사용하십시오.

JMS_IBM_* 특성과는 달리, JMS_IBM_MQMD_* 특성은 IBM MQ classes for JMS가 JMS 메시지를 구성하는 방법을 제어하지 않습니다. JMS 메시지의 특성과 충돌하는 메시지 디스크립터 특성을 작성할 수 있습니다.

- e. 메소드를 완료하는 코드 행은 메시지의 필드로서 클래스의 속성을 직렬화합니다.

문자열 속성은 공백으로 채워집니다. 문자열은 레코드에 대해 정의된 문자 세트를 사용하여 바이트로 변환되며, 메시지 필드의 길이로 잘립니다.

5. 가져오기를 추가하여 클래스를 완료하십시오.

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSEException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
```

6. 추가 필드를 포함하도록 RECORD 클래스를 확장하는 클래스를 작성하십시오. 기본 구성자를 포함하십시오.

```
public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
            + DATA_LENGTH);
    }
}
```

참고:

- a. RECORD 서브클래스, MyRecord는 헤더의 길이 및 아이 캐처, 형식을 사용자 정의합니다.

7. Getter 또는 Setter를 작성하거나 생성하십시오.

- a) Getter 작성:

```
public int getFlags() { return flags; }
public String getRecordData() { return recordData; } .
```

- b) Setter 작성:

```
public void setFlags(int flags) {
    this.flags = flags; }
public void setRecordData(String recordData) {
    this.recordData = recordData; }
}
```

8. JMSBytesMessage에서 MyRecord 인스턴스를 작성하기 위한 구성자를 작성하십시오.

```
public MyRecord(BytesMessage message) throws JMSEException, IOException,
    MQDataException {
    super(message);
    setFlags(message.readInt());
    byte[] recordData = new byte[DATA_LENGTH];
    message.readBytes(recordData, DATA_LENGTH);
    setRecordData(new String(recordData, super.getMessageCharset()));
}
```

참고:

- a. 표준 메시지 템플릿을 구성하는 필드는 우선 RECORD 클래스에서 읽어옵니다.
 - b. 텍스트는 recordData 메시지의 문자 집합 속성을 사용하여 String 변환됩니다.
9. 이용자로부터 메시지를 가져오고 새 MyRecord 인스턴스를 작성하는 정적 메소드를 작성하십시오.

```
public static MyRecord get(MyConsumer myConsumer) throws JMSException,
    MQDataException, IOException {
    BytesMessage message = (BytesMessage) myConsumer.receive();
    return new MyRecord(message);
}
```

참고:

- a. 예제에서는 단순화를 위해 MyRecord(BytesMessage) 구성자가 정적 Get 메소드에서 호출됩니다. 일반적으로, 메시지 수신을 새 MyRecord 인스턴스 작성과 분리할 수 있습니다.
10. 고객 필드를 메시지 헤더가 포함된 JMSBytesMessage에 추가하는 Put 메소드를 작성하십시오.

```
public BytesMessage put(MyProducer myProducer) throws JMSException,
    IOException {
    BytesMessage bytes = super.put(myProducer);
    bytes.writeInt(getFlags());
    bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
        + DATA_LENGTH + "s", getRecordData())
        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
    myProducer.send(bytes);
    return bytes;
}
```

참고:

- a. 코드의 메소드 호출은 메시지의 필드로서 MyRecord 클래스의 속성을 직렬화합니다.
 - 속성은 recordDataString 공백으로 채워진 후 레코드에 정의된 문자 집합을 사용하여 바이트로 변환되고, 필드의 RecordData 길이로 잘립니다.
11. 포함 명령문을 추가하여 클래스를 완료하십시오.

```
package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
```

결과

• TryMyRecord 클래스 실행의 결과:

- 코드화 문자 세트 37의 메시지를 송신하고 큐 관리자 변환 엑시트 사용:

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 273 CCSID UTF-8
```

- 코드화 문자 세트 37의 메시지를 송신하고 큐 관리자 변환 엑시트 사용 안함:

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID IBM037
```

- 메시지를 수신하지 않도록 TryMyRecord 클래스를 수정하고, 그 대신 수정된 amqsget0.c 샘플을 사용하여 이를 수신한 결과입니다. 수정된 샘플은 포맷된 레코드를 수락합니다; 참조하십시오 [181 페이지의 『비-JMS 애플리케이션에서 형식화된 레코드 교환』](#) [184 페이지의 그림 38](#).
- 코드화 문자 세트 37의 메시지를 송신하고 큐 관리자 변환 엑시트 사용:


```
Sample AMQSGETO start
ccsid <850>, flags <1>, message <ABCDEFGHIJKLMNOPQRSTUVWXYZ012345>
no more messages
Sample AMQSGETO end
```

- 코드화 문자 세트 37의 메시지를 송신하고 큐 관리자 변환 엑시트 사용 안함:

```
Sample AMQSGETO start
MQGET ended with reason code 2110
ccsid <37>, flags <1>, message <--+--ãÃ++ðÊÊÊiðÎð+ôôôöµþÞÚ-±=¼¶§>
no more messages
Sample AMQSGETO end
```

예제를 시도하고 상이한 코드 페이지 및 데이터 변환 엑시트로 시험합니다. Java 클래스를 작성하고, IBM MQ를 구성한 후에 기본 프로그램 TryMyRecord를 실행하십시오. [191 페이지의 『#unique_199/unique_199_Connect_42_Try』](#)의 내용을 참조하십시오.

1. 예제를 실행할 수 있도록 IBM MQ 및 JMS를 구성하십시오. 지시사항은 Windows에서 예제를 실행하기 위한 것입니다.

- a. 큐 관리자 작성

```
crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
strmqm QM1
```

- b. 큐 작성

```
echo DEFINE QL('Q1') REPLACE | runmqsc QM1
```

- c. JNDI 디렉토리 작성

```
cd c:\
md JNDI-Directory
```

- d. JMS bin 디렉토리로 전환

JMS 관리 프로그램을 여기서 실행해야 합니다. 경로는 `MQ_INSTALLATION_PATH\java\bin`입니다.

- e. JMSQM1Q1.txt(이)라는 파일에 다음 JMS 목적지 작성

```
DEF CF(QM1) PROVIDERVERSION(7) QMANAGER(QM1)
DEF Q(Q1) CCSID(37) ENCODING(RRR) MSGBODY(MQ) QMANAGER(QM1) QUEUE(Q1) TARGCLIENT(MQ)
VERSION(7)
END
```

- f. JMSAdmin 프로그램을 실행하여 JMS 자원 작성

```
JMSAdmin < JMSQM1Q1.txt
```

2. IBM MQ Explorer를 사용하여 작성한 정의를 작성, 대체하고 찾아볼 수 있습니다.
3. TryMyRecord을(를) 실행하십시오.

예제 실행에 사용된 클래스

다음 코드 블록에 나열된 클래스들은 압축 파일로도 제공됩니다. 다운로드하거나 [jm25529.zip](#) [jm25529.tar.gz](#).

TryMyRecord

```
package com.ibm.mq.id;
public class TryMyRecord {
    public static void main(String[] args) throws Exception {
        MyProducer producer = new MyProducer();
        MyRecord outrec = new MyRecord();
        System.out.println("Out flags " + outrec.getFlags() + " text "
            + outrec.getRecordData() + " Encoding "
            + producer.getEncoding() + " CCSID " + producer.getCCSID()
            + " MQ " + producer.getMQDest());
        outrec.put(producer);
        System.out.println("Out flags " + outrec.getFlags() + " text "
```

```

        + outrec.getRecordData() + " Encoding "
        + producer.getEncoding() + " CCSID " + producer.getCCSID()
        + " MQ " + producer.getMQDest());
MyRecord inrec = MyRecord.get(new MyConsumer());
System.out.println("In flags " + inrec.getFlags() + " text "
        + inrec.getRecordData() + " Encoding "
        + inrec.getMessageEncoding() + " CCSID "
        + inrec.getMessageCharset());
    }
}

```

RECORD

JM 3.0

```

package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSEException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;

public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }

    public RECORD(BytesMessage message) throws JMSEException, IOException,
        MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding; }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }

    protected BytesMessage put(MyProducer myProducer) throws IOException,
        JMSEException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
            myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$s-" + RECORD_STRUCT_ID_LENGTH + "-"
            + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
            .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }
}

```

```

    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}

```

JMS 2.0

```

package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import javax.jms.BytesMessage;
import javax.jms.JMSEException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }
    public RECORD(BytesMessage message) throws JMSEException, IOException,
        MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding; }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }

    protected BytesMessage put(MyProducer myProducer) throws IOException,
        JMSEException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
            myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "s",
            getStructID()).getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }
}

```

```

    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}

```

MyRecord

```


package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSEException;
import com.ibm.mq.headers.MQDataException;

public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
            + DATA_LENGTH);
    }

    public MyRecord(BytesMessage message) throws JMSEException, IOException,
        MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }

    public static MyRecord get(MyConsumer myConsumer) throws JMSEException,
        MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }

    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; }

    public BytesMessage put(MyProducer myProducer) throws JMSEException,
        IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + " "
            + DATA_LENGTH + "s", getRecordData())
            .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }

    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}

```

> JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSEException;
import com.ibm.mq.headers.MQDataException;
public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
            + DATA_LENGTH);
    }
    public MyRecord(BytesMessage message) throws JMSEException, IOException,
        MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }
    public static MyRecord get(MyConsumer myConsumer) throws JMSEException,
        MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }
    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; }

    public BytesMessage put(MyProducer myProducer) throws JMSEException,
        IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + ". "
            + DATA_LENGTH + "s", getRecordData())
            .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }
    public void setFlags(int flags) {
        this.flags = flags;
    }
    public void setRecordData(String recordData) {
        this.recordData = recordData;
    }
}
```

EndPoint

> JMS 3.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import jakarta.jms.Connection;
import jakarta.jms.ConnectionFactory;
import jakarta.jms.Destination;
import jakarta.jms.JMSEException;
import jakarta.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSEException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
    }
}
```

```

        System.setProperty("java.naming.factory.initial",
            "com.sun.jndi.fscontext.ReifFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
            (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
protected EndPoint(String cFactory, String dest) throws NamingException,
    JMSEException {
    System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
    System.setProperty("java.naming.factory.initial",
        "com.sun.jndi.fscontext.ReifFSContextFactory");
    ctx = new InitialContext();
    cf = (ConnectionFactory) ctx.lookup(cFactory);
    connection = cf.createConnection();
    destination = (Destination) ctx.lookup(dest);
    ((MQDestination)destination).setReceiveConversion
        (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
    session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
public int getCCSID() throws JMSEException {
    return (((MQDestination) destination)
        .getIntProperty(WMQConstants.WMQ_CCSID)); }
public String getCharset() throws UnsupportedEncodingException,
    JMSEException {
    return CCSID.getCodepage(getCCSID()); }
public int getEncoding() throws JMSEException {
    return (((MQDestination) destination)
        .getIntProperty(WMQConstants.WMQ_ENCODING)); }
public boolean getMQDest() throws JMSEException {
    if (((((MQDestination) destination).getMessageBodyStyle()
        == WMQConstants.WMQ_MESSAGE_BODY_MQ)
        || (((MQDestination) destination).getMessageBodyStyle()
        == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
        && (((MQDestination) destination).getTargetClient()
        == WMQConstants.WMQ_TARGET_DEST_MQ)))
        return true;
    else
        return false; }
public void setCCSID(int ccsid) throws JMSEException {
    ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
        ccsid); }
public void setEncoding(int encoding) throws JMSEException {
    ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
        encoding); }
public void setMQBody() throws JMSEException {
    ((MQDestination) destination)
        .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED); }
public void setMQBody(boolean mqbody) throws JMSEException {
    if (mqbody) ((MQDestination) destination)
        .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
    else
        ((MQDestination) destination)
        .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS); }
public void setMQClient(boolean mqclient) throws JMSEException {
    if (mqclient){
        ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
        if (!getMQDest()) setMQBody();
    }
    else
        ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
}

```

JMS 2.0

```

package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSEException;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
}

```



```

public Connection connection;
public Destination destination;
public Session session;
protected EndPoint() throws NamingException, JMSEException {
    System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
    System.setProperty("java.naming.factory.initial",
        "com.sun.jndi.fscontext.RefFSContextFactory");
    ctx = new InitialContext();
    cf = (ConnectionFactory) ctx.lookup("QM1");
    connection = cf.createConnection();
    destination = (Destination) ctx.lookup("Q1");
    ((MQDestination)destination).setReceiveConversion
        (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
    session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
protected EndPoint(String cFactory, String dest) throws NamingException,
    JMSEException {
    System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
    System.setProperty("java.naming.factory.initial",
        "com.sun.jndi.fscontext.RefFSContextFactory");
    ctx = new InitialContext();
    cf = (ConnectionFactory) ctx.lookup(cFactory);
    connection = cf.createConnection();
    destination = (Destination) ctx.lookup(dest);
    ((MQDestination)destination).setReceiveConversion
        (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
    session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
public int getCCSID() throws JMSEException {
    return (((MQDestination) destination)
        .getIntProperty(WMQConstants.WMQ_CCSID)); }
public String getCharset() throws UnsupportedEncodingException,
    JMSEException {
    return CCSID.getCodepage(getCCSID()); }
public int getEncoding() throws JMSEException {
    return (((MQDestination) destination)
        .getIntProperty(WMQConstants.WMQ_ENCODING)); }
public boolean getMQDest() throws JMSEException {
    if (((MQDestination) destination).getMessageBodyStyle()
        == WMQConstants.WMQ_MESSAGE_BODY_MQ)
        || (((MQDestination) destination).getMessageBodyStyle()
            == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
            && (((MQDestination) destination).getTargetClient()
                == WMQConstants.WMQ_TARGET_DEST_MQ))
        return true;
    else
        return false; }
public void setCCSID(int ccid) throws JMSEException {
    ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
        ccid); }
public void setEncoding(int encoding) throws JMSEException {
    ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
        encoding); }
public void setMQBody() throws JMSEException {
    ((MQDestination) destination)
        .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED); }
public void setMQBody(boolean mqbody) throws JMSEException {
    if (mqbody) ((MQDestination) destination)
        .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
    else
        ((MQDestination) destination)
        .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS); }
public void setMQClient(boolean mqclient) throws JMSEException {
    if (mqclient){
        ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
        if (!getMQDest()) setMQBody();
    }
    else
        ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
}

```

MyProducer

JM 3.0

```

package com.ibm.mq.id;
import jakarta.jms.JMSEException;
import jakarta.jms.Message;
import jakarta.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSEException {
        super();
    }
}

```

```

        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
        JMSEException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSEException {
        producer.send(message); }
}

```

JMS 2.0

```

package com.ibm.mq.id;
import javax.jms.JMSEException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends Endpoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSEException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
        JMSEException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSEException {
        producer.send(message); }
}

```

MyConsumer

JM 3.0

```

package com.ibm.mq.id;
import jakarta.jms.JMSEException;
import jakarta.jms.Message;
import jakarta.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends Endpoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSEException {
        super();
        consumer = session.createConsumer(destination);
        connection.start(); }
    public MyConsumer(String cFactory, String dest) throws NamingException,
        JMSEException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start(); }
    public Message receive() throws JMSEException {
        return consumer.receive(); }
}

```

JMS 2.0

```

package com.ibm.mq.id;
import javax.jms.JMSEException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends Endpoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSEException {
        super();
        consumer = session.createConsumer(destination);
        connection.start(); }
    public MyConsumer(String cFactory, String dest) throws NamingException,
        JMSEException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start(); }
    public Message receive() throws JMSEException {
        return consumer.receive(); }
}

```

연결 팩토리 및 목적지 작성 및 구성

애플리케이션은 IBM MQ classes for JMS/IBM MQ classes for Jakarta Messaging JNDI(Naming and Directory Java Interface) 네임스페이스에서 관리 객체로 검색하거나, JAR 확장 또는 JAR 확장 기능을 IBMJMS/IBM MQJMS 사용하여 연결 팩토리와 목적지를 생성할 수 있습니다. 또한 애플리케이션은 IBM JMS 확장 또는 IBM MQ JMS 확장을 사용하여 연결 팩토리 및 목적지의 특성을 설정할 수도 있습니다.

연결 팩토리와 목적지는 애플리케이션의 JMS/Jakarta Messaging 논리 흐름에서 시작점 역할을 합니다. 애플리케이션은 ConnectionFactory 오브젝트를 사용하여 메시징 서버에 대한 연결을 작성하며, 큐 또는 토픽 오브젝트를 메시지가 송신되는 대상으로서 또는 메시지가 수신되는 소스로서 사용합니다. 따라서 애플리케이션은 최소한 하나의 연결 팩토리와 하나 이상의 목적지를 작성해야 합니다. 연결 팩토리 또는 목적지가 작성된 경우, 애플리케이션은 하나 이상의 자체 특성을 설정하여 오브젝트를 구성해야 할 수 있습니다.

요약하면, 애플리케이션은 다음과 같은 방법으로 연결 팩토리 및 목적지를 작성하고 구성할 수 있습니다.

JNDI를 사용하여 관리 대상 오브젝트 검색

관리자는 관리 IBM MQJMS 도구를 사용하여 JMS 및 Jakarta Messaging 객체를 구성하는 방법에 설명된 대로, IBM MQ Explorer 또는 JMS 2.0 객체를 관리 탐색기 (IBM MQ Explorer)를 사용하여 구성하는 방법에 설명된 대로, JNDI 네임스페이스 내에서 관리 객체로서 연결 팩토리 및 목적지를 생성하고 구성할 수 있습니다. 그리고 애플리케이션은 JNDI 네임스페이스에서 관리 대상 오브젝트를 검색할 수 있습니다. 관리 대상 오브젝트가 검색된 경우, 애플리케이션은 필요하면 IBM JMS 확장 또는 IBM MQ JMS 확장을 사용하여 하나 이상의 해당 특성을 설정하거나 변경할 수 있습니다.

참고: **JMS 3.0** IBM MQ Explorer/Jakarta Messaging 3.0의 경우, 를 사용하여 JNDI를 관리할 수 없습니다. **JMS30Admin**JNDI 관리 기능은 의 **JMSAdmin** 변형인 Jakarta Messaging 3.0 에서 지원됩니다.

IBM JMS 확장 사용

애플리케이션은 IBM JMS 확장을 사용하여 런타임에 동적으로 연결 팩토리 및 목적지를 작성할 수 있습니다. 애플리케이션은 우선 JmsFactoryFactory 오브젝트를 작성한 후에 이 오브젝트의 메소드를 사용하여 연결 팩토리 및 목적지를 작성합니다. 연결 팩토리 또는 목적지를 작성한 경우, 애플리케이션은 JmsPropertyContext 인터페이스에서 상속한 메소드를 사용하여 해당 특성을 설정할 수 있습니다. 또는 애플리케이션은 URI(Uniform Resource Identifier)를 사용하여 목적지를 작성할 때 목적지의 하나 이상의 특성을 지정할 수 있습니다.

IBM MQ JMS 확장 사용

애플리케이션은 IBM MQ JMS 확장을 사용하여 런타임에 동적으로 연결 팩토리 및 목적지를 작성할 수도 있습니다. 애플리케이션은 제공된 구성자를 사용하여 연결 팩토리 및 목적지를 작성합니다. 연결 팩토리 또는 목적지를 작성한 경우, 애플리케이션은 오브젝트의 메소드를 사용하여 해당 특성을 설정할 수 있습니다. 또는 애플리케이션은 URI를 사용하여 목적지를 작성할 때 목적지의 하나 이상의 특성을 지정할 수 있습니다.

관련 태스크

JMS 및 Jakarta Messaging 리소스 구성

JNDI를 사용하여 또는 *Jakarta Messaging JMS* 애플리케이션에서 관리되는 객체를 검색하기
명명 및 디렉터리 인터페이스 Java (JNDI) 네임스페이스에서 관리되는 객체를 검색하려면 애플리케이션은 JMS/Jakarta Messaging 초기 컨텍스트를 생성한 후 lookup() 메서드를 사용하여 객체를 검색해야 합니다.

애플리케이션이 JNDI 네임스페이스에서 관리 대상 오브젝트를 검색할 수 있으려면 우선 관리자가 관리 대상 오브젝트를 작성해야 합니다.

JMS 2.0 관리자는 JNDI 네임스페이스 내에서 관리 대상 객체를 생성하고 유지 관리하기 위해 JMS 2.0 관리 IBM MQ Explorer 도구, **JMSAdmin** IBM MQJMS 또는 를 사용할 수 있습니다. 자세한 정보는 JNDI 네임스페이스에서 연결 팩토리 및 목적지 구성을 참조하십시오.

JMS 3.0 IBM MQ Explorer/Jakarta Messaging 3.0의 경우, 를 사용하여 JNDI를 관리할 수 없습니다. **JMS30Admin**JNDI 관리 기능은 의 **JMSAdmin** 변형인 Jakarta Messaging 3.0 에서 지원됩니다.

애플리케이션 서버는 일반적으로 관리 대상 오브젝트에 대한 자체 저장소 및 오브젝트 작성과 유지보수를 위한 자체 도구를 제공합니다.

JNDI 네임스페이스에서 관리 대상 오브젝트를 검색하려면, 애플리케이션이 다음 예제에서 표시된 대로 우선 초기 컨텍스트를 작성해야 합니다.

> JM 3.0

```
import jakarta.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

> JMS 2.0

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

이 코드에서 문자열 변수 `url` 및 `icf`의 의미는 다음과 같습니다.

URL

디렉토리 서비스의 URL(Uniform Resource Locator)입니다. URL의 형식은 다음 중 하나일 수 있습니다.

- `ldap://hostname/contextName` - LDAP 서버를 기반으로 하는 디렉토리 서비스의 경우
- `file://directoryPath` - 로컬 파일 시스템을 기반으로 하는 디렉토리 서비스의 경우

icf

초기 컨텍스트 팩토리의 클래스 이름입니다. 이는 다음 값 중 하나일 수 있습니다.

- `com.sun.jndi.ldap.LdapCtxFactory` - LDAP 서버를 기반으로 하는 디렉토리 서비스의 경우
- `com.sun.jndi.fscontext.RefFSContextFactory` - 로컬 파일 시스템을 기반으로 하는 디렉토리 서비스의 경우

참고로, JNDI 패키지 및 LDAP(Lightweight Directory Access Protocol) 서비스 제공자의 일부 조합은 LDAP 오류 84가 발생하는 원인이 될 수 있습니다. 이 문제점을 해결하려면 `InitialDirContext()`를 호출하기 전에 다음 코드 행을 삽입하십시오.

```
environment.put(Context.REFERRAL, "throw");
```

초기 컨텍스트를 확보한 후에, 애플리케이션은 다음 예제에서 표시된 대로 `lookup()` 메소드를 사용하여 JNDI 네임스페이스에서 관리 대상 오브젝트를 검색할 수 있습니다.

```
ConnectionFactory factory;
Queue queue;
Topic topic;
.
.
factory = (ConnectionFactory)ctx.lookup("cn=myCF");
queue = (Queue)ctx.lookup("cn=myQ");
topic = (Topic)ctx.lookup("cn=myT");
```

이 코드는 LDAP 기반 네임스페이스에서 다음 오브젝트를 검색합니다.

- 이름 myCF로 바인딩된 ConnectionFactory 오브젝트
- 이름 myQ로 바인딩된 큐 오브젝트
- 이름 myT로 바인딩된 토픽 오브젝트

JNDI 사용에 대한 자세한 정보는 Oracle Corporation에서 제공하는 JNDI 문서를 참조하십시오.

관련 태스크

[IBM MQ Explorer를 사용한 JMS 2.0 객체 구성](#)

[관리 도구를 사용하여 JMS 및 Jakarta Messaging 객체 구성](#)

[WebSphere Application Server 에서 JMS 2.0 리소스 구성](#)

IBM JMS 확장 사용

IBM MQ classes for JMS (JMS 2.0)와 IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0)는 각 각 API에 JMS 대한 기능적으로 동일한 확장 집합인 확장 기능을 IBMJMS 포함합니다. 애플리케이션은 이러한 확장 기능을 사용하여 런타임에 동적으로 연결 팩토리 및 대상(destination)을 생성하고, 또는 IBM MQ classes for Jakarta Messaging 객체의 IBM MQ classes for JMS 속성을 설정할 수 있습니다. 확장은 메시징 제공자에서 사용될 수 있습니다.

IBM JMS 확장은 다음 패키지의 인터페이스 및 클래스 세트입니다.

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

Jakarta Messaging 3.0 이러한 패키지들은 에 있습니다 com.ibm.jakarta.client.jar.

JMS 2.0 이러한 패키지는 또는 com.ibm.mqjms.jar 에 있습니다 JMS 2.0com.ibm.mq.allclient.jar.

이러한 확장은 다음 기능을 제공합니다.

- JNDI(Java Naming and Directory Interface) 네임스페이스에서 관리 대상 오브젝트로서 검색하는 대신, 런타임에 동적으로 연결 팩토리 및 목적지를 작성하기 위한 팩토리 기반 메커니즘
- 객체 IBM MQ classes for Jakarta Messaging 또는 IBM MQ classes for JMS 객체의 속성을 설정하기 위한 일련의 방법
- 문제점에 대한 세부 정보를 확보하기 위한 메소드의 예외 클래스 세트
- 추적을 제어하기 위한 메소드 세트
- 또는에 IBM MQ classes for JMS 대한 버전 정보를 얻기 위한 일련의 방법 IBM MQ classes for Jakarta Messaging

런타임 시점에 동적으로 연결 팩토리 및 목적지를 생성하고, 해당 속성을 설정 및 가져오기 위해 확장 기능은 IBMJMS 기존 IBM MQJMS 확장 기능에 대한 대체 인터페이스 집합을 제공합니다. 하지만 IBM MQ JMS 확장은 IBM MQ 메시징 제공자에 고유한 반면, IBM JMS 확장은 IBM MQ에 고유하지 않으며 [JMS용 IBM MQ 클래스 아키텍처](#)에 설명된 계층화된 아키텍처 내의 모든 메시징 제공자와 함께 사용할 수 있습니다.

com.ibm.msg.client.wmq.WMQConstants (JMS 2.0) 또는 com.ibm.msg.jakarta.client.wmq.WMQConstants (Jakarta Messaging 3.0) 인터페이스에는 애플리케이션이 확장 IBMJMS 기능을 사용하여 또는 IBM MQ classes for Jakarta Messaging IBM MQ classes for JMS 객체의 속성을 설정할 때 사용할 수 있는 상수 정의가 포함되어 있습니다. 인터페이스에는 IBM MQ 메시징 제공자의 상수 및 메시징 제공자와는 독립적인 JMS 상수가 포함되어 있습니다.

다음에 제시된 코드 예제는 해당 Java 클래스에 다음 импорт 문이 포함되어 있다고 가정합니다:

JM 3.0

```
import com.ibm.msg.jakarta.client.jms.*;
import com.ibm.msg.jakarta.client.services.*;
import com.ibm.msg.jakarta.client.wmq.WMQConstants;
```

JMS 2.0

```
import com.ibm.msg.client.jms.*;
```

```
import com.ibm.msg.client.services.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

연결 팩토리 및 목적지 작성

IBM JMS 확장을 사용하여 연결 팩토리 및 목적지를 작성하기 전에, 애플리케이션은 우선 JmsFactoryFactory 오브젝트를 작성해야 합니다. 다음 예제에서 표시된 대로, JmsFactoryFactory 오브젝트를 작성하기 위해 애플리케이션은 JmsFactoryFactory 클래스의 getInstance() 메소드를 호출합니다.

JM 3.0

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.JAKARTA_WMQ_PROVIDER);
```

JMS 2.0

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
```

getInstance() 호출의 매개변수는 선택된 메시징 제공자로서 IBM MQ 메시징 제공자를 식별하는 상수입니다. 그리고 애플리케이션은 JmsFactoryFactory 오브젝트를 사용하여 연결 팩토리 및 목적지를 작성할 수 있습니다.

다음 예제에서 표시된 대로, 연결 팩토리를 작성하기 위해 애플리케이션은 JmsFactoryFactory 오브젝트의 createConnectionFactory() 메소드를 호출합니다.

```
JmsConnectionFactory factory = ff.createConnectionFactory();
```

이 명령문은 모든 해당 특성에 대한 기본값으로 JmsConnectionFactory 오브젝트를 작성하며, 이는 애플리케이션이 바인딩 모드로 기본 큐 관리자에 연결함을 의미합니다. 애플리케이션이 클라이언트 모드로 연결하거나 기본 큐 관리자가 아닌 큐 관리자에 연결하도록 하려면, 애플리케이션이 연결을 작성하기 전에 JmsConnectionFactory 오브젝트의 적절한 특성을 설정해야 합니다. 이를 수행하는 방법에 대한 정보는 [203 페이지의 『IBM MQ classes for JMS 오브젝트의 특성 설정』](#)의 내용을 참조하십시오.

JmsFactoryFactory 클래스에는 다음 유형의 연결 팩토리를 작성하기 위한 메소드도 포함되어 있습니다.

- JmsQueueConnectionFactory
- JmsTopicConnectionFactory
- JmsXAConnectionFactory
- JmsXAQueueConnectionFactory
- JmsXATopicConnectionFactory

큐 오브젝트를 작성하기 위해 애플리케이션은 다음 예제에서 표시된 대로 JmsFactoryFactory 오브젝트의 createQueue() 메소드를 호출합니다.

```
JmsQueue q1 = ff.createQueue("Q1");
```

이 명령문은 모든 해당 특성에 대한 기본값으로 JmsQueue 오브젝트를 작성합니다. 오브젝트는 로컬 큐 관리자에 속하는 Q1이라고 하는 IBM MQ 큐를 표시합니다. 이 큐는 로컬 큐, 알리어스 큐 또는 리모트 큐 정의일 수 있습니다.

또한 createQueue() 메소드는 큐 URI(Uniform Resource Identifier)를 매개변수로서 허용할 수 있습니다. 큐 URI는 IBM MQ 큐의 이름 및 선택적으로 큐를 소유하는 큐 관리자의 이름, 그리고 JmsQueue 오브젝트의 하나 이상의 특성을 지정하는 문자열입니다. 다음 명령문에는 큐 URI의 예가 포함되어 있습니다.

```
JmsQueue q2 = ff.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

이 명령문으로 작성된 JmsQueue 오브젝트는 큐 관리자 QM2가 소유한 Q2라는 IBM MQ 큐를 나타내고, 이 목적지로 전송되는 모든 메시지는 지속적이며 우선순위는 5입니다. 큐 URI에 대한 자세한 정보는 [214 페이지의 『URI\(Uniform Resource Identifier\)』](#)의 내용을 참조하십시오. JmsQueue 오브젝트의 특성을 설정하는 대체 방법은 [203 페이지의 『IBM MQ classes for JMS 오브젝트의 특성 설정』](#)의 내용을 참조하십시오.

다음 예제에서 표시된 대로, 토픽 오브젝트를 작성하기 위해 애플리케이션은 JmsFactoryFactory 오브젝트의 createTopic() 메소드를 사용할 수 있습니다.


```
JmsTopic t1 = ff.createTopic("Sport/Football/Results");
```

이 명령문은 모든 해당 특성에 대한 기본값으로 JmsTopic 오브젝트를 작성합니다. 오브젝트는 Sport/Football/Results라고 하는 토픽을 표시합니다.

또한 createTopic() 메소드는 매개변수로서 토픽 URI를 허용할 수 있습니다. 토픽 URI는 토픽의 이름 및 선택적으로 JmsTopic 오브젝트의 하나 이상의 특성을 지정하는 문자열입니다. 다음 명령문에는 토픽 URI의 예제가 포함되어 있습니다.

```
String s1 = "topic://Sport/Tennis/Results?persistence=1&priority=0";  
JmsTopic t2 = ff.createTopic(s1);
```

이러한 명령문으로 작성된 JmsTopic 오브젝트는 Sport/Tennis/Results라는 토픽을 나타내고, 이 목적지로 전송되는 모든 메시지는 비지속적이며 우선순위는 0입니다. 토픽 URI에 대한 자세한 정보는 [214 페이지의 『URI\(Uniform Resource Identifier\)』](#)의 내용을 참조하십시오. JmsTopic 오브젝트의 특성을 설정하는 대체 방법은 [203 페이지의 『IBM MQ classes for JMS 오브젝트의 특성 설정』](#)의 내용을 참조하십시오.

애플리케이션이 연결 팩토리 및 목적지를 작성한 후에 해당 오브젝트는 선택된 메시징 공급자에서만 사용될 수 있습니다.

IBM MQ classes for JMS 오브젝트의 특성 설정

확장 IBMJMS 기능을 사용하여 객체의 IBM MQ classes for JMS 속성을 설정하려면 애플리케이션은 com.ibm.msg.client.JmsPropertyContext 인터페이스의 메서드를 사용합니다. 마찬가지로, 확장 IBMJMS 기능을 사용하여 객체의 IBM MQ classes for Jakarta Messaging 속성을 설정하려면 애플리케이션은 com.ibm.msg.jakarta.client.JmsPropertyContext 인터페이스의 메서드를 사용합니다.

각각의 Java 데이터 유형마다, JmsPropertyContext 인터페이스에는 해당 데이터 유형의 특성 값을 설정하는 메소드와 해당 데이터 유형의 특성 값을 가져오는 메소드가 포함되어 있습니다. 예를 들어, 애플리케이션은 setIntProperty() 메소드를 호출하여 정수 값의 특성을 설정하며 getIntProperty() 메소드를 호출하여 정수 값의 특성을 가져옵니다.

com.ibm.mq.jms 및 com.ibm.mq.jakarta.jms 패키지의 클래스 인스턴스는 해당 JmsPropertyContext 인터페이스의 메서드를 상속합니다. 따라서 애플리케이션은 이러한 메소드를 사용하여 MQConnectionFactory, MQQueue 및 MQTopic 오브젝트의 특성을 설정합니다.

애플리케이션이 객체를 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 생성할 때, 기본 값이 있는 속성은 자동으로 설정됩니다. 애플리케이션이 특성을 설정하면, 새 값은 특성이 보유한 이전 값을 대체합니다. 특성이 설정된 후에 이는 삭제될 수 없지만 해당 값은 변경될 수 있습니다.

애플리케이션이 속성에 유효하지 않은 값을 설정하려고 시도하면 JMSException 예외를 발생시킵니다 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging. 애플리케이션이 아직 설정되지 않은 특성을 가져오려고 시도하는 경우, 해당 작동은 JMS 스펙에 설명된 대로입니다. IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 기본 데이터 유형에 대해서는 예외를 NumberFormatException 발생시키고, 참조 데이터 유형에 대해서는 null을 반환합니다.

객체의 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 사전 정의된 속성 외에도, 애플리케이션은 자체 속성을 설정할 수 있습니다. 이러한 애플리케이션 정의 속성은 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging에 의해 무시됩니다.

객체의 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 속성에 대한 자세한 내용은 [참조하십시오 객체의 IBM MQ classes for JMS 속성](#).

다음 코드는 IBM JMS 확장을 사용하여 특성을 설정하는 방법의 예제입니다. 코드는 연결 팩토리의 5개 특성을 설정합니다.

```
factory.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE,  
    WMQConstants.WMQ_CM_CLIENT);  
factory.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");  
factory.setStringProperty(WMQConstants.WMQ_HOST_NAME, "HOST1");  
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);  
factory.setStringProperty(WMQConstants.WMQ_CHANNEL, "QM1.SVR");  
factory.setStringProperty(WMQConstants.WMQ_APPLICATIONNAME, "My Application");
```

이러한 특성을 설정한 효과로 애플리케이션은 QM1.SVR이라고 하는 MQI 채널을 사용하여 클라이언트 모드로 큐 관리자 QM1에 연결합니다. 큐 관리자는 호스트 이름 HOST1의 시스템에서 실행 중이며, 큐 관리자의 리스너는 포트 번호 1415에서 대기합니다. 이 연결 및 그 아래의 세션과 연관된 기타 큐 관리자 연결에는 애플리케이션 이름 "My Application"이 이와 연관되어 있습니다.

참고: z/OS 플랫폼에서 실행 중인 큐 관리자는 애플리케이션 이름의 설정을 지원하지 않으며, 따라서 이 설정은 무시됩니다.

JmsPropertyContext 인터페이스에는 애플리케이션이 특성 설정에 사용할 수 있는 setObjectProperty() 메소드도 포함되어 있습니다. 메소드의 두 번째 매개변수는 특성의 값을 캡슐화하는 오브젝트입니다. 예를 들어, 다음 코드는 정수 1415를 캡슐화하는 정수 오브젝트를 작성한 후에 setObjectProperty()를 호출하여 연결 팩토리의 PORT 특성을 1415 값으로 설정합니다.

```
Integer port = new Integer(1415);
factory.setObjectProperty(WMQConstants.WMQ_PORT, port);
```

따라서 이 코드는 다음 명령문과 동일합니다.

```
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
```

이와 반대로, getObjectProperty() 메소드는 특성의 값을 캡슐화하는 오브젝트를 리턴합니다.

한 데이터 유형에서 다른 데이터 유형으로 특성 값의 암시적 변환

응용 프로그램이 JmsPropertyContext 인터페이스의 메서드를 사용하여 객체의 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 속성을 설정하거나 가져올 때, 해당 속성의 값은 한 데이터 유형에서 다른 데이터 유형으로 암시적으로 변환될 수 있습니다.

예를 들어, 다음 명령문은 JmsQueue 오브젝트 q1의 PRIORITY 특성을 설정합니다.

```
q1.setStringProperty(WMQConstants.WMQ_PRIORITY, "5");
```

PRIORITY 특성에 정수 값이 있으므로 setStringProperty() 호출은 암시적으로 문자열 "5"(소스 값)를 정수 5(대상 값)로 변환하며, 이는 다시 PRIORITY 특성의 값이 됩니다.

이와는 반대로, 다음 명령문은 JmsQueue 오브젝트 q1의 PRIORITY 특성을 가져옵니다.

```
String s1 = q1.getStringProperty(WMQConstants.WMQ_PRIORITY);
```

PRIORITY 특성의 값인 정수 5(소스 값)는 getStringProperty() 호출에 의해 문자열 "5"(대상 값)로 암시적으로 변환됩니다.

에서 지원하는 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 변환은 [예 표시됩니다 204 페이지의 표 34](#).

표 34. 한 데이터 유형에서 다른 데이터 유형으로 지원되는 변환	
소스 데이터 유형	지원되는 대상 데이터 유형
부울	문자열
byte	int, long, short, 문자열
char	문자열
double	문자열
float	double, 문자열
int	long, 문자열
long	문자열
short	int, long, 문자열
문자열	부울, 바이트, double, float, int, long, short

지원되는 변환에 적용되는 일반 규칙은 다음과 같습니다.

- 변환 중에 데이터 유실이 없는 경우 숫자 값은 한 데이터 유형에서 다른 데이터 유형으로 변환될 수 있습니다. 예를 들어, 데이터 유형 `int`의 값은 데이터 유형 `long`의 값으로 변환될 수 있지만 데이터 유형 `short`의 값으로는 변환될 수 없습니다.
- 데이터 유형의 값은 문자열로 변환될 수 있습니다.
- 문자열이 변환에 대해 올바른 형식이면, 문자열을 임의의 기타 데이터 유형(`char` 제외)의 값으로 변환할 수 있습니다. 정확한 형식이 아닌 문자열을 변환하려는 애플리케이션이 있을 경우 예외를 `NumberFormatException` 발생시킵니다 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging .
- 응용 프로그램이 지원되지 않는 변환을 시도할 경우 예외를 `MessageFormatException` 발생시킵니다 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging .

한 데이터 유형에서 다른 데이터 유형으로 값을 변환하는 특정 규칙은 다음과 같습니다.

- 부울 값을 문자열로 변환할 때 `true` 값은 문자열 `"true"`로 변환되며 `false` 값은 문자열 `"false"`로 변환됩니다.
- 문자열을 부울 값으로 변환할 때 문자열 `"true"`(대소문자 구분 없음)는 `true`로 변환되며 문자열 `"false"`(대소문자 구분 없음)는 `false`로 변환됩니다. 다른 문자열은 `false`로 변환됩니다.
- 데이터 유형 `byte`, `int`, `long` 또는 `short`의 값을 문자열로 변환할 때 문자열의 형식은 다음과 같아야 합니다.

[*blanks*][*sign*] *digits*

문자열의 컴포넌트의 의미는 다음과 같습니다.

공백

선택적 선두 공백 문자입니다.

sign

선택적 더하기 부호(+) 또는 빼기 부호(-)입니다.

digits

연속적 숫자 시퀀스(0-9)입니다. 적어도 하나의 숫자가 있어야 합니다.

숫자 시퀀스 이후, 문자열에는 숫자가 아닌 기타 문자가 포함될 수 있지만 변환은 이러한 문자 중 첫 번째에 도달하는 순간 중지됩니다. 문자열은 10진수 정수를 표시한다고 가정됩니다.

문자열이 올바른 형식이 아닌 경우 예외를 `NumberFormatException` 발생시킵니다 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging .

- 데이터 유형 `double` 또는 `float`의 값으로 문자열을 변환할 때 문자열의 형식은 다음과 같아야 합니다.

[*blanks*][*sign*] *digits* [*e_char* [*e_sign*] *e_digits*]

문자열의 컴포넌트의 의미는 다음과 같습니다.

공백

선택적 선두 공백 문자입니다.

sign

선택적 더하기 부호(+) 또는 빼기 부호(-)입니다.

digits

연속적 숫자 시퀀스(0-9)입니다. 적어도 하나의 숫자가 있어야 합니다.

e_char

지수 문자(`E` 또는 `e`)입니다.

e_sign

지수의 선택적 더하기 부호(+) 또는 빼기 부호(-)입니다.

e_digits

지수의 연속적 숫자 시퀀스(0-9)입니다. 문자열에 지수 문자가 포함된 경우에는 최소한 하나의 숫자가 존재해야 합니다.

숫자 시퀀스 또는 지수를 표시하는 선택적 문자 이후, 문자열에는 숫자가 아닌 기타 문자가 포함될 수 있지만 변환은 이러한 문자 중 첫 번째에 도달하는 순간 중지됩니다. 문자열은 10제곱인 지수의 10진수 부동 소수점 숫자를 표시한다고 가정됩니다.

문자열이 올바른 형식이 아닌 경우 예외를 `NumberFormatException` 발생시킵니다 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging .

- 숫자 값(데이터 유형 `byte`의 값 포함)을 문자열로 변환할 때 값은 해당 값의 ASCII 문자를 포함하는 문자열이 아닌 10진수 숫자로서 값의 문자열 표현으로 변환됩니다. 예를 들어, 정수 65는 문자열 "A"가 아닌 문자열 "65"로 변환됩니다.

단일 호출에서 둘 이상의 특성 설정

`JmsPropertyContext` 인터페이스에는 `setBatchProperties()` 메소드도 포함되어 있으며, 애플리케이션은 이를 사용하여 단일 호출에서 둘 이상의 특성을 설정할 수 있습니다. 메소드의 매개변수는 특성 이름-값 쌍의 세트를 캡슐화하는 맵 오브젝트입니다.

예를 들어, 다음 코드는 `setBatchProperties()` 메소드를 사용하여 203 페이지의 『IBM MQ classes for JMS 오브젝트의 특성 설정』에서 표시된 대로 연결 팩토리의 동일한 5개 특성을 설정합니다. 코드는 맵 인터페이스를 구현하는 `HashMap` 클래스의 인스턴스를 작성합니다.

```
HashMap batchProperties = new HashMap();
batchProperties.put(WMQConstants.WMQ_CONNECTION_MODE,
    new Integer(WMQConstants.WMQ_CM_CLIENT));
batchProperties.put(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
batchProperties.put(WMQConstants.WMQ_WMQ_HOST_NAME, "HOST1");
batchProperties.put(WMQConstants.WMQ_PORT, "1414");
batchProperties.put(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setBatchProperties(batchProperties);
```

참고로, `Map.put()` 메소드의 두 번째 매개변수는 오브젝트여야 합니다. 따라서 기본 데이터 유형의 특성 값은 예제에 표시된 대로 오브젝트 내에 캡슐화되거나 문자열로 표시되어야 합니다.

`setBatchProperties()` 메소드는 각 특성을 유효성 검증합니다. 예를 들어, 해당 값이 올바르지 않아서 `setBatchProperties()` 메소드가 특성을 설정할 수 없는 경우에는 지정된 어떤 특성도 설정되지 않습니다.

특성 이름 및 값

응용 프로그램이 객체의 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 속성을 설정하고 가져오기 위해 적절한 `JmsPropertyContext` 인터페이스의 메서드를 사용하는 경우, 응용 프로그램은 다음 방법 중 하나로 속성의 이름과 값을 지정할 수 있습니다. 동반된 각각의 예제에서는 큐에 송신된 메시지의 우선순위가 `send()` 호출에 지정될 수 있도록 `JmsQueue` 오브젝트 `q1`의 `PRIORITY` 특성을 설정하는 방법을 표시합니다.

com.ibm.msg.client.wmq.WMQConstants 인터페이스의 상수로서 정의된 특성 이름 및 값 사용

다음 명령문은 이러한 방법으로 특성의 이름 및 값을 지정하는 방법의 예제입니다.

```
q1.setIntProperty(WMQConstants.WMQ_PRIORITY, WMQConstants.WMQ_PRI_APP);
```

큐 및 토픽 **URI(Uniform Resource Identifier)**에서 사용될 수 있는 특성 이름 및 값 사용

다음 명령문은 이러한 방법으로 특성의 이름 및 값을 지정하는 방법의 예제입니다.

```
q1.setIntProperty("priority", -2);
```

목적지의 이름 및 값 특성만 이러한 방법으로 지정될 수 있습니다.

IBM MQ JMS 관리 도구에서 인식하는 특성 이름 및 값 사용

다음 명령문은 이러한 방법으로 특성의 이름 및 값을 지정하는 방법의 예제입니다.

```
q1.setStringProperty("PRIORITY", "APP");
```

다음 명령문에서 표시된 대로 특성 이름의 단축 양식도 허용됩니다.

```
q1.setStringProperty("PRI", "APP");
```

애플리케이션이 특성을 가져오는 경우, 리턴된 값은 애플리케이션이 특성의 이름을 지정하는 방법에 달려 있습니다. 예를 들어, 애플리케이션이 상수 `WMQConstants.WMQ_PRIORITY`를 특성 이름으로 지정하는 경우에 리턴된 값은 정수 -2입니다.

```
int n1 = getIntProperty(WMQConstants.WMQ_PRIORITY);
```

애플리케이션이 문자열 "priority"를 특성 이름으로 지정하는 경우에는 동일한 값이 리턴됩니다.

```
int n2 = getIntProperty("priority");
```

그러나 애플리케이션이 문자열 "PRIORITY" 또는 "PRI"를 특성 이름으로 지정하는 경우에는 리턴된 값이 문자열 "APP"입니다.

```
String s1 = getStringProperty("PRI");
```

내부적으로 속성 이름과 값을 매칭되는 `WMQConstants` 인터페이스에 정의된 리터럴 값으로 저장합니다 `IBM MQ classes for JMS` 그리고 `IBM MQ classes for Jakarta Messaging`. 이는 특성 이름 및 값에 대해 정의된 기본 형식입니다. 일반적으로 애플리케이션이 속성 이름과 값을 지정하는 다른 두 방법 중 하나로 속성을 설정하는 경우, 지정된 입력 형식의 이름과 값을 표준 형식으로 변환해야 합니다 `IBM MQ classes for JMS` 그리고 `IBM MQ classes for Jakarta Messaging`. 마찬가지로, 애플리케이션이 속성 이름과 값을 지정하는 다른 두 방법 중 하나로 속성을 가져오는 경우, 지정된 입력 형식의 이름을 표준 형식으로 변환하고, `IBM MQ classes for JMS` 그리고 `IBM MQ classes for Jakarta Messaging` 표준 형식의 값을 필요한 출력 형식으로 변환해야 합니다. 이러한 변환을 수행해야 하면 성능에 영향을 줄 수 있습니다.

예외, 추적 파일 또는 로그에서 `IBM MQ classes for JMS` 또는 `IBM MQ classes for Jakarta Messaging` 반환되는 속성 이름과 값은 항상 표준 형식으로 표시됩니다.

맵 인터페이스 사용

`JmsPropertyContext` 인터페이스는 `java.util.Map` 인터페이스를 확장합니다. 따라서 애플리케이션은 `Map` 인터페이스의 메서드를 사용하여 객체의 `IBM MQ classes for JMS` 또는 `IBM MQ classes for Jakarta Messaging` 속성에 접근할 수 있습니다.

예를 들어, 다음 코드는 연결 팩토리의 모든 특성의 이름 및 값을 인쇄합니다. 코드는 단지 맵 인터페이스의 메소드만 사용하여 특성의 이름 및 값을 가져옵니다.

```
// Get the names of all the properties
Set propNameSet = factory.keySet();

// Loop round all the property names and get the property values
Iterator iterator = propNameSet.iterator();
while (iterator.hasNext()){
    String propName = (String)iterator.next();
    System.out.println(propName+"="+factory.get(propName));
}
```

맵 인터페이스의 메소드 사용에서는 특성 유효성 검증 또는 변환을 무시하지 않습니다.

IBM MQ JMS 확장 사용

`IBM MQ classes for JMS API`에 JMS 대한 일련의 확장 기능인 확장 `IBM MQJMS` 기능들을 포함합니다. 애플리케이션은 이러한 확장을 사용하여 런타임에 동적으로 연결 팩토리 및 목적지를 작성하고 연결 팩토리 및 목적지의 특성을 설정할 수 있습니다.

`IBM MQ classes for JMS`에는 패키지 `com.ibm.jms` 및 `com.ibm.mq.jms`의 클래스 세트가 포함되어 있습니다. 이러한 클래스는 JMS 인터페이스를 구현하며 `IBM MQ JMS` 확장을 포함합니다. 다음의 코드 예제에서는 이러한 패키지를 다음 명령문으로 가져왔다고 가정합니다.

```
import com.ibm.jms.*;
import com.ibm.mq.jms.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

애플리케이션은 `IBM MQ JMS` 확장을 사용하여 다음 기능을 수행할 수 있습니다.

- **JNDI(Java Naming and Directory Interface)** 네임스페이스에서 관리 대상 오브젝트로서 검색하는 대신, 런타임에 동적으로 연결 팩토리 및 목적지 작성

- 연결 팩토리 및 목적지의 특성 설정

연결 팩토리 작성

연결 팩토리를 작성하기 위해 애플리케이션은 다음 예제에 표시된 대로 MQConnectionFactory 구성자를 사용할 수 있습니다.

```
MQConnectionFactory factory = new MQConnectionFactory();
```

이 명령문은 모든 해당 특성에 대한 기본값으로 MQConnectionFactory 오브젝트를 작성하며, 이는 애플리케이션이 바인딩 모드로 기본 큐 관리자에 연결함을 의미합니다. 애플리케이션이 클라이언트 모드로 연결하거나 기본 큐 관리자가 아닌 큐 관리자에 연결하도록 하려면, 애플리케이션이 연결을 작성하기 전에 MQConnectionFactory 오브젝트의 적절한 특성을 설정해야 합니다. 이를 수행하는 방법에 대한 정보는 [208 페이지의 『연결 팩토리의 특성 설정』](#)의 내용을 참조하십시오.

애플리케이션은 유사한 방법으로 다음 유형의 연결 팩토리를 작성할 수 있습니다.

- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQXAConnectionFactory
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory

연결 팩토리의 특성 설정

애플리케이션은 연결 팩토리의 적절한 메소드를 호출하여 연결 팩토리의 특성을 설정할 수 있습니다. 연결 팩토리는 관리 대상 오브젝트 또는 런타임에 동적으로 작성된 오브젝트일 수 있습니다.

예를 들어, 다음 코드를 고려하십시오.

```
MQConnectionFactory factory = new MQConnectionFactory();
factory.setTransportType(WMQConstants.WMQ_CM_CLIENT);
factory.setQueueManager("QM1");
factory.setHostName("HOST1");
factory.setPort(1415);
factory.setChannel("QM1.SVR");
```

이 코드는 MQConnectionFactory 오브젝트를 작성한 후에 오브젝트의 5개 특성을 설정합니다. 이러한 특성의 설정 효과로 애플리케이션은 QM1.SVR이라고 하는 MQI 채널을 사용하여 클라이언트 모드에서 큐 관리자 QM1에 연결합니다. 큐 관리자는 호스트 이름 HOST1의 시스템에서 실행 중이며, 큐 관리자의 리스너는 포트 번호 1415에서 대기합니다.

브로커에 대한 실시간 연결을 사용하는 애플리케이션은 발행/구독 스타일의 메시징만 사용할 수 있습니다. 포인트-투-포인트 스타일의 메시징은 사용할 수 없습니다.

연결 팩토리의 특성에 대해 특정 조합만 유효합니다. 어떤 조합이 올바른지에 대한 정보는 [IBM MQ classes for JMS 오브젝트의 특성 간의 종속성](#)을 참조하십시오.

연결 팩토리의 특성 및 해당 특성을 설정하는 데 사용되는 메소드에 대한 자세한 정보는 [IBM MQ classes for JMS 오브젝트의 특성](#)을 참조하십시오.

목적지 작성

Queue 오브젝트를 작성하기 위해 애플리케이션은 다음 예제에 표시된 대로 MQQueue 구성자를 사용할 수 있습니다.

```
MQQueue q1 = new MQQueue("Q1");
```

이 명령문은 모든 해당 특성에 대한 기본값으로 MQQueue 오브젝트를 작성합니다. 오브젝트는 로컬 큐 관리자에 속하는 Q1이라고 하는 IBM MQ 큐를 표시합니다. 이 큐는 로컬 큐, 알리어스 큐 또는 리모트 큐 정의일 수 있습니다.

MQQueue 구성자의 대체 양식에는 다음 예제에 표시된 대로 두 개의 매개변수가 있습니다.

```
MQQueue q2 = new MQQueue("QM2", "Q2");
```

이 명령문에 의해 작성되는 MQQueue 오브젝트는 큐 관리자 QM2이 소유하는 Q2라고 하는 IBM MQ 큐를 표시합니다. 이 방법으로 식별되는 큐 관리자는 로컬 큐 관리자 또는 리모트 큐 관리자일 수 있습니다. 리모트 큐 관리자인 경우, IBM MQ는 애플리케이션이 이 목적지에 메시지를 송신할 때 WebSphere MQ가 로컬 큐 관리자의 메시지를 리모트 큐 관리자로 라우팅할 수 있도록 구성되어야 합니다.

또한 MQQueue 구성자는 큐 URI(Uniform Resource Identifier)를 단일 매개변수로서 허용할 수 있습니다. 큐 URI는 IBM MQ 큐의 이름 및 선택적으로 큐를 소유하는 큐 관리자의 이름, 그리고 MQQueue 오브젝트의 하나 이상의 특성을 지정하는 문자열입니다. 다음 명령문에는 큐 URI의 예가 포함되어 있습니다.

```
MQQueue q3 = new MQQueue("queue://QM3/Q3?persistence=2&priority=5");
```

이 명령문으로 작성된 MQQueue 오브젝트는 큐 관리자 QM3이 소유한 Q3이라는 IBM MQ 큐를 나타내고, 이 목적지로 전송되는 모든 메시지는 지속적이며 우선순위는 5입니다. 큐 URI에 대한 자세한 정보는 [214 페이지의 『URI\(Uniform Resource Identifier\)』](#)의 내용을 참조하십시오. MQQueue 오브젝트의 특성을 설정하는 대체 방법은 [209 페이지의 『목적지의 특성 설정』](#)의 내용을 참조하십시오.

큐 오브젝트를 작성하기 위해 애플리케이션은 다음 예제에 표시된 대로 MQTopic 구성자를 사용할 수 있습니다.

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
```

이 명령문은 모든 해당 특성에 대한 기본값으로 MQTopic 오브젝트를 작성합니다. 오브젝트는 Sport/Football/Results라고 하는 토픽을 표시합니다.

또한 MQTopic 구성자는 매개변수로서 토픽 URI를 허용할 수 있습니다. 토픽 URI는 토픽의 이름 및 선택적으로 MQTopic 오브젝트의 하나 이상의 특성을 지정하는 문자열입니다. 다음 명령문에는 토픽 URI의 예제가 포함되어 있습니다.

```
MQTopic t2 = new MQTopic("topic://Sport/Tennis/Results?persistence=1&priority=0");
```

이러한 명령문으로 작성된 MQTopic 오브젝트는 Sport/Tennis/Results라는 토픽을 나타내고, 이 목적지로 전송되는 모든 메시지는 비지속적이며 우선순위는 0입니다. 토픽 URI에 대한 자세한 정보는 [214 페이지의 『URI\(Uniform Resource Identifier\)』](#)의 내용을 참조하십시오. MQTopic 오브젝트의 특성을 설정하는 대체 방법은 [209 페이지의 『목적지의 특성 설정』](#)의 내용을 참조하십시오.

목적지의 특성 설정

애플리케이션은 목적지의 적절한 메소드를 호출하여 목적지의 특성을 설정할 수 있습니다. 목적지는 관리 대상 오브젝트 또는 런타임에 동적으로 작성된 오브젝트일 수 있습니다.

예를 들어, 다음 코드를 고려하십시오.

```
MQQueue q1 = new MQQueue("Q1");
q1.setPersistence(WMQConstants.WMQ_PER_PER);
q1.setPriority(5);
```

이 코드는 MQQueue 오브젝트를 작성한 후에 오브젝트의 2개 특성을 설정합니다. 이러한 특성의 설정 효과로 목적지에 송신된 모든 메시지는 지속적이며 우선순위는 5입니다.

다음 예제에서 표시된 대로, 애플리케이션은 유사한 방법으로 MQTopic 오브젝트의 특성을 설정할 수 있습니다.

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
t1.setPersistence(WMQConstants.WMQ_PER_NON);
t1.setPriority(0);
```

이 코드는 MQTopic 오브젝트를 작성한 후에 오브젝트의 2개 특성을 설정합니다. 이러한 특성의 설정 효과로 목적지에 송신된 모든 메시지는 비지속적이며 우선순위는 0입니다.

목적지의 특성 및 해당 특성을 설정하는 데 사용되는 메소드에 대한 자세한 정보는 [IBM MQ classes for JMS](#) 오브젝트의 특성을 참조하십시오.

연결을 구축하기 위해 애플리케이션은 JMS `ConnectionFactory` 객체를 사용하여 `Connection` 객체를 생성한 후 연결을 시작합니다.

ConnectionFactory 객체와 `JMS 2.0 connect()` `createContext()` 메서드를 사용하여 애플리케이션은 일반적으로 메시징 공급자에 연결합니다.

이전 JMS버전의에서는 먼저 `Connection` 객체를 생성하기 위해 `createConnection` `CreateConnection` 을 사용한 다음, 연결을 시작한 후, 마지막으로 메시징 작업을 수행할 수 있는 `Session` 객체를 생성하기 위해 `getSession()` `CreateSession` 을 호출해야 했습니다.

JMSContext 객체는 Connection 객체와 Session 객체를 효과적으로 캡슐화합니다. 전통적인 방식을 사용하여 연결 및 세션 객체를 직접 생성하려면 [211 페이지의 『JMS 애플리케이션에서 세션 작성』](#) 를 [210 페이지의 『JMS 애플리케이션에서 연결 빌드』](#) 참조하십시오.

JMSContext 객체를 생성하려면 애플리케이션은 다음 예시와 같이 ConnectionFactory 객체의 createContext() 메서드를 사용합니다:

```
ConnectionFactory factory;
JMSContext context;
.
.
.
context = factory.createContext();
```

컨텍스트가 JMS 생성되면, 연결 핸들(Hconn)을 IBM MQ classes for JMS 생성하고 큐 매니저와 대화를 시작합니다.

참고: 참고로, 애플리케이션 프로세스 ID는 큐 관리자에 전달되는 기본 사용자 ID로서 사용됩니다. 애플리케이션이 클라이언트 전송 모드에서 실행 중인 경우에는 서버에서 관련 권한 부여와 함께 이 프로세스 ID가 존재해야 합니다. 다른 ID를 사용하려면 해당 createConnection(username,password) 메서드를 사용하십시오.

이 메커니즘은 인증 토큰을 제공하기 위해 사용할 수도 있습니다. [선택한 토큰 발급자로부터 인증 토큰을 획득하는 방법](#)을 참조하십시오.

JMS 1.0

JMS 애플리케이션에서 연결 빌드

연결을 구축 하기 JMS 1.0위해 애플리케이션은 JMS `ConnectionFactory` 객체를 사용하여 `Connection` 객체를 생성한 후 연결을 시작합니다.

다음 예제에서 표시된 대로, Connection 오브젝트를 작성하기 위해 애플리케이션은 ConnectionFactory 오브젝트의 createConnection() 메소드를 사용합니다.

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createConnection();
```

JMS 연결이 작성될 때 IBM MQ classes for JMS는 연결 핸들(Hconn)을 작성하며 큐 관리자와 대화를 시작합니다.

QueueConnectionFactory 인터페이스 및 TopicConnectionFactory 인터페이스는 각각 ConnectionFactory 인터페이스에서 createConnection() 메소드를 상속합니다. 따라서 다음 예제에서 표시된 대로 createConnection() 메소드를 사용하여 도메인 특정 오브젝트를 작성할 수 있습니다.

```
QueueConnectionFactory qcf;
Connection connection;
.
.
.
connection = qcf.createConnection();
```

이 코드 단편은 QueueConnection 오브젝트를 작성합니다. 애플리케이션은 이제 이 오브젝트에서 도메인 독립적인 조작을 수행하거나 포인트-투-포인트 도메인에만 적용되는 조작을 수행할 수 있습니다. 그러나 애플리케이션

션이 발행/구독 도메인에만 적용되는 조작을 수행하려고 시도하는 경우에는 다음 메시지와 함께 `IllegalStateException` 예외가 전달됩니다.

```
JMSMQ1112: Operation for a domain specific object was not valid.  
Operation createProducer() is not valid for type com.ibm.mq.jms.MQTopic
```

이는 도메인 특정 연결 팩토리에서 연결이 작성되었기 때문입니다.

참고: 참고로, 애플리케이션 프로세스 ID는 큐 관리자에 전달되는 기본 사용자 ID로서 사용됩니다. 애플리케이션이 클라이언트 전송 모드에서 실행 중인 경우에는 서버에서 관련 권한 부여와 함께 이 프로세스 ID가 존재해야 합니다. 다른 ID의 사용을 원하는 경우에는 `createConnection(username, password)` 메소드를 사용하십시오.

JMS 스펙에서는 연결이 `stopped` 상태에서 작성되었음을 기술합니다. 연결이 시작될 때까지 연결과 연관된 메시징 이용자는 메시지를 수신할 수 없습니다. 다음 예제에서 표시된 대로, 연결을 시작하기 위해 애플리케이션은 `Connection` 오브젝트의 `start()` 메소드를 사용합니다.

```
connection.start();
```

이 메커니즘은 인증 토큰을 제공하기 위해 사용할 수도 있습니다. [선택한 토큰 발급자로부터 인증 토큰을 획득하는 방법을 참조하십시오.](#)

JMS 1.0 JMS 애플리케이션에서 세션 작성

세션을 생성하기 위해 JMS 1.0 애플리케이션은 `JMS Connection` 객체의 `createSession()` 메소드를 사용합니다.

`createSession()` 메소드에는 두 개의 매개변수가 있습니다.

1. 세션이 트랜잭션되거나 트랜잭션되지 않는지 여부를 지정하는 매개변수
2. 세션의 수신확인 모드를 지정하는 매개변수

예를 들어, 다음 코드는 트랜잭션되지 않았으며 수신확인 모드가 `AUTO_ACKNOWLEDGE`인 세션을 작성합니다.

```
Session session;  
.  
boolean transacted = false;  
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
```

JMS 세션이 작성될 때 IBM MQ classes for JMS는 연결 핸들(`Hconn`)을 작성하며 큐 관리자와 대화를 시작합니다.

`Session` 오브젝트 및 여기서 작성된 `MessageProducer` 또는 `MessageConsumer` 오브젝트는 멀티스레드 애플리케이션의 서로 다른 스레드에 의해 동시에 사용될 수 없습니다. 이러한 오브젝트가 동시에 사용되지 않음을 보장하는 가장 간단한 방법은 각 스레드에 대해 별도의 `Session` 오브젝트를 작성하는 것입니다.

이 메커니즘은 인증 토큰을 제공하기 위해 사용할 수도 있습니다. [선택한 토큰 발급자로부터 인증 토큰을 획득하는 방법을 참조하십시오.](#)

JMS 애플리케이션의 트랜잭션 세션

JMS 애플리케이션은 우선 트랜잭션 세션을 작성하여 로컬 트랜잭션을 실행할 수 있습니다. 애플리케이션은 트랜잭션을 커밋하거나 롤백할 수 있습니다.

JMS 애플리케이션은 로컬 트랜잭션을 실행할 수 있습니다. 로컬 트랜잭션은 애플리케이션이 연결된 큐 관리자의 자원에 대한 변경사항만 포함되는 트랜잭션입니다. 로컬 트랜잭션을 실행하기 위해, 애플리케이션은 우선 세션이 트랜잭션된 매개변수로서 지정하여 `Connection` 오브젝트의 `createSession()` 메소드를 호출하여 트랜잭션 세션을 작성해야 합니다. 결과적으로, 세션 내에서 송신 및 수신된 모든 메시지는 트랜잭션의 시퀀스로 그룹화됩니다. 트랜잭션이 시작된 이후에 송신하거나 수신한 메시지를 애플리케이션이 커밋하거나 롤백하면 트랜잭션이 종료됩니다.

트랜잭션을 커밋하기 위해 애플리케이션은 `Session` 오브젝트의 `commit()` 메소드를 호출합니다. 트랜잭션이 커밋되는 경우, 트랜잭션 내에서 송신된 모든 메시지가 다른 애플리케이션에 전달하기 위해 사용 가능하며 트랜잭션 내에서 수신된 모든 메시지가 수신확인되므로 메시징 서버는 이를 다시 애플리케이션에 전달하려고 시도하지 않습니다. 포인트-투-포인트 도메인에서, 메시징 서버는 해당 큐에서 수신된 메시지도 제거합니다.

트랜잭션을 롤백하기 위해 애플리케이션은 `Session` 오브젝트의 `rollback()` 메소드를 호출합니다. 트랜잭션이 롤백되면 메시징 서버는 트랜잭션 내의 송신된 모든 메시지를 버리며, 트랜잭션 내의 수신된 모든 메시지가 다시 전

달을 위해 사용 가능합니다. 포인트-투-포인트 도메인에서, 수신된 메시지는 해당 큐에 다시 놓이며 다시 기타 애플리케이션에서 이를 볼 수 있습니다.

애플리케이션이 트랜잭션 세션을 작성하거나 `commit()` 또는 `rollback()` 메소드를 호출하면 새 트랜잭션이 자동으로 시작됩니다. 따라서 트랜잭션 세션에는 항상 활성 트랜잭션이 있습니다.

애플리케이션이 트랜잭션 세션을 닫으면 암시적 롤백이 발생합니다. 애플리케이션이 연결을 닫으면 모든 연결의 트랜잭션 세션에 대해 암시적 롤백이 발생합니다.

연결을 닫지 않고 애플리케이션이 종료되면 모든 연결의 트랜잭션 세션에 대해 암시적 롤백도 발생합니다.

트랜잭션은 전적으로 트랜잭션 세션 내에 포함됩니다. 트랜잭션은 세션 간에 걸칠 수 없습니다. 이는 애플리케이션이 두 개 이상의 트랜잭션 세션에서 메시지를 송신하고 수신한 후에 이 모든 조치를 단일 트랜잭션으로 커밋하거나 롤백할 수 없음을 의미합니다.

JMS 세션의 수신확인 모드

트랜잭션되지 않은 모든 세션에는 애플리케이션이 수신한 메시지가 수신확인되는 방법을 결정하는 수신확인 모드가 있습니다. 3개의 수신확인 모드가 사용 가능하며, 수신확인 모드의 선택사항은 애플리케이션의 디자인에 영향을 줍니다.

세션이 트랜잭션되지 않은 경우, 애플리케이션이 수신한 메시지가 수신확인되는 방법은 세션의 수신확인 모드에 의해 판별됩니다. 다음 단락에서는 세 개의 수신확인 모드가 설명되어 있습니다.

AUTO_ACKNOWLEDGE

세션은 애플리케이션이 수신한 각 메시지를 자동으로 수신확인합니다.

메시지가 애플리케이션에 동기적으로 전달되는 경우, 세션은 수신 호출이 완료될 때마다 메시지의 수신을 수신확인합니다. 메시지가 비동기적으로 전달되는 경우, 세션은 메시지 리스너의 `onMessage()` 메소드에 대한 호출이 완료될 때마다 메시지의 수신을 수신확인합니다.

애플리케이션이 메시지를 성공적으로 수신하지만 장애로 인해 수신확인이 발생하지 않는 경우, 해당 메시지는 다시 전달에 사용될 수 있습니다. 따라서 애플리케이션은 다시 전달되는 메시지를 핸들링할 수 있어야 합니다.

DUPS_OK_ACKNOWLEDGE

세션은 선택된 시간에 애플리케이션이 수신한 메시지를 수신확인합니다.

이 수신확인 모드를 사용하면 세션이 수행해야 하는 작업의 양이 줄어들지만, 메시지 수신확인을 방지하는 장애의 결과로 인해 둘 이상의 메시지가 다시 전달에 사용될 수 있습니다. 따라서 애플리케이션은 다시 전달되는 메시지를 핸들링할 수 있어야 합니다.

제한사항: AUTO_ACKNOWLEDGE 및 DUPS_OK_ACKNOWLEDGE 모드에서, JMS는 메시지 리스너에서 핸들링되지 않은 예외를 전달하는 애플리케이션을 지원하지 않습니다. 이는 (장애가 치명적이지 않으며 애플리케이션의 진행을 방해하지 않는 한) 성공적으로 처리되었는지 여부와 무관하게 메시지 리스너가 리턴할 때 메시지가 항상 수신확인됨을 의미합니다. 메시지 수신확인의 보다 미세한 제어가 필요하다면

CLIENT_ACKNOWLEDGE 또는 트랜잭션 모드를 사용하십시오. 이는 수신확인 기능에 대한 전체 제어를 애플리케이션에 제공합니다.

CLIENT_ACKNOWLEDGE

애플리케이션은 Message 클래스의 Acknowledge 메소드를 호출하여 수신한 메시지를 수신확인합니다.

애플리케이션은 각 메시지의 수신을 개별적으로 수신확인할 수도 있으며, 또는 메시지의 일괄처리를 수신하고 수신한 마지막 메시지에 대해서만 Acknowledge 메소드를 호출할 수도 있습니다. Acknowledge 메소드가 호출되는 경우, 메소드가 마지막으로 호출된 후에 수신된 모든 메시지가 수신확인됩니다.

이러한 수신확인 모드와 결합하여, 애플리케이션은 Session 클래스의 Recover 메소드를 호출하여 세션에서 메시지의 전달을 중지하고 다시 시작할 수 있습니다. 수신되었지만 이전에 수신확인되지 않은 메시지는 다시 전달됩니다. 그러나 이는 이전에 전달되었던 동일한 순서대로 전달되지 않을 수 있습니다. 그 동안에 보다 높은 우선 순위의 메시지가 도착할 수도 있으며, 원래 메시지 중 일부가 만료될 수도 있습니다. 또한 포인트-투-포인트 도메인에서 원래 메시지의 일부를 다른 애플리케이션이 이용했을 수도 있습니다.

애플리케이션은 메시지의 JMSRedelivered 헤더 필드의 콘텐츠를 검사하여 메시지를 재전달하는지 여부를 판별할 수 있습니다. 애플리케이션은 Message 클래스의 getJMSRedelivered() 메소드를 호출하여 이를 수행합니다.

JMS 애플리케이션에서 목적지 작성

JNDI(Java Naming and Directory Interface)에서 관리 대상 오브젝트로서 목적지를 검색하는 대신, JMS 애플리케이션은 세션을 사용하여 런타임에 동적으로 목적지를 작성할 수 있습니다. 애플리케이션은 URI(Uniform Resource Identifier)를 사용하여 IBM MQ 큐 또는 토픽을 식별하고, 선택적으로 큐 또는 토픽 오브젝트의 하나 이상의 특성을 지정할 수 있습니다.

세션을 사용하여 큐 오브젝트 작성

다음 예제에서 표시된 대로, 큐 오브젝트를 작성하기 위해 애플리케이션은 Session 오브젝트의 `createQueue()` 메소드를 사용할 수 있습니다.

```
Session session;  
Queue q1 = session.createQueue("Q1");
```

이 코드는 모든 해당 특성에 대한 기본값으로 큐 오브젝트를 작성합니다. 오브젝트는 로컬 큐 관리자에 속하는 Q1이라고 하는 IBM MQ 큐를 표시합니다. 이 큐는 로컬 큐, 알리어스 큐 또는 리모트 큐 정의일 수 있습니다.

`createQueue()` 메소드는 큐 URI를 매개변수로서 허용합니다. 큐 URI는 IBM MQ 큐의 이름 및 선택적으로 큐를 소유하는 큐 관리자의 이름, 그리고 큐 오브젝트의 하나 이상의 특성을 지정하는 문자열입니다. 다음 명령문에는 큐 URI의 예가 포함되어 있습니다.

```
Queue q2 = session.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

이 명령문으로 작성된 Queue 오브젝트는 큐 관리자 QM2가 소유한 Q2라는 IBM MQ 큐를 나타내고, 이 목적지로 전송되는 모든 메시지는 지속적이며 우선순위는 5입니다. 이 방법으로 식별되는 큐 관리자는 로컬 큐 관리자 또는 리모트 큐 관리자일 수 있습니다. 리모트 큐 관리자인 경우, IBM MQ는 애플리케이션이 이 목적지에 메시지를 송신할 때 WebSphere MQ가 로컬 큐 관리자의 메시지를 큐 관리자 QM2로 라우팅할 수 있도록 구성되어야 합니다. URI에 대한 자세한 정보는 214 페이지의 『URI(Uniform Resource Identifier)』의 내용을 참조하십시오.

참고로, `createQueue()` 메소드의 매개변수에는 제공자 특정 정보가 포함되어 있습니다. 따라서 JNDI 네임스페이스에서 관리 대상 오브젝트로서 큐 오브젝트를 검색하는 대신 `createQueue()` 메소드를 사용하여 큐 오브젝트를 작성하면 애플리케이션의 휴대성이 낮아질 수 있습니다.

다음 예제에서 표시된 대로, 애플리케이션은 Session 오브젝트의 `createTemporaryQueue()` 메소드를 사용하여 TemporaryQueue 오브젝트를 작성할 수 있습니다.

```
TemporaryQueue q3 = session.createTemporaryQueue();
```

세션을 사용하여 임시 큐를 작성함에도 불구하고, 임시 큐의 범위는 세션 작성에 사용된 연결입니다. 연결의 세션은 임시 큐에 대한 메시지 작성자 및 메시지 이용자를 작성할 수 있습니다. 임시 큐는 연결이 종료되거나 애플리케이션이 `TemporaryQueue.delete()` 메소드를 사용하여 임시 큐를 명시적으로 삭제할 때까지 유지됩니다(둘 중에서 빠른 쪽이 적용됨).

애플리케이션이 임시 큐를 작성하는 경우, IBM MQ classes for JMS는 애플리케이션이 연결되는 큐 관리자에서 동적 큐를 작성합니다. 연결 팩토리의 `TEMPMODEL` 특성은 동적 큐를 작성하는 데 사용되는 모델 큐의 이름을 지정하며, 연결 팩토리의 `TEMPQPREFIX` 특성은 동적 큐의 이름을 구성하는 데 사용되는 접두부를 지정합니다.

세션을 사용하여 토픽 오브젝트 작성

다음 예제에서 표시된 대로, 토픽 오브젝트를 작성하기 위해 애플리케이션은 Session 오브젝트의 `createTopic()` 메소드를 사용할 수 있습니다.

```
Session session;  
Topic t1 = session.createTopic("Sport/Football/Results");
```

이 코드는 모든 해당 특성에 대한 기본값으로 토픽 오브젝트를 작성합니다. 오브젝트는 Sport/Football/Results라고 하는 토픽을 표시합니다.

또한 `createTopic()` 메소드는 매개변수로서 토픽 URI를 허용합니다. 토픽 URI는 토픽의 이름 및 선택적으로 토픽 오브젝트의 하나 이상의 특성을 지정하는 문자열입니다. 다음 코드에는 토픽 URI의 예제가 포함되어 있습니다.


```
String uri = "topic://Sport/Tennis/Results?persistence=1&priority=0";
Topic t2 = session.createTopic(uri);
```

이러한 명령문으로 작성된 Topic 오브젝트는 Sport/Tennis/Results라는 토픽을 나타내고, 이 목적지로 전송되는 모든 메시지는 비지속적이며 우선순위는 0입니다. 토픽 URI에 대한 자세한 정보는 [214 페이지의 『URI\(Uniform Resource Identifier\)』](#)의 내용을 참조하십시오.

참고로, createTopic() 메소드의 매개변수에는 제공자 특정 정보가 포함되어 있습니다. 따라서 JNDI 네임스페이스에서 관리 대상 오브젝트로서 Topic 오브젝트를 검색하는 대신 createTopic() 메소드를 사용하여 Topic 오브젝트를 작성하면 애플리케이션의 휴대성이 낮아질 수 있습니다.

다음 예제에서 표시된 대로, 애플리케이션은 Session 오브젝트의 createTemporaryTopic() 메소드를 사용하여 TemporaryTopic 오브젝트를 작성할 수 있습니다.

```
TemporaryTopic t3 = session.createTemporaryTopic();
```

세션을 사용하여 임시 토픽을 작성함에도 불구하고, 임시 토픽의 범위는 세션 작성에 사용된 연결입니다. 연결의 세션은 임시 토픽에 대한 메시지 작성자 및 메시지 이용자를 작성할 수 있습니다. 임시 토픽은 연결이 종료되거나 애플리케이션이 TemporaryTopic.delete() 메소드를 사용하여 임시 토픽을 명시적으로 삭제할 때까지 유지됩니다(둘 중에서 빠른 쪽이 적용됨).

애플리케이션이 임시 토픽을 작성하는 경우, IBM MQ classes for JMS는 TEMP/tempTopicPrefix 문자로 시작되는 이름의 토픽을 작성합니다. 여기서 tempTopicPrefix는 연결 팩토리의 TEMPTOPICPREFIX 특성의 값입니다.

URI(Uniform Resource Identifier)

큐 URI는 IBM MQ 큐의 이름 및 선택적으로 큐를 소유하는 큐 관리자의 이름, 그리고 애플리케이션이 작성한 큐 오브젝트의 하나 이상의 특성을 지정하는 문자열입니다. 토픽 URI는 토픽의 이름 및 선택적으로 애플리케이션이 작성한 토픽 오브젝트의 하나 이상의 특성을 지정하는 문자열입니다.

큐 URI의 형식은 다음과 같습니다.

```
queue://[ qMgrName ]/qName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

토픽 URI의 형식은 다음과 같습니다.

```
topic://topicName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

이 형식의 변수에는 다음 의미가 있습니다.

qMgrName

URI가 식별하는 큐를 소유하는 큐 관리자의 이름입니다.

큐 관리자는 로컬 큐 관리자 또는 리모트 큐 관리자일 수 있습니다. 리모트 큐 관리자인 경우, IBM MQ는 애플리케이션이 큐에 메시지를 송신할 때 WebSphere MQ가 로컬 큐 관리자의 메시지를 리모트 큐 관리자로 라우팅할 수 있도록 구성되어야 합니다.

이름을 지정하지 않으면, 로컬 큐 관리자라고 가정합니다.

qName

IBM MQ 큐의 이름입니다.

큐는 로컬 큐, 알리어스 큐 또는 리모트 큐 정의일 수 있습니다.

큐 이름 작성의 규칙은 [IBM MQ 오브젝트의 이름 지정 규칙](#)을 참조하십시오.

topicName

토픽의 이름입니다.

토픽 이름 작성의 규칙은 [IBM MQ 오브젝트의 이름 지정 규칙](#)을 참조하십시오. 와일드카드 문자 +, #, * 및 ? 를 토픽 이름에서 사용하지 마십시오. 이러한 문자가 포함된 토픽 이름은 이를 구독할 때 예상치 못한 결과를 유발할 수 있습니다. [주제 문자열 결합](#) 참조.

propertyName1, propertyName2, ...

애플리케이션이 작성한 큐 또는 토픽 오브젝트의 특성의 이름입니다. [215 페이지의 표 35](#)에는 URI에서 사용될 수 있는 올바른 특성 이름이 나열되어 있습니다.

특성을 지정하지 않으면 큐 또는 토픽 오브젝트가 모든 해당 특성에 대해 기본값을 갖습니다.

propertyValue1, propertyValue2, ...

애플리케이션이 작성한 큐 또는 토픽 오브젝트의 특성의 값입니다. [215 페이지의 표 35](#)에는 URI에서 사용될 수 있는 올바른 특성 값이 나열되어 있습니다.

대괄호([])는 선택적 컴포넌트를 표시하며, 생략 기호(...)는 특성 이름-값 쌍의 목록(존재하는 경우)에 하나 이상의 이름-값 쌍이 포함될 수 있음을 의미합니다.

[215 페이지의 표 35](#)에는 큐 및 토픽 URI에서 사용될 수 있는 올바른 특성 이름 및 올바른 값이 나열되어 있습니다. IBM MQ JMS 관리 도구가 특성의 값에 대해 기호 상수를 사용하지만, URI에는 기호 상수가 포함될 수 없습니다.

표 35. 큐 및 토픽 URI에 사용할 특성 이름 및 올바른 값		
특성 이름	설명	올바른 값
CCSID	IBM MQ classes for JMS가 메시지를 목적지에 전달할 때 메시지 본문의 문자 데이터가 표시되는 방법	IBM MQ에서 지원하는 코드화 문자 세트 ID.
encoding	IBM MQ classes for JMS가 메시지를 목적지에 전달할 때 메시지 본문의 숫자 데이터가 표시되는 방법	메시지 IBM MQ 디스크립터의 인코딩 필드에 대한 유효한 값.
expiry	목적지에 송신되는 메시지의 잔존 시간	-2 - send() 호출에 지정된 대로이거나, send() 호출에 지정되지 않은 경우에는 메시지 생성자의 기본 잔존 시간입니다. 0 - 목적지에 송신된 메시지가 절대 만료되지 않습니다. - 잔존 시간을 지정하는 양의 정수(밀리초).
multicast	브로커에 대한 실시간 연결을 사용 중일 때 토픽에 대한 멀티캐스트 설정	다음 목록에는 올바른 값이 포함되어 있습니다. IBM MQ JMS 관리 도구에서 사용된 MULTICAST 특성의 해당되는 값이 각 값과 연관되어 있습니다. MULTICAST 특성 및 올바른 값에 대한 설명은 IBM MQ classes for JMS 오브젝트의 특성을 참조하십시오. -1 - ASCF 0 - DISABLED 3 - NOTR 5 - RELIABLE 7 - ENABLED

표 35. 큐 및 토픽 URI에 사용할 특성 이름 및 올바른 값 (계속)

특성 이름	설명	올바른 값
persistence	목적지에 송신된 메시지의 지속성	-2 - send() 호출에 지정된 대로이거나, send() 호출에 지정되지 않은 경우에는 메시지 생성자의 기본 지속성입니다. -1 - 큐 IBM MQ 또는 토픽의 속성에 <i>DefPersistence</i> 명시된 대로. 1 - 비지속. 2 - 지속. 3 - IBM MQ JMS 관리 도구에서 사용된 PERSISTENCE 특성의 HIGH 값과 동등합니다. 이 값에 대한 설명은 242 페이지의 『JMS 지속 메시지』의 내용을 참조하십시오.
priority	목적지에 송신된 메시지의 우선순위	-2 - send() 호출에 지정된 대로이거나, send() 호출에 지정되지 않은 경우에는 메시지 생성자의 기본 우선순위입니다. -1 - 큐 IBM MQ 또는 토픽의 속성에 <i>DefPriority</i> 명시된 대로. - 목적지에 송신된 메시지의 우선순위를 지정하는 0-9 범위의 정수.
targetClient	목적지에 송신된 메시지에 MQRFH2 헤더가 포함되는지 여부	0 - 메시지에 MQRFH2 헤더가 포함됩니다. 1 - 메시지에 MQRFH2 헤더가 포함되지 않습니다.

예를 들어, 다음 URI는 로컬 큐 관리자가 소유하는 Q1이라고 하는 IBM MQ 큐를 식별합니다. 이 URI를 사용하여 작성된 큐 오브젝트는 모든 해당 특성에 대해 기본값을 갖습니다.

```
queue:///Q1
```

다음 URI는 QM2라고 하는 큐 관리자가 소유하는 Q2라고 하는 IBM MQ 큐를 식별합니다. 이 목적지로 전송되는 모든 메시지의 우선순위는 6입니다. 이 URI를 사용하여 작성된 Queue 오브젝트의 나머지 특성은 해당 기본값을 가집니다.

```
queue://QM2/Q2?priority=6
```

다음 URI는 Sport/Athletics/Results라고 하는 토픽을 식별합니다. 이 목적지로 전송되는 모든 메시지는 비지속적이며 우선순위는 0입니다. 이 URI를 사용하여 작성된 Topic 오브젝트의 나머지 특성은 해당 기본값을 가집니다.

```
topic://Sport/Athletics/Results?persistence=1&priority=0
```

JMS 애플리케이션에서 메시지 송신

JMS 애플리케이션이 목적지에 메시지를 송신하기 전에, 이는 우선 목적지에 대한 MessageProducer 오브젝트를 작성해야 합니다. 메시지를 목적지에 송신하기 위해, 애플리케이션은 메시지 오브젝트를 작성한 후에 MessageProducer 오브젝트의 send() 메소드를 호출합니다.

애플리케이션은 MessageProducer 오브젝트를 사용하여 메시지를 송신합니다. 메시지 생성자를 사용하여 송신된 모든 메시지가 동일한 목적지로 송신될 수 있도록, 애플리케이션은 일반적으로 큐 또는 토픽이 될 수 있는 특정 목적지에 대한 MessageProducer 오브젝트를 작성합니다. 따라서 애플리케이션이 MessageProducer 오브

젝트를 작성할 수 있으려면 우선 애플리케이션이 Queue 또는 Topic 오브젝트를 작성해야 합니다. Queue 또는 Topic 오브젝트 작성에 대한 정보는 다음 주제를 참조하십시오.

- [199 페이지의 『JNDI를 사용하여 또는 Jakarta Messaging JMS 애플리케이션에서 관리되는 객체를 검색하기』](#)
- [201 페이지의 『IBM JMS 확장 사용』](#)
- [207 페이지의 『IBM MQ JMS 확장 사용』](#)
- [213 페이지의 『JMS 애플리케이션에서 목적지 작성』](#)

다음 예제에서 표시된 대로, MessageProducer 오브젝트를 작성하기 위해 애플리케이션은 Session 오브젝트의 createProducer() 메소드를 사용합니다.

```
MessageProducer producer = session.createProducer(destination);
```

destination 매개변수는 애플리케이션이 이전에 작성한 큐 또는 토픽 오브젝트입니다.

애플리케이션이 메시지를 송신하기 전에, 이는 메시지 오브젝트를 작성해야 합니다. 메시지의 본문에는 애플리케이션 데이터가 포함되어 있으며, JMS는 5가지 유형의 메시지 본문을 정의합니다.

- 바이트
- 맵
- 오브젝트
- 스트림
- 텍스트

메시지 본문의 각 유형에는 메시지 인터페이스의 하위 인터페이스인 자체 JMS 인터페이스 및 해당 본문 유형으로 메시지를 작성하기 위한 Session 인터페이스의 메소드가 있습니다. 예를 들어, 텍스트 메시지의 인터페이스를 TextMessage라고 하며 다음 예제에서 표시된 대로 애플리케이션은 Session 오브젝트의 createTextMessage() 메소드를 사용하여 텍스트 메시지를 작성합니다.

```
TextMessage outMessage = session.createTextMessage(outString);
```

메시지 및 메시지 본문에 대한 자세한 정보는 [140 페이지의 『JMS 메시지』](#)의 내용을 참조하십시오.

다음 예제에서 표시된 대로, 메시지를 송신하기 위해 애플리케이션은 MessageProducer 오브젝트의 send() 메소드를 사용합니다.

```
producer.send(outMessage);
```

애플리케이션은 send() 메소드를 사용하여 두 메시징 도메인 중 하나의 메시지를 송신할 수 있습니다. 목적지의 네이처는 사용되는 메시징 도메인을 판별합니다. 그러나 발행/구독 도메인에 특정한 MessageProducer의 하위 인터페이스인 TopicPublisher에는 send() 메소드 대신 사용될 수 있는 publish() 메소드도 있습니다. 두 메소드는 기능적으로 동일합니다.

애플리케이션은 지정된 목적지 없이 MessageProducer 오브젝트를 작성할 수 있습니다. 이 경우에 애플리케이션은 send() 메소드를 호출할 때 목적지를 지정해야 합니다.

애플리케이션이 트랜잭션 내에서 메시지를 송신하는 경우, 메시지는 트랜잭션이 커밋될 때까지 해당 목적지에 전달되지 않습니다. 이는 동일 애플리케이션이 트랜잭션 내에서 메시지를 송신하고 메시지에 대한 응답을 수신할 수 없음을 의미합니다.

애플리케이션이 메시지를 송신할 때 큐 관리자가 메시지를 안전하게 수신하는지 여부를 판별함이 없이 IBM MQ classes for JMS가 메시지를 전달하고 제어를 다시 애플리케이션으로 리턴할 수 있도록 목적지를 구성할 수 있습니다. 이를 종종 비동기 넣기이라고 합니다. 자세한 정보는 [294 페이지의 『IBM MQ classes for JMS에서 비동기로 메시지 넣기』](#)의 내용을 참조하십시오.

JMS 애플리케이션에서 메시지 수신

애플리케이션은 메시지 이용자를 사용하여 메시지를 수신합니다. 지속 가능한 토픽 구독자는 이용자가 비활성인 동안 송신된 메시지를 포함하여 목적지에 송신된 모든 메시지를 수신하는 메시지 이용자입니다. 애플리케이션은 메시지 선택자를 사용하여 수신을 원하는 메시지를 선택할 수 있으며, 메시지 리스너를 사용하여 비동기로 메시지를 수신할 수 있습니다.

애플리케이션은 `MessageConsumer` 오브젝트를 사용하여 메시지를 수신합니다. 메시지 이용자를 사용하여 수신된 모든 메시지가 동일한 목적지로부터 수신될 수 있도록, 애플리케이션은 큐 또는 토픽이 될 수 있는 특정 목적지에 대한 `MessageConsumer` 오브젝트를 작성합니다. 따라서 애플리케이션이 `MessageConsumer` 오브젝트를 작성할 수 있으려면 우선 애플리케이션이 `Queue` 또는 `Topic` 오브젝트를 작성해야 합니다. `Queue` 또는 `Topic` 오브젝트 작성에 대한 정보는 다음 주제를 참조하십시오.

- [199 페이지의 『JNDI를 사용하여 또는 Jakarta Messaging JMS 애플리케이션에서 관리되는 객체를 검색하기』](#)
- [201 페이지의 『IBM JMS 확장 사용』](#)
- [207 페이지의 『IBM MQ JMS 확장 사용』](#)
- [213 페이지의 『JMS 애플리케이션에서 목적지 작성』](#)

다음 예제에서 표시된 대로, `MessageConsumer` 오브젝트를 작성하기 위해 애플리케이션은 `Session` 오브젝트의 `createConsumer()` 메소드를 사용합니다.

```
MessageConsumer consumer = session.createConsumer(destination);
```

`destination` 매개변수는 애플리케이션이 이전에 작성한 큐 또는 토픽 오브젝트입니다.

그리고 애플리케이션은 다음 예제에서 표시된 대로 `MessageConsumer` 오브젝트의 `receive()` 메소드를 사용하여 목적지로부터 메시지를 수신합니다.

```
Message inMessage = consumer.receive(1000);
```

`receive()` 호출의 매개변수는 메시지가 즉시 사용 가능하지 않은 경우 메소드가 적당한 메시지의 도착을 대기하는 시간(밀리초)을 지정합니다. 매개변수를 생략하는 경우, 적당한 메시지가 도착할 때까지 호출이 무제한 차단됩니다. 애플리케이션이 메시지를 대기하지 않도록 하려면 `receiveNoWait()` 메소드를 대신 사용하십시오.

`receive()` 메소드는 특정 유형의 메시지를 리턴합니다. 예를 들어, 애플리케이션이 텍스트 메시지를 수신할 때 `receive()` 호출에 의해 리턴된 오브젝트는 `TextMessage` 오브젝트입니다.

그러나 `receive()` 호출에 의해 리턴된 오브젝트의 선언 유형은 `Message` 오브젝트입니다. 따라서 방금 수신된 메시지의 본문으로부터 데이터를 추출하려면 애플리케이션이 `Message` 클래스에서 보다 특정한 서브클래스(예: `TextMessage`)로 캐스팅해야 합니다. 메시지의 유형이 알려져 있지 않은 경우, 애플리케이션은 `instanceof` 연산자를 사용하여 유형을 판별할 수 있습니다. 오류를 잘 핸들링할 수 있도록 애플리케이션이 캐스팅 전에 메시지 유형을 판별하는 것이 언제나 바람직합니다.

다음 코드는 `instanceof` 연산자를 사용하며, 텍스트 메시지의 본문에서 데이터를 추출하는 방법을 표시합니다.

```
if (inMessage instanceof TextMessage) {
    String replyString = ((TextMessage) inMessage).getText();
    .
    .
} else {
    // Print error message if Message was not a TextMessage.
    System.out.println("Reply message was not a TextMessage");
}
```

애플리케이션이 트랜잭션 내에서 메시지를 송신하는 경우, 메시지는 트랜잭션이 커밋될 때까지 해당 목적지에 전달되지 않습니다. 이는 동일 애플리케이션이 트랜잭션 내에서 메시지를 송신하고 메시지에 대한 응답을 수신할 수 없음을 의미합니다.

미리 읽기에 대해 구성된 목적지로부터 메시지 이용자가 메시지를 수신하는 경우, 애플리케이션이 종료될 때 미리 읽기 버퍼에 있는 비지속 메시지는 버려집니다.

발행/구독 도메인에서 JMS는 두 가지 유형의 메시지 이용자, 지속 불가능한 토픽 구독자 및 지속 가능한 토픽 구독자를 식별하며, 이는 다음의 두 절에서 설명되어 있습니다.

지속 불가능한 토픽 구독자

지속 불가능한 토픽 구독자는 구독자가 활성인 동안 발행된 해당 메시지만 수신합니다. 지속 불가능한 구독은 애플리케이션이 지속 불가능한 토픽 구독자를 작성할 때 시작되며, 애플리케이션이 구독자를 닫을 때나 구독자가

범위를 벗어날 때 종료됩니다. IBM MQ classes for JMS의 확장으로서, 지속 불가능한 토픽 구독자는 보유된 발행도 수신합니다.

지속 불가능한 토픽 구독자를 작성하기 위해, 애플리케이션은 목적지로서 토픽 오브젝트를 지정하여 도메인 독립적인 `createConsumer()` 메소드를 사용할 수 있습니다. 또는 다음 예제에서 표시된 대로 애플리케이션은 도메인 특정 `createSubscriber()` 메소드를 사용할 수 있습니다.

```
TopicSubscriber subscriber = session.createSubscriber(topic);
```

`topic` 매개변수는 애플리케이션이 이전에 작성한 토픽 오브젝트입니다.

지속 가능한 토픽 구독자

제한사항: 브로커에 대한 실시간 연결을 사용 중일 때 애플리케이션은 지속 가능한 토픽 구독자를 작성할 수 없습니다.

지속 가능한 토픽 구독자는 지속 가능한 구독의 수명 중에 발행된 모든 메시지를 수신합니다. 이러한 메시지에는 구독자가 활성이 아닌 동안 발행된 모든 메시지가 포함됩니다. IBM MQ classes for JMS의 확장으로서, 지속 가능한 토픽 구독자는 보유된 발행도 수신합니다.

다음 예제에서 표시된 대로, 지속 가능한 토픽 구독자를 작성하기 위해 애플리케이션은 `Session` 오브젝트의 `createDurableSubscriber()` 메소드를 사용합니다.

```
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001");
```

`createDurableSubscriber()` 호출에서, 첫 번째 매개변수는 애플리케이션이 이전에 작성한 토픽 오브젝트이며 두 번째 매개변수는 지속 가능한 구독을 식별하는 데 사용된 이름입니다.

지속 가능한 토픽 구독자를 작성하기 위해 사용되는 세션에는 연관된 클라이언트 ID가 있어야 합니다. 세션과 연관된 클라이언트 ID는 세션 작성에 사용된 연결에 대한 클라이언트 ID와 동일합니다. 클라이언트 ID는 `ConnectionFactory` 오브젝트의 `CLIENTID` 특성을 설정하여 지정될 수 있습니다. 또는 애플리케이션은 연결 오브젝트의 `setClientID()` 메소드를 호출하여 클라이언트 ID를 지정할 수 있습니다.

지속 가능한 구독을 식별하는 데 사용되는 이름은 클라이언트 ID 내에서만 고유해야 하므로, 클라이언트 ID는 지속 가능한 구독의 전체, 고유 ID의 일부를 구성합니다. 이전에 작성된 지속 가능한 구독을 계속 사용하려면, 애플리케이션이 지속 가능한 구독과 연관된 것과 동일한 클라이언트 ID의 세션을 사용하고 동일한 구독 이름을 사용하여 지속 가능한 토픽 구독자를 작성해야 합니다.

지속 가능한 구독은 지속 가능한 구독이 현재 존재하지 않는 클라이언트 ID 및 구독 이름을 사용하여 애플리케이션이 지속 가능한 토픽 구독자를 작성할 때 시작됩니다. 그러나 애플리케이션이 지속 가능한 토픽 구독자를 닫을 때는 지속 가능한 구독이 종료되지 않습니다. 지속 가능한 구독을 종료하려면, 애플리케이션이 지속 가능한 구독과 연관된 것과 동일한 클라이언트 ID를 갖는 `Session` 오브젝트의 `unsubscribe()` 메소드를 호출해야 합니다. `unsubscribe()` 호출의 매개변수는 다음 예제에서 표시된 대로 구독 이름입니다.

```
session.unsubscribe("D_SUB_000001");
```

지속 가능한 구독의 범위는 큐 관리자입니다. 지속 가능한 구독이 큐 관리자에 존재하며 다른 큐 관리자에 연결된 애플리케이션이 클라이언트 ID 및 구독 이름이 동일한 지속 가능한 구독을 작성하는 경우, 두 개의 지속 가능한 구독은 완전히 독립적입니다.

메시지 선택자

애플리케이션은 특정 기준을 충족하는 해당 메시지만 연속 `receive()` 호출에 의해 리턴됨을 지정할 수 있습니다. `MessageConsumer` 오브젝트를 작성하는 경우, 애플리케이션은 검색되는 메시지를 판별하는 SQL(Structured Query Language) 표현식을 지정할 수 있습니다. 이 SQL 표현식을 메시지 선택자라고 합니다. 메시지 선택자에는 JMS 메시지 헤더 필드 및 메시지 특성의 이름이 포함될 수 있습니다. 메시지 선택자를 구성하는 방법에 대한 정보는 140 페이지의 『JMS의 메시지 선택자』의 내용을 참조하십시오.

다음 예제는 `myProp`라고 하는 사용자 정의 특성을 기반으로 애플리케이션이 메시지를 선택할 수 있는 방법을 표시합니다.

```
MessageConsumer consumer;
.
consumer = session.createConsumer(destination, "myProp = 'blue'");
```

JMS 스펙은 애플리케이션이 메시지 이용자의 메시지 선택자를 변경할 수 있도록 허용하지 않습니다. 애플리케이션이 메시지 선택자로 메시지 이용자를 작성한 이후, 메시지 선택자는 해당 이용자의 수명 중에 유지됩니다. 애플리케이션이 둘 이상의 메시지 선택자를 필요로 하는 경우, 애플리케이션은 각 메시지 선택자에 대한 메시지 이용자를 작성해야 합니다.

참고로, 애플리케이션이 버전 7 큐 관리자에 연결된 경우에 연결 팩토리의 MSGSELECTION 특성에는 영향이 없습니다. 성능 최적화를 위해 모든 메시지 선택은 큐 관리자에 의해 수행됩니다.

로컬 발행 억제

애플리케이션은 이용자의 자체 연결에서 발행된 발행을 무시하는 메시지 이용자를 작성할 수 있습니다. 다음 예제에서 표시된 대로, 애플리케이션은 createConsumer() 호출의 세 번째 매개변수를 true로 설정하여 이를 수행합니다.

```
MessageConsumer consumer = session.createConsumer(topic, null, true);
```

createDurableSubscriber() 호출에서, 애플리케이션은 다음 예제에서 표시된 대로 네 번째 매개변수를 true로 설정하여 이를 수행합니다.

```
String selector = "company = 'IBM'";
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001",
                                                             selector, true);
```

메시지의 비동기 전달

애플리케이션은 메시지 이용자에서 메시지 리스너를 등록하여 비동기적으로 메시지를 수신할 수 있습니다. 메시지 리스너에는 onMessage라고 하는 메소드가 있으며, 이는 적합한 메시지가 사용 가능하며 해당 용도가 메시지 처리인 경우에 비동기적으로 호출됩니다. 다음 코드는 해당 메커니즘을 설명합니다.

```
JM 3.0
import jakarta.jms.*;

public class MyClass implements MessageListener
{
    // The method that is called asynchronously when a suitable message is available
    public void onMessage(Message message)
    {
        System.out.println("Message is "+message);

        // The code to process the message
        .
        .
        .
    }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

```
JMS 2.0
import javax.jms.*;

public class MyClass implements MessageListener
{
    // The method that is called asynchronously when a suitable message is available
```



```

    public void onMessage(Message message)
    {
        System.out.println("Message is "+message);

        // The code to process the message
        .
        .
        .
    }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing

```

애플리케이션은 `receive()` 호출을 사용하여 동기적으로 메시지를 수신하거나 메시지 리스너를 사용하여 비동기적으로 메시지를 수신하기 위해 세션을 사용할 수 있지만, 둘 모두에는 적용되지 않습니다. 애플리케이션이 동기 및 비동기적으로 메시지를 수신해야 하는 경우, 이는 별도의 세션을 작성해야 합니다.

일단 메시지를 비동기적으로 수신하도록 세션이 설정된 경우, 다음 메소드는 해당 세션에서 또는 해당 세션에서 작성된 오브젝트에서 호출될 수 없습니다.

- `MessageConsumer.receive()`
- `MessageConsumer.receive(long)`
- `MessageConsumer.receiveNoWait()`
- `Session.acknowledge()`
- `MessageProducer.send(Destination, Message)`
- `MessageProducer.send(Destination, Message, int, int, long)`
- `MessageProducer.send(Message)`
- `MessageProducer.send(Message, int, int, long)`
- `MessageProducer.send(Destination, Message, CompletionListener)`
- `MessageProducer.send(Destination, Message, int, int, long, CompletionListener)`
- `MessageProducer.send(Message, CompletionListener)`
- `MessageProducer.send(Message, int, int, long, CompletionListener)`
- `Session.commit()`
- `Session.createBrowser(Queue)`
- `Session.createBrowser(Queue, String)`
- `Session.createBytesMessage()`
- `Session.createConsumer(Destination)`
- `Session.createConsumer(Destination, String, boolean)`
- `Session.createDurableSubscriber(Topic, String)`
- `Session.createDurableSubscriber(Topic, String, String, boolean)`
- `Session.createMapMessage()`
- `Session.createMessage()`
- `Session.createObjectMessage()`
- `Session.createObjectMessage(Serializable)`
- `Session.createProducer(Destination)`
- `Session.createQueue(String)`

- Session.createStreamMessage()
- Session.createTemporaryQueue()
- Session.createTemporaryTopic()
- Session.createTextMessage()
- Session.createTextMessage(String)
- Session.createTopic()
- Session.getAcknowledgeMode()
- Session.getMessageListener()
- Session.getTransactioned()
- Session.rollback()
- Session.unsubscribe(String)

이러한 메소드가 호출되면 다음 메시지가 포함된 `JMSEException`이 전달됩니다.

JMSCC0033: 세션이 비동기적으로 사용 중인 경우 동기식 메서드 호출은 허용되지 않습니다: '메서드 이름' is thrown.

변조 메시지 수신

애플리케이션은 처리될 수 없는 메시지를 수신할 수 있습니다. 메시지가 처리될 수 없는 여러 이유가 있을 수 있습니다(예: 메시지의 형식이 올바르지 않음). 해당 메시지는 변조 메시지로 설명되며, 메시지가 반복적으로 처리되지 않도록 특별한 핸들링이 필요합니다.

변조 메시지를 핸들링하는 방법에 대한 세부사항은 [224 페이지의 『IBM MQ classes for JMS에서 변조 메시지 핸들링』](#)의 내용을 참조하십시오.

수신되는 메시지에 맞게 버퍼 크기를 조정하기

비용용 JMS 프로그램으로부터 메시지를 IBM MQ 수신할 때, 해당 메시지를 기록하기 위해 응용 프로그램이 메시지 버퍼를 제공해야 합니다. JMS 애플리케이션은 수동으로 버퍼를 생성할 필요가 없습니다. 수신되는 메시지의 크기에 맞게 메시지 버퍼를 IBM MQ classes for JMS 자동으로 생성하고 크기를 조정합니다. 대부분의 애플리케이션에서는 자동으로 관리되는 버퍼가 애플리케이션 개발자에게 성능과 편의성 사이의 적절한 균형을 제공합니다. 특정 상황에서는 메시지 버퍼의 초기 크기를 수동으로 지정하는 것이 유리할 수 있습니다. 수신 IBM MQ JMS 버퍼의 기본 초기 크기는 4KB입니다. 응용 프로그램이 항상 256KB 크기의 메시지를 수신할 예정이라면, 초기 버퍼 크기를 256KB로 설정하는 것이 바람직할 수 있습니다. 이렇게 하면 메시지를 4KB 버퍼로 수신하려는 IBM MQ classes for JMS 시도가 실패한 후, 버퍼 크기를 256KB로 확장하여 성공적으로 수신하는 과정을 피할 수 있습니다. 클라이언트 연결 애플리케이션의 경우, 이는 올바른 버퍼 크기를 결정하는 IBM MQ classes for JMS 동안 잠재적으로 낭비될 수 있는 네트워크 왕복 통신을 피할 수 있습니다.

초기 버퍼 크기는 `com.ibm.mq.jmqi.defaultMaxMsgSizeJava` 속성을 원하는 값(바이트 단위)으로 설정하여 구성할 수 있습니다. 이 속성은 내부에서 실행 중인 모든 IBM MQ JMS Java Virtual Machine 애플리케이션에 영향을 미치므로, 다른 크기의 메시지를 수신하는 다른 메시지 소비자들에 부정적인 영향을 미치지 않도록 주의하십시오.

여러 개의 메시지가 설정된 크기보다 작게 수신될 경우, 버퍼 크기를 자동으로 줄이려는 시도가 IBM MQ classes for JMS 여전히 이루어집니다. 기본적으로, 버퍼 크기보다 작은 메시지가 10개 수신되면 이 현상이 발생합니다. 예를 들어, 128KB 크기의 메시지가 연속으로 10개 수신되면 버퍼 크기가 256KB에서 128KB로 감소합니다. 그런 다음 더 큰 메시지가 수신되면 다시 증가합니다. 버퍼 크기가 축소되기 전에 수신해야 하는 메시지 수를 구성할 수 있습니다. 예를 들어, 애플리케이션이 5개의 큰 메시지를 수신한 후 10개의 작은 메시지를 수신하고 다시 5개의 큰 메시지를 수신하는 패턴으로 동작하는 것으로 알려진 경우에 유용할 수 있습니다. 기본 설정에서는 10개의 작은 메시지를 수신한 후 버퍼가 축소되며, 더 큰 메시지를 위해 다시 증가시켜야 합니다. 시스템 속성 `com.ibm.mq.jmqi.smallMsgBufferReductionThresholdJava` 은 버퍼 크기가 축소되기 전에 수신해야 하는 메시지 수로 설정할 수 있습니다. 이 예시에서는 10개의 작은 메시지로 인해 버퍼 크기가 줄어드는 것을 방지하기 위해 20으로 설정할 수 있습니다.

속성은 서로 독립적으로 설정할 수 있습니다. 예를 들어, 초기 버퍼 크기는 기본값인 4KB로 유지하되, 버퍼 크기가 증가된 후 그 크기를 더 오래 유지하도록 버퍼 유지 시간 (com.ibm.mq.jmqi.smallMsgBufferReductionThreshold) 값을 늘릴 수 있습니다.

MQI의 통계 기록에서 해당 JMS 애플리케이션에 대해 MQRC_TRUNCATED_MSG_FAILED (2080) 반환 코드가 다수 확인된다면, 해당 애플리케이션의 초기 버퍼 크기를 늘리거나 버퍼 크기가 축소되는 빈도를 줄이는 것이 도움이 될 수 있음을 시사하는 것입니다. 그러나 장기 실행 애플리케이션의 경우

MQRC_TRUNCATED_MSG_FAILED 반환 코드가 극히 드물게 발생할 것임을 유의해야 합니다. 이는 일반적으로 버퍼가 첫 번째 대용량 메시지를 수신한 직후 올바른 크기로 증가되며, 여러 개의 소용량 메시지를 수신하지 않는 한 크기가 축소되지 않기 때문입니다. 따라서 다수의 MQRC_TRUNCATED_MSG_FAILED 오류는 단 한두 개의 메시지만 수신한 IBM MQ 후 연결을 끊는 등의 다른 부적절한 애플리케이션 관행을 나타낼 가능성이 있습니다.

구독 사용자 데이터의 검색

IBM MQ classes for JMS 애플리케이션이 큐에서 사용하는 메시지가 관리적으로 정의된 지속 가능한 구독에 의해 넣어지면 애플리케이션은 구독과 관련된 사용자 데이터 정보에 액세스해야 합니다. 이 정보는 특성으로 메시지에 추가됩니다.

RFH2 헤더에 MQPS 폴더가 포함된 큐에서 메시지가 소비될 때, Sud 키와 연관된 값이 존재하면 해당 값이 문자열 속성으로 애플리케이션에 IBM MQ classes for JMS 반환되는 Message JMS 객체에 추가됩니다. 메시지에서 이 속성을 검색하려면, JmsConstants 인터페이스의 상수 JMS_IBM_SUBSCRIPTION_USER_DATA를 사용하여 다음 메서드로 구독 사용자 데이터를 얻을 수 있습니다:

- ▶ **JM 3.0** `jakarta.jms.Message.getStringProperty(java.lang.String)`
- ▶ **JMS 2.0** `javax.jms.Message.getStringProperty(java.lang.String)`

다음 예시에서는 MQSC 명령어를 사용하여 관리형 지속 **DEFINE SUB** 구독을 정의합니다:

```
DEFINE SUB('MY.SUBSCRIPTION') TOPICSTR('PUBLIC') DEST('MY.SUBSCRIPTION.Q')
USERDATA('Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q')
```

토픽 문자열 PUBLIC에 발행된 메시지 사본이 MY.SUBSCRIPTION.Q 큐에 들어갑니다. 지속 가능한 구독과 관련된 사용자 데이터는 RFH2 헤더의 MQPS 폴더에 Sud 키와 함께 저장되어 있는 메시지에 특성으로 추가됩니다.

IBM MQ classes for JMS 애플리케이션은 다음을 호출할 수 있습니다.

```
▶ JM 3.0 jakarta.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

```
▶ JMS 2.0 javax.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

그러면 다음 문자열이 리턴됩니다.

```
Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q
```

관련 개념

[146 페이지의 『MQRFH2 헤더 및 JMS』](#)

관련 태스크

[관리 구독 정의](#)

관련 참조

[DEFINE SUB](#)

[인터페이스 JmsConstants](#)

IBM MQ classes for JMS 닫기

중지 전에 IBM MQ classes for JMS 애플리케이션이 특정 JMS 오브젝트를 명시적으로 닫는 것이 중요합니다. 파일널라이저가 호출되지 않을 수 있으므로, 이에 의존하여 자원을 해제하지 마십시오. 애플리케이션이 압축 추적 활성으로 종료되도록 허용하지 마십시오.

가비지 콜렉션만으로는 모든 IBM MQ classes for JMS 및 IBM MQ 자원을 적시에 해제할 수 없습니다. 특히 애플리케이션이 세션 레벨 이하에서 수명이 짧은 많은 JMS 오브젝트를 작성하는 경우에는 더욱 그렇습니다. 따라

서 더 이상 필요하지 않을 때 애플리케이션이 Connection, Session, MessageConsumer 또는 MessageProducer 오브젝트를 닫는 것이 중요합니다.

Connection을 닫지 않고 애플리케이션이 종료되면 모든 연결의 트랜잭션 세션에 대해 암시적 롤백이 발생합니다. 애플리케이션이 작성한 변경사항이 커밋되는지 확인하려면, 애플리케이션을 닫기 전에 Connection을 명시적으로 닫으십시오.

애플리케이션에서 파이널라이저를 사용하여 JMS 오브젝트를 닫지 마십시오. 파이널라이저가 호출되지 않을 수 있으므로 자원을 해제하지 못할 수 있습니다. Connection이 닫히면 이로부터 작성된 모든 Session이 닫힙니다. 이와 유사하게, Session에서 작성된 MessageConsumers 및 MessageProducers는 Session이 닫힐 때 닫힙니다. 그러나 자원이 시기적절하게 비워지는지 확인할 수 있도록 Sessions, MessageConsumers 및 MessageProducers를 명시적으로 닫는 것을 고려하십시오.

추적 압축이 활성화된 경우, System.Halt() 종료 및 비정상, 비제어 JVM 종료는 결과적으로 추적 파일의 손상을 유발할 수 있습니다. 가급적이면, 필요한 추적 정보가 수집된 경우에는 추적 기능을 끄십시오. 비정상 종료까지 애플리케이션을 추적하는 경우에는 압축되지 않은 추적 출력을 사용하십시오.

참고: 큐 관리자와의 연결을 끊기 위해 JMS 애플리케이션은 연결 오브젝트에서 close() 메소드를 호출합니다.

IBM MQ classes for JMS에서 변조 메시지 핸들링

독성 메시지는 수신 애플리케이션이 처리할 수 없는 메시지를 의미합니다. 예를 들어 메시지의 유형이 예상치 못했거나, 애플리케이션의 로직이 처리할 수 없는 정보를 포함하는 경우 등이 해당됩니다. 애플리케이션이 메시지를 처리할 수 없는 경우, 해당 메시지를 원래 있던 큐로 롤백합니다. 기본적으로 중독된 메시지를 애플리케이션에 반복적으로 IBM MQ classes for JMS 재전송하며, 이로 인해 애플리케이션이 중독된 메시지를 처리하고 롤백하려는 루프에 계속 갇힐 수 있습니다. 이를 방지하기 위해, 애플리케이션에 전달된 후 지정된 횟수만큼 롤백된 독성 메시지를 백아웃 큐로 이동시킬 수 IBM MQ classes for JMS 있습니다.

독 메시지를 감지하고, 다음 속성을 활용하여 대체 목적지로 이동시킬 수 있습니다 IBM MQ classes for JMS :

- 감지된 메시지의 MQMD 내에 있는 BackoutCount 필드 값
- 메시지가 포함되어 있는 입력 큐의 IBM MQ 큐 속성 **BOTHRESH**(백아웃 임계값) 및 **BOQNAME**(백아웃 큐)

및 **BOQNAMEBOTHRESH** 속성을 설정하려면 다음 MQSC 명령을 실행하십시오:

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

애플리케이션에서 메시지를 롤백할 때마다 큐 관리자가 메시지의 BackoutCount 필드 값을 자동으로 증분시킵니다.

IBM MQ classes for JMS에서 BackoutCount 값이 0보다 큰 메시지를 감지할 경우, BackoutCount 값을 **BOTHRESH** 속성의 값에 비교합니다.

- BackoutCount가 **BOTHRESH** 속성 값보다 작을 경우, IBM MQ classes for JMS에서는 처리를 위해 이를 애플리케이션에 전달합니다.
- 그러나 BackoutCount가 **BOTHRESH**보다 크거나 같을 경우 이 메시지는 변조 메시지로 간주됩니다. 이 경우 IBM MQ classes for JMS에서는 메시지를 **BOQNAME** 속성에 지정된 큐로 이동시킵니다. 메시지를 백아웃 큐에 넣을 수 없는 경우, 메시지의 보고 옵션에 따라 큐 관리자의 불량 메일 큐로 이동되거나 폐기됩니다.

참고:

- **BOTHRESH** 속성이 기본값 0일 경우 변조 메시지 핸들링이 사용 안함으로 설정됩니다. 즉, 변조 메시지가 다시 입력 큐에 배치됩니다.
- 주의해야 할 다른 사항은 IBM MQ classes for JMS에서 BackoutCount가 0보다 큰 메시지를 처음 감지할 때 큐의 **BOTHRESH** 및 **BOQNAME** 속성을 조회된다는 점입니다. 이러한 속성 값은 캐시된 후, IBM MQ classes for JMS에서 BackoutCount가 0보다 큰 메시지를 발견할 때마다 사용됩니다.

변조 메시지 핸들링을 수행하도록 시스템 구성

IBM MQ classes for JMS에서 **BOTHRESH** 및 **BOQNAME** 속성을 조회할 때 사용할 큐는 수행하는 메시징 스타일에 따라 달라집니다.

- 포인트-투-포인트 메시징의 경우, 이는 기본 로컬 큐입니다. JMS 애플리케이션이 알리어스 큐 또는 클러스터 큐의 메시지를 이용할 경우에 중요합니다.
- 발행/구독 메시징의 경우, 애플리케이션의 메시지를 보관하기 위해 관리 큐가 작성됩니다. IBM MQ classes for JMS에서는 관리 큐를 조회하여 **BOTHRESH** 및 **BOQNAME** 속성의 값을 판별합니다.

관리 큐는 애플리케이션이 구독하는 토픽 오브젝트와 연관된 모델 큐에서 작성되며, **BOTHRESH** 및 **BOQNAME** 속성의 값을 모델 큐로부터 상속받습니다. 수신 애플리케이션이 지속 가능 구독 또는 지속 불가능 구독을 제공했는지 여부에 따라 사용되는 모델 큐가 달라집니다.

- 지속 가능 구독에 사용되는 모델 큐는 토픽의 **MDURMDL** 속성에 지정됩니다. 이 속성의 기본값은 **SYSTEM.DURABLE.MODEL.QUEUE**입니다.
- 지속 불가능 구독에 사용되는 모델 큐는 **MNDURMDL** 속성에 지정됩니다. **MNDURMDL** 속성의 기본값은 **SYSTEM.NDURABLE.MODEL.QUEUE**입니다.

BOTHRESH 및 **BOQNAME** 속성을 조회할 때 IBM MQ classes for JMS에서는 다음을 수행합니다.

- 로컬 큐 또는 알리어스 큐의 대상 큐를 엽니다.
- **BOTHRESH** 및 **BOQNAME** 속성을 조회합니다.
- 로컬 큐 또는 알리어스 큐의 대상 큐를 닫습니다.

로컬 큐 또는 알리어스 큐의 대상 큐를 열 때 사용되는 열기 옵션은 사용 중인 IBM MQ classes for JMS의 버전에 따라 다릅니다.

- for IBM MQ classes for JMS in IBM MQ 9.1.0 Fix Pack 1 and earlier, or IBM MQ 9.1.1, 만약 로컬 큐 또는 별칭 큐의 대상 큐가 클러스터 큐인 경우, **MQ00_INPUT_AS_Q_DEF**, **MQ00_INQUIRE** 및 **MQ00_FAIL_IF QUIESCING** 옵션으로 큐를 IBM MQ classes for JMS 엽니다. 즉, 수신 애플리케이션을 실행하는 사용자가 클러스터 큐의 로컬 인스턴스를 조회하고 이 인스턴스에 액세스할 수 있는 권한을 갖고 있어야 합니다.

IBM MQ classes for JMS의 경우 열기 옵션 **MQ00_INQUIRE** 및 **MQ00_FAIL_IF QUIESCING**을 사용하여 로컬 큐의 다른 모든 유형을 엽니다. IBM MQ classes for JMS에서 속성 값을 조회하려면 수신 애플리케이션을 실행하는 사용자가 로컬 큐에 대한 조회 액세스 권한을 갖고 있습니다.

- in IBM MQ 9.1.0 Fix Pack 2 및 이후 버전을 사용하거나 IBM MQ classes for JMS, in 및 이후 IBM MQ 9.1.2 버전을 위한 경우, 수신 애플리케이션을 실행하는 사용자는 큐의 유형과 관계없이 로컬 큐에 대한 **inquire** 접근 권한을 가져야 합니다.

독성 메시지를 백아웃 재큐잉 큐 또는 큐 관리자의 불활성 메시지 큐로 이동시키려면, 애플리케이션을 실행하는 사용자에게 **put** 권한과 **passall** 권한을 부여해야 합니다.

동기 애플리케이션의 변조 메시지 처리

애플리케이션이 다음 방법 중 하나를 호출하여 메시지를 동시에 수신할 경우 IBM MQ classes for JMS에서는 애플리케이션이 메시지를 가져오려고 할 때 활성 상태인 작업 단위 내에서 변조 메시지를 리큐잉합니다.

- **JMSConsumer.receive()**
- **JMSConsumer.receive(long timeout)**
- **JMSConsumer.receiveBody(Class<T> c)**
- **JMSConsumer.receiveBody(Class<T> c, long timeout)**
- **JMSConsumer.receiveBodyNoWait Class<T> c)**
- **JMSConsumer.receiveNoWait()**
- **MessageConsumer.receive()**
- **MessageConsumer.receive(long timeout)**
- **MessageConsumer.receiveNoWait()**
- **QueueReceiver.receive()**
- **QueueReceiver.receive(long timeout)**

- QueueReceiver.receiveNoWait()
- TopicSubscriber.receive()
- TopicSubscriber.receive(long timeout)
- TopicSubscriber.receiveNoWait()

즉, 애플리케이션이 트랜잭션 JMS 컨텍스트 또는 세션 중 하나를 사용 중인 경우 트랜잭션이 커밋될 때까지 백아웃 큐로의 메시지 이동이 커밋되지 않습니다.

BOTHRESH 속성이 0 이외의 값으로 설정된 경우, **BOQNAME** 속성도 설정해야 합니다. **BOTHRESH**가 0보다 큰 값으로 설정되어 있고 **BOQNAME**이 설정되지 않은 경우, 메시지의 보고 옵션에 따라 동작이 결정됩니다.

- 메시지에 보고 옵션 **MQRO_DISCARD_MSG**가 설정되어 있는 경우, 해당 메시지가 제거됩니다.
- 메시지에 **MQRO_DEAD_LETTER_Q** 보고 옵션이 지정된 경우, 메시지를 큐 관리자의 불활성 메시지 큐로 이동시키려고 시도합니다 IBM MQ classes for JMS .
- 메시지에 **MQRO_DISCARD_MSG** 또는 **MQRO_DEAD_LETTER_Q** 중 하나가 설정되어 있지 않은 경우 IBM MQ classes for JMS에서는 메시지를 큐 관리자의 데드-레터 큐에 넣으려고 시도합니다.

메시지를 데드-레터 큐에 넣으려는 시도가 어떤 이유로 실패할 경우, 수신 애플리케이션이 트랜잭션 또는 비트랜잭션 JMS 컨텍스트 또는 세션을 사용 중인지 여부에 따라 메시지에 발생하는 상황이 결정됩니다.

- 애플리케이션이 트랜잭션 JMS 컨텍스트 또는 세션 중 하나를 사용 중이고 트랜잭션이 커밋된 경우 메시지가 제거됩니다.
- 애플리케이션이 트랜잭션 JMS 컨텍스트 또는 세션을 사용 중이고 트랜잭션을 롤백하는 경우 메시지가 입력 큐로 돌아갑니다.
- 수신 애플리케이션에서 비트랜잭션 JMS 컨텍스트 또는 세션을 작성한 경우 메시지가 제거됩니다.

비동기 애플리케이션의 변조 메시지 처리

메시지 리스너를 통해 비동기적으로 메시지를 수신하는 애플리케이션의 경우, 메시지 전달에 영향을 주지 않으면서 유해 메시지를 IBM MQ classes for JMS 재큐잉합니다. 재대기 처리 과정은 애플리케이션에 대한 실제 메시지 전달과 관련된 작업 단위 외부에서 수행됩니다.

BOTHRESH가 0보다 큰 값으로 설정되어 있고 **BOQNAME**이 설정되지 않은 경우, 메시지의 보고 옵션에 따라 동작이 결정됩니다.

- 메시지에 보고 옵션 **MQRO_DISCARD_MSG**가 설정되어 있는 경우, 해당 메시지가 제거됩니다.
- 메시지에 **MQRO_DEAD_LETTER_Q** 보고 옵션이 지정된 경우, 메시지를 큐 관리자의 불활성 메시지 큐로 이동시키려고 시도합니다 IBM MQ classes for JMS .
- 메시지에 **MQRO_DISCARD_MSG** 또는 **MQRO_DEAD_LETTER_Q** 중 하나가 설정되어 있지 않은 경우 IBM MQ classes for JMS에서는 메시지를 큐 관리자의 데드-레터 큐에 넣으려고 시도합니다.

메시지를 데드-레터 큐에 넣으려는 시도가 어떤 이유로 실패할 경우, IBM MQ classes for JMS에서 메시지가 입력 큐로 돌아갑니다.

활성화 사양 및 연결 소비자에서 유독 메시지를 처리하는 방법에 대한 정보는 [ASF에서 큐에서 메시지 제거](#)를 참조하십시오.

메시지가 백아웃 큐로 이동할 때 메시지에 발생하는 상황

독성 메시지가 백아웃 재대기 큐에 재대기될 때, 해당 메시지에 RFH2IBM MQ classes for JMS 헤더를 추가하고 (기존에 없었던 경우), 메시지 디스크립터(MQMD) 내 일부 필드를 업데이트합니다.

독성 메시지에 RFH2 헤더가 포함된 경우(예: 해당 JMS 메시지가 메시지인 경우), 메시지를 백아웃 재대기열(backout re-queue)로 이동할 때 MQMD 내의 다음 필드를 변경합니다 IBM MQ classes for JMS :

- BackoutCount 필드가 0으로 재설정됩니다.
- 메시지의 만기필드는 변조 메시지가 JMS 애플리케이션에 의해 수신되었을 때 남은 만기 시간을 반영하도록 업데이트됩니다.

변조 메시지에 RFH2 헤더가 포함되어 있지 않은 경우 IBM MQ classes for JMS에서는 백아웃 처리의 일부로 이 헤더를 추가하고 MQMD 내의 다음 필드를 업데이트합니다.

- BackoutCount 필드가 0으로 재설정됩니다.
- 메시지의 만기필드는 변조 메시지가 JMS 애플리케이션에 의해 수신되었을 때 남은 만기 시간을 반영하도록 업데이트됩니다.
- 메시지의 형식 필드는 MQHRF2로 변경됩니다.
- CCSID 필드가 1208로 변경됩니다.
- 인코딩 필드가 273으로 수정됩니다.

이와 더불어, 악성 메시지의 CCSID 및 인코딩 필드가 RFH2 헤더의 CCSID 및 인코딩 필드로 복사되어, 백아웃 재 대기 큐에서 메시지의 헤더 체이닝이 올바르게 이루어지도록 합니다.

관련 개념

[311 페이지의 『ASF에서 변조 메시지 핸들링』](#)

ASF(Application Server Facilities)에서, 변조 메시지 핸들링은 IBM MQ classes for JMS의 다른 위치에서와는 조금 다르게 핸들링됩니다.

IBM MQ classes for JMS의 예외

IBM MQ classes for JMS 애플리케이션은 예외 핸들러에 전달되거나 JMS API 호출에서 발생시킨 예외를 처리해야 합니다.

IBM MQ classes for JMS는 예외를 발생시켜서 런타임 문제점을 보고합니다. 발생한 예외 유형 및 이러한 예외를 처리해야 하는 방법은 애플리케이션에서 사용하는 JMS 스펙의 버전에 따라 다릅니다.

- JMS 1.1 이하에서 정의된 인터페이스에서 메소드는 선택된 예외를 발생시킵니다. 이러한 예외의 기본 클래스는 JMSEException입니다. 선택된 예외를 처리하는 방법에 대한 자세한 정보는 [227 페이지의 『선택된 예외 처리』](#)의 내용을 참조하십시오.
- 추가된 인터페이스의 메서드들은 JMS 2.0 검사되지 않은 예외를 발생시킵니다. 이러한 예외의 기본 클래스는 JMSRuntimeException입니다. 선택 취소된 예외를 처리하는 방법에 대한 자세한 정보는 [230 페이지의 『선택 취소된 예외 처리』](#)의 내용을 참조하십시오.

또한 JMS Connection 또는 JMSContext로 ExceptionListener를 등록할 수 있습니다. 그런 다음, JMS 용 MQ 클래스는 메시지를 비동기로 전달하는 중에 문제가 발생하는 경우 또는 큐 관리자에 대한 연결에서 문제가 감지된 경우 ExceptionListener에 알리십시오. 자세한 정보는 [233 페이지의 『ExceptionListeners』](#)의 내용을 참조하십시오.

관련 개념

[JMS용 IBM MQ 클래스](#)

관련 참조

[ASYNCEXCEPTION](#)

선택된 예외 처리

JMS 1.1 이하에서 정의된 인터페이스에서 메소드는 선택된 예외를 발생시킵니다. 해당 예외의 기본 클래스는 JMSEException입니다. 따라서 JMSEExceptions을(를) 발견하면 해당 유형의 예외를 처리하는 일반적인 방법이 제공됩니다.

모든 JMSEException은(는) 다음 정보를 캡슐화합니다.

- 애플리케이션이 Throwable.getMessage() 메소드를 호출하여 얻을 수 있는 제공자별 예외 메시지입니다.
- 애플리케이션이 JMSEException.getErrorCode() 메소드를 호출하여 얻을 수 있는 제공자별 오류 코드입니다.
- 링크된 예외. JMS 1.1 API 호출로 발생된 예외는 종종 하위 레벨 문제점의 결과이며, 이는 이 예외에 링크된 다른 예외에서 보고됩니다. 애플리케이션은 JMSEException.getLinkedException() 메소드 또는 Throwable.getCause() 메소드를 호출하여 링크된 예외를 가져올 수 있습니다.

JMS 1.1 API를 사용하는 경우, IBM MQ classes for JMS에서 발생하는 대부분의 예외는 `JMSException`의 서브클래스 인스턴스입니다. 해당 서브클래스는 다음과 같은 추가 정보를 제공하는 `com.ibm.msg.client.jms.JmsExceptionDetail` 인터페이스를 구현합니다.

- 예외 메시지에 대한 설명. 애플리케이션은 `JmsExceptionDetail.getExplanation()` 메소드를 호출하여 이 메시지를 얻을 수 있습니다.
- 권장되는 예외에 대한 사용자 응답. 애플리케이션은 `JmsExceptionDetail.getUserAction()` 메소드를 호출하여 이 메시지를 얻을 수 있습니다.
- 예외 메시지에서 메시지 삽입을 위한 키. 애플리케이션은 `JmsExceptionDetail.getKeys()` 메소드를 호출하여 모든 키에 대한 반복자를 얻을 수 있습니다.
- 예외 메시지의 메시지 삽입. 예를 들어, 메시지 삽입이 예외를 발생시킨 큐의 이름일 수 있으며 이는 애플리케이션이 해당 이름에 액세스하도록 하는 데 유용할 수 있습니다. 애플리케이션은 `JmsExceptionDetail.getValue()` 메소드를 호출하여 지정된 키에 대응하는 메시지 삽입을 얻을 수 있습니다.

세부사항을 사용할 수 없으면 `JmsExceptionDetail` 인터페이스의 모든 메소드가 널을 리턴합니다.

예를 들어, 애플리케이션이 존재하지 않는 IBM MQ 큐에 대한 메시지 생성자를 작성하려고 시도하면 다음 정보와 함께 예외가 발생합니다.

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

던져진 예외는 다음 클래스의 하위 클래스이며, `com.ibm.msg.client.jms.DetailedInvalidDestinationException` `com.ibm.msg.client.jms.JmsExceptionDetail` 인터페이스를 구현합니다.

- **JM 3.0** `jakarta.jms.InvalidDestinationException`
- **JMS 2.0** `javax.jms.InvalidDestinationException`

링크된 예외

링크된 예외는 런타임 문제점에 대한 추가 정보를 제공합니다. 따라서 예외 처리된 `JMSException`마다 애플리케이션은 링크된 예외를 확인해야 합니다.

링크된 예외 자체에 또 다른 링크된 예외가 있을 수 있으므로, 링크된 예외는 원래의 기본 문제점으로 다시 되돌리는 체인을 작성합니다. 링크된 예외는 `java.lang.Throwable` 클래스의 체인된 예외 메커니즘을 사용하여 구현되며, 애플리케이션은 `Throwable.getCause()` 메소드를 호출하여 링크된 예외를 얻을 수 있습니다. `JMSException`의 경우 `getLinkedException()` 메소드는 `Throwable.getCause()` 메소드에 위임합니다.

예를 들어, 큐 관리자에 연결할 때 애플리케이션이 잘못된 포트 번호를 지정하면 예외에서 다음 체인을 작성합니다.

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

일반적으로, 체인의 각 예외는 코드의 다른 계층에서 발생합니다. 예를 들어, 선행 체인의 예외가 다음 계층에서 발생합니다.

- `JMSEException`의 서브클래스 인스턴스인 첫 번째 예외는 IBM MQ classes for JMS의 공통 계층을 통해 처리됩니다.
- `com.ibm.mq.MQException`의 인스턴스인 다음 예외는 IBM MQ 메시징 제공자를 통해 처리됩니다.
- 다음 두 예외는 모두의 인스턴스이며 `com.ibm.mq.jmqi.JmqiException`, 메시지 큐잉 인터페이스 Java (JMQUI)에 의해 발생합니다. JMQUI는 IBM MQ classes for JMS에서 큐 관리자와 통신하는 데 사용되는 컴포넌트입니다.
- `java.net.ConnectionException`의 인스턴스인 최종 예외는 Java 클래스 라이브러리를 통해 처리됩니다.

IBM MQ classes for JMS의 계층화된 아키텍처에 대한 자세한 정보는 [JMS용 IBM MQ 클래스 아키텍처](#)를 참조하십시오.

다음 예제와 같이 모든 적절한 정보를 추출하도록 이 체인을 통해 반복할 애플리케이션을 코드화할 수 있습니다.

JM 3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSEException;
.
.
.
catch (JMSEException je) {
    System.err.println("Caught JMSEException");
    // Check for linked exceptions in JMSEException
    Throwable t = je;
    while (t != null) {
        // Write out the message that is applicable to all exceptions
        System.err.println("Exception Msg: " + t.getMessage());
        // Write out the exception stack trace
        t.printStackTrace(System.err);

        // Add on specific information depending on the type of exception
        if (t instanceof JMSEException) {
            JMSEException je1 = (JMSEException) t;
            System.err.println("JMS Error code: " + je1.getErrorCode());
            if (t instanceof JmsExceptionDetail){
                JmsExceptionDetail jed = (JmsExceptionDetail)je1;
                System.err.println("JMS Explanation: " + jed.getExplanation());
                System.err.println("JMS Explanation: " + jed.getUserAction());
            }
        } else if (t instanceof MQException) {
            MQException mqe = (MQException) t;
            System.err.println("WMQ Completion code: " + mqe.getCompCode());
            System.err.println("WMQ Reason code: " + mqe.getReason());
        } else if (t instanceof JmqiException){
            JmqiException jmqie = (JmqiException)t;
            System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
            System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
            System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
            System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
            System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
        }
        // Get the next cause
        t = t.getCause();
    }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSEException;
.
.
.
catch (JMSEException je) {
    System.err.println("Caught JMSEException");
    // Check for linked exceptions in JMSEException
```

```

Throwable t = je;
while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSEException) {
        JMSEException je1 = (JMSEException) t;
        System.err.println("JMS Error code: " + je1.getErrorCode());
        if (t instanceof JmsExceptionDetail){
            JmsExceptionDetail jed = (JmsExceptionDetail)je1;
            System.err.println("JMS Explanation: " + jed.getExplanation());
            System.err.println("JMS Explanation: " + jed.getUserAction());
        }
    } else if (t instanceof MQException) {
        MQException mqe = (MQException) t;
        System.err.println("WMQ Completion code: " + mqe.getCompCode());
        System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
        JmqiException jmqie = (JmqiException)t;
        System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
        System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
        System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
        System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
        System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
}
}

```

참고로, 예외의 유형이 다양할 수 있으며 서로 다른 유형의 예외가 다른 정보를 캡슐화할 수 있으므로 애플리케이션은 항상 체인에서 각 예외의 유형을 확인해야 합니다.

문제점에 대한 IBM MQ 특정 정보 확보

`com.ibm.mq.MQException` 및 `com.ibm.mq.jmqi.JmqiException`의 인스턴스가 문제점에 대한 IBM MQ 고유 정보를 캡슐화합니다.

`MQException`에서는 다음 정보를 캡슐화합니다.

- 애플리케이션이 `getCompCode()` 메소드를 호출하여 얻을 수 있는 완료 코드.
- 애플리케이션이 `getReason()` 메소드를 호출하여 얻을 수 있는 이유 코드.

이러한 메소드를 사용하는 방법에 대한 예제는 [링크된 예외](#)의 샘플 코드를 참조하십시오.

`JmqiException`은(는) 완료 코드 및 이유 코드도 캡슐화합니다. 이 외에도 `JmqiException`에는 `AMQ nnnn` 또는 `CSQ nnnn` 메시지의 정보가 포함되어 있습니다(예외와 연관된 경우). 애플리케이션은 다음 메소드를 호출하여 이 메시지의 다양한 컴포넌트를 확보할 수 있습니다.

- `getWmqMsgExplanation()` 메소드는 `AMQ nnnn` 또는 `CSQ nnnn` 메시지에 대한 설명을 리턴합니다.
- `getWmqMsgSeverity()` 메소드는 `AMQ nnnn` 또는 `CSQ nnnn` 메시지의 심각도를 리턴합니다.
- `getWmqMsgSummary()` 메소드는 `AMQ nnnn` 또는 `CSQ nnnn` 메시지의 요약을 리턴합니다.
- `getWmqMsgUserResponse()` 메소드는 `AMQ nnnn` 또는 `CSQ nnnn` 메시지와 연관된 사용자 응답을 리턴합니다.

선택 취소된 예외 처리

JMS 2.0에 정의된 인터페이스에서 메소드는 선택 취소된 예외를 발생시킵니다. 해당 예외의 기본 클래스는 `JMSRuntimeException`입니다. 따라서 `JMSRuntimeExceptions`을(를) 발견하면 해당 유형의 예외를 처리하는 일반적인 방법이 제공됩니다.

모든 `JMSRuntimeException`은(는) 다음 정보를 캡슐화합니다.

- 애플리케이션이 `JMSRuntimeException.getMessage()` 메소드를 호출하여 얻을 수 있는 제공자별 예외 메시지입니다.

- 애플리케이션이 `JMSRuntimeException.getErrorCode()` 메소드를 호출하여 얻을 수 있는 제공자별 오류 코드입니다.
- 링크된 예외. JMS 2.0 API 호출로 발생한 예외는 종종 하위 레벨 문제점의 결과이며, 이는 이 예외에 링크된 다른 예외에서 보고됩니다. 애플리케이션은 `JMSRuntimeException.getCause()` 메소드를 호출하여 링크된 예외를 가져올 수 있습니다.

API가 JMS 2.0 제공하는 인터페이스의 메서드를 호출할 때, 해당 인터페이스가 던지는 대부분의 예외는 의 IBM MQ classes for JMS 하위 클래스 `JMSRuntimeException` 인스턴스입니다. 이러한 서브클래스는 다음과 같은 추가 정보를 제공하는 `com.ibm.msg.client.jms.JmsExceptionDetail` 인터페이스를 구현합니다.

- 예외 메시지에 대한 설명. 애플리케이션은 `JmsExceptionDetail.getExplanation()` 메소드를 호출하여 이 메시지를 얻을 수 있습니다.
- 권장되는 예외에 대한 사용자 응답. 애플리케이션은 `JmsExceptionDetail.getUserAction()` 메소드를 호출하여 이 메시지를 얻을 수 있습니다.
- 예외 메시지에서 메시지 삽입을 위한 키. 애플리케이션은 `JmsExceptionDetail.getKeys()` 메소드를 호출하여 모든 키에 대한 반복자를 얻을 수 있습니다.
- 예외 메시지의 메시지 삽입. 예를 들어, 메시지 삽입이 예외를 발생시킨 큐의 이름일 수 있으며 이는 애플리케이션이 해당 이름에 액세스하도록 하는 데 유용할 수 있습니다. 애플리케이션은 `JmsExceptionDetail.getValue()` 메소드를 호출하여 지정된 키에 대응하는 메시지 삽입을 얻을 수 있습니다.

세부사항을 사용할 수 없으면 `JmsExceptionDetail` 인터페이스의 모든 메소드가 널을 리턴합니다.

예를 들어, 애플리케이션이 존재하지 않는 IBM MQ 큐에 대해 `JMSProducer`을(를) 작성하려고 하면 다음 정보와 함께 예외가 발생합니다.

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

던져진 예외는 다음 클래스의 하위 클래스이며, `com.ibm.msg.client.jms.DetailedInvalidDestinationException` `com.ibm.msg.client.jms.JmsExceptionDetail` 인터페이스를 구현합니다.

- **JM 3.0** `jakarta.jms.InvalidDestinationException`
- **JMS 2.0** `javax.jms.InvalidDestinationException`

체인화된 예외

일반적으로 예외는 다른 예외에 의해 발생합니다. 따라서 처리되는 `JMSRuntimeException`마다 애플리케이션은 링크된 예외를 확인해야 합니다.

`JMSRuntimeException`의 원인은 다른 예외일 수 있습니다. 이러한 예외는 원래 기본 문제점으로 다시 되돌리는 체인을 구성합니다. 예외의 원인은 `java.lang.Throwable` 클래스의 체인된 예외 메커니즘을 사용하여 구현되며, 애플리케이션은 `Throwable.getCause()` 메소드를 호출하여 링크된 예외를 얻을 수 있습니다.

예를 들어, 큐 관리자에 연결할 때 애플리케이션이 잘못된 포트 번호를 지정하면 예외에서 다음 체인을 작성합니다.

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
```

```

|
+--->
    java.net.ConnectionException

```

일반적으로, 체인의 각 예외는 코드의 다른 계층에서 발생합니다. 예를 들어, 선행 체인의 예외가 다음 계층에서 발생합니다.

- JMSRuntimeException의 서브클래스 인스턴스인 첫 번째 예외는 IBM MQ classes for JMS의 공통 계층을 통해 처리됩니다.
- com.ibm.mq.MQException의 인스턴스인 다음 예외는 IBM MQ 메시징 제공자를 통해 처리됩니다.
- 다음 두 예외는 모두의 인스턴스이며 com.ibm.mq.jmqi.JmqiException, 메시지 큐잉 인터페이스 Java (JMQUI)에 의해 발생합니다. JMQUI는 IBM MQ classes for JMS에서 큐 관리자와 통신하는 데 사용되는 컴포넌트입니다.
- java.net.ConnectionException의 인스턴스인 최종 예외는 Java 클래스 라이브러리를 통해 처리됩니다.

IBM MQ classes for JMS의 계층화된 아키텍처에 대한 자세한 정보는 [JMS용 IBM MQ 클래스 아키텍처](#)를 참조하십시오.

다음 예제와 같이 모든 적절한 정보를 추출하도록 이 체인을 통해 반복할 애플리케이션을 코드화할 수 있습니다.

JM 3.0

```

import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
    System.err.println("Caught JMSRuntimeException");
    // Check for linked exceptions in JMSRuntimeException
    Throwable t = je;
    while (t != null) {
        // Write out the message that is applicable to all exceptions
        System.err.println("Exception Msg: " + t.getMessage());
        // Write out the exception stack trace
        t.printStackTrace(System.err);

        // Add on specific information depending on the type of exception
        if (t instanceof JMSRuntimeException) {
            JMSRuntimeException je1 = (JMSRuntimeException) t;
            System.err.println("JMS Error code: " + je1.getErrorCode());
            if (t instanceof JmsExceptionDetail){
                JmsExceptionDetail jed = (JmsExceptionDetail)je1;
                System.err.println("JMS Explanation: " + jed.getExplanation());
                System.err.println("JMS Explanation: " + jed.getUserAction());
            }
        } else if (t instanceof MQException) {
            MQException mqe = (MQException) t;
            System.err.println("WMQ Completion code: " + mqe.getCompCode());
            System.err.println("WMQ Reason code: " + mqe.getReason());
        } else if (t instanceof JmqiException){
            JmqiException jmqie = (JmqiException)t;
            System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
            System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
            System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
            System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
            System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
        }
        // Get the next cause
        t = t.getCause();
    }
}

```

JMS 2.0

```

import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSRuntimeException;
.
.
.

```



```

catch (JMSRuntimeException je) {
    System.err.println("Caught JMSRuntimeException");
    // Check for linked exceptions in JMSRuntimeException
    Throwable t = je;
    while (t != null) {
        // Write out the message that is applicable to all exceptions
        System.err.println("Exception Msg: " + t.getMessage());
        // Write out the exception stack trace
        t.printStackTrace(System.err);

        // Add on specific information depending on the type of exception
        if (t instanceof JMSRuntimeException) {
            JMSRuntimeException je1 = (JMSRuntimeException) t;
            System.err.println("JMS Error code: " + je1.getErrorCode());
            if (t instanceof JmsExceptionDetail){
                JmsExceptionDetail jed = (JmsExceptionDetail)je1;
                System.err.println("JMS Explanation: " + jed.getExplanation());
                System.err.println("JMS Explanation: " + jed.getUserAction());
            }
        } else if (t instanceof MQException) {
            MQException mqe = (MQException) t;
            System.err.println("WMQ Completion code: " + mqe.getCompCode());
            System.err.println("WMQ Reason code: " + mqe.getReason());
        } else if (t instanceof JmqiException){
            JmqiException jmqie = (JmqiException)t;
            System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
            System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
            System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
            System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
            System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
        }
        // Get the next cause
        t = t.getCause();
    }
}

```

참고로, 예외의 유형이 다양할 수 있으며 서로 다른 유형의 예외가 다른 정보를 캡슐화할 수 있으므로 애플리케이션은 항상 체인에서 각 예외의 유형을 확인해야 합니다.

문제점에 대한 IBM MQ 특정 정보 확보

`com.ibm.mq.MQException` 및 `com.ibm.mq.jmqi.JmqiException`의 인스턴스가 문제점에 대한 IBM MQ 고유 정보를 캡슐화합니다.

`MQException`에서는 다음 정보를 캡슐화합니다.

- 애플리케이션이 `getCompCode()` 메소드를 호출하여 얻을 수 있는 완료 코드.
- 애플리케이션이 `getReason()` 메소드를 호출하여 얻을 수 있는 이유 코드.

이러한 메소드를 사용하는 방법에 대한 예제는 [체인화된 예외의 샘플 코드](#)를 참조하십시오.

`JmqiException`은(는) 완료 코드 및 이유 코드도 캡슐화합니다. 이 외에도 `JmqiException`에는 AMQ *nnnn* 또는 CSQ *nnnn* 메시지의 정보가 포함되어 있습니다(예외와 연관된 경우). 애플리케이션은 다음 메소드를 호출하여 이 메시지의 다양한 컴포넌트를 확보할 수 있습니다.

- `getWmqMsgExplanation()` 메소드는 AMQ *nnnn* 또는 CSQ *nnnn* 메시지에 대한 설명을 리턴합니다.
- `getWmqMsgSeverity()` 메소드는 AMQ *nnnn* 또는 CSQ *nnnn* 메시지의 심각도를 리턴합니다.
- `getWmqMsgSummary()` 메소드는 AMQ *nnnn* 또는 CSQ *nnnn* 메시지의 요약을 리턴합니다.
- `getWmqMsgUserResponse()` 메소드는 AMQ *nnnn* 또는 CSQ *nnnn* 메시지와 연관된 사용자 응답을 리턴합니다.

ExceptionListeners

JMS Connection 및 JMSContext 오브젝트에는 큐 관리자에 대한 연관된 연결이 있습니다. 애플리케이션이 `ExceptionListener`을(를) JMS Connection 또는 JMSContext에 등록할 수 있습니다. Connection 또는 JMSContext과(와) 연관된 연결을 사용하지 못하게 설정하는 데 문제가 발생하면 IBM MQ classes for JMS에서 `onException()` 메소드를 호출하여 `ExceptionListener`에 예외를 전달합니다. 그런 다음, 애플리케이션이 연결을 다시 설정할 수 있습니다.

비동기로 메시지를 전달하려고 시도하는 동안 문제점이 발생하는 경우 IBM MQ classes for JMS가 예외 리스너에 예외를 전달할 수도 있습니다.

예외 리스너

IBM MQ 8.0.0 Fix Pack 2에서 JMS `MessageListener` 및 JMS `ExceptionListener`을(를) 구성하는 현재 JMS 애플리케이션의 동작을 유지보수하고 IBM MQ classes for JMS이(가) JMS 스펙과 일관되게 하려면 `ConnectionFactory` 특성 `ASYNCEXCEPTION`의 기본값이 `ASYNC_EXCEPTIONS_CONNECTIONBROKEN`으로 변경됩니다. 따라서 끊긴 연결 오류 코드에 해당하는 예외만 애플리케이션의 `ExceptionListener`에 전달됩니다.

[APAR IT14820](#)은 IBM MQ 9.0.0 Fix Pack 1에서 포함되어, IBM MQ classes for JMS를 업데이트해서 다음을 수행합니다.

- 애플리케이션이 사용 중인 메시지 이용자(동기 또는 비동기)에 상관없이 애플리케이션이 등록한 `ExceptionListener`이(가) 연결 중단 예외에 대해 호출됩니다.
- 애플리케이션이 비동기 메시지 이용자를 사용 중이고 애플리케이션이 사용하는 JMS `ConnectionFactory`에 `ASYNC_EXCEPTIONS_ALL` 값으로 설정된 `ASYNC_EXCEPTION` 특성이 있는 경우 메시지 전달 중에 발생하는 비연결 중단 예외(예: `MQRC_GET_INHIBITED`)가 애플리케이션의 `ExceptionListener`(으)로 전달됩니다.

참고: `ExceptionListener`은(는) 두 개의 TCP/IP 연결(하나는 JMS `Connection`에서 사용하고 하나는 JMS `Session`에서 사용)이 끊어진 경우에도 연결 중단 예외에 대해 한 번만 호출됩니다.

다른 유형의 문제점에 대해서는 예외가 현재 JMS API 호출에서 발생합니다. 발생하는 예외 유형은 애플리케이션에서 사용하는 JMS API의 버전에 따라 다릅니다.

- 애플리케이션이 JMS 1.1 스펙에서 제공하는 인터페이스를 사용 중이면 예외는 `JMSException`입니다. 이러한 예외를 처리하는 방법에 대한 자세한 정보는 [227 페이지의 『선택된 예외 처리』](#)의 내용을 참조하십시오.
- 애플리케이션에서 JMS 2.0 인터페이스를 사용 중이면 예외는 `JMSRuntimeException`입니다. 이러한 예외를 처리하는 방법에 대한 자세한 정보는 [230 페이지의 『선택 취소된 예외 처리』](#)의 내용을 참조하십시오.

애플리케이션이 `Connection` 또는 `JMSContext`에 예외 리스너를 등록하지 않은 경우 예외 리스너에 전달되는 모든 예외는 IBM MQ classes for JMS 로그에 기록합니다.

IBM MQ classes for JMS 애플리케이션에서 IBM MQ 기능에 액세스

IBM MQ classes for JMS는 IBM MQ의 다수의 기능을 최대한 활용하는 기능을 제공합니다.



주의: 이러한 기능은 JMS 스펙을 벗어나거나, 특정 경우에는 JMS 스펙을 위반합니다. 이를 사용하는 경우, 애플리케이션이 다른 JMS 제공자와 호환 가능하지 않을 수 있습니다. JMS 스펙을 준수하지 않는 해당 기능에는 주의 표시가 레이블로 지정되어 있습니다.

IBM MQ classes for JMS 애플리케이션에서 메시지 디스크립터 읽기 및 쓰기

사용자는 목적지 및 메시지에 대한 특성을 설정하여 메시지 디스크립터(MQMD)에 액세스하는 기능을 제어합니다.

일부 IBM MQ 애플리케이션에서는 이에 송신된 메시지의 MQMD에 특정 값이 설정되도록 요구합니다. IBM MQ classes for JMS는 JMS 애플리케이션이 MQMD 필드를 설정하도록 허용함으로써 JMS 애플리케이션이 IBM MQ 애플리케이션을 "구동"할 수 있도록 하는 메시지 속성을 제공합니다.

MQMD 특성의 설정을 적용하려면 목적지 오브젝트 특성 `WMQ_MQMD_WRITE_ENABLED`를 `true`로 설정해야 합니다. 그리고 메시지의 특성 설정 메소드를 사용하여(예: `setStringProperty`) MQMD 필드에 값을 지정할 수 있습니다. `StrucId` 및 `Version`을 제외한 모든 MQMD 필드가 노출됩니다. `BackoutCount`를 읽을 수는 있지만 이에 쓸 수는 없습니다.

이 예제의 결과로 `MQMD.UserIdentifier`가 "JoeBloggs"로 설정된 큐 또는 토픽에 메시지를 넣습니다.

```
// Create a ConnectionFactory, connection, session, producer, message
// ...

// Create a destination
// ...

// Enable MQMD write
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_WRITE_ENABLED, true);
```

```
// Optionally, set a message context if applicable for this MD field
dest.setIntProperty(WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT,
    WMQConstants.WMQ_MDCTX_SET_IDENTITY_CONTEXT);

// On the message, set property to provide custom UserId
msg.setStringProperty("JMS_IBM_MQMD_UserIdentifier", "JoeBloggs");

// Send the message
// ...
```

JMS_IBM_MQMD_UserIdentifier를 설정하기 전에 WMQ_MQMD_MESSAGE_CONTEXT를 설정해야 합니다. WMQ_MQMD_MESSAGE_CONTEXT의 사용에 대한 자세한 정보는 [237 페이지의 『JMS 메시지 오브젝트 특성』](#)의 내용을 참조하십시오.

이와 유사하게, 메시지를 수신하기 전에 WMQ_MQMD_READ_ENABLED를 true로 설정한 후에 메시지의 Get 메소드(예: getStringProperty)를 사용하여 MQMD 필드의 콘텐츠를 추출할 수 있습니다. 수신된 특성은 읽기 전용입니다.

이 예제의 결과로 value 필드는 큐 또는 토픽에서 가져온 메시지의 MQMD.ApplIdentityData 필드의 값을 보유하게 됩니다.

```
// Create a ConnectionFactory, connection, session, consumer
// ...

// Create a destination
// ...

// Enable MQMD read
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_READ_ENABLED, true);

// Receive a message
// ...

// Get MQMD field value using a property
String value = rcvMsg.getStringProperty("JMS_IBM_MQMD_ApplIdentityData");
```

JMS 목적지 오브젝트 특성

목적지 오브젝트의 두 특성은 JMS에서 MQMD에 대한 액세스를 제어하며, 세 번째는 메시지 컨텍스트를 제어합니다.

표 36. 특성 이름 및 설명		
특성	축약형	설명
WMQ_MQMD_WRITE_ENABLED	MDW	JMS 애플리케이션이 MQMD 필드의 값을 설정할 수 있는지 여부
WMQ_MQMD_READ_ENABLED	MDR	JMS 애플리케이션이 MQMD 필드의 값을 추출할 수 있는지 여부
WMQ_MQMD_MESSAGE_CONTEXT	MDCTX	JMS 애플리케이션이 설정하는 메시지 컨텍스트의 레벨. 이 특성을 적용하려면 애플리케이션이 적절한 컨텍스트 권한으로 실행되어야 합니다.

표 37. 특성 이름, 값 및 Set 메소드			
특성	관리 도구의 올바른 값(기본값은 굵은체)	프로그램의 올바른 값	Set 메소드
WMQ_MQMD_WRITE_ENABLED	• NO 모든 JMS_IBM_MQMD* 특성이 무시되며, 해당 값이 기본 MQMD 구조로 복사되지 않습니다. • YES JMS_IBM_MQMD* 특성이 처리됩니다. 해당 값이 기본 MQMD 구조로 복사됩니다.	• False • True	setMQMDWriteEnabled
WMQ_MQMD_READ_ENABLED	• NO 메시지를 송신할 때 송신된 메시지의 JMS_IBM_MQMD* 특성이 MQMD에서 업데이트된 필드 값을 반영하도록 업데이트되지 않습니다. 메시지를 수신할 때 어떤 JMS_IBM_MQMD* 특성도 수신된 메시지에서 사용 가능하지 않습니다(송신자가 이 중에서 일부 또는 모두를 설정한 경우에도). • YES 메시지를 송신할 때 송신된 메시지의 모든 JMS_IBM_MQMD* 특성이 업데이트되어 송신자가 명시적으로 설정하지 않은 것을 포함하여 MQMD의 업데이트된 필드 값을 반영합니다. 메시지를 수신할 때 송신자가 명시적으로 설정하지 않은 것을 포함하여 모든 JMS_IBM_MQMD* 특성이 수신된 메시지에서 사용 가능합니다.	• False • True	setMQMDReadEnabled

표 37. 특성 이름, 값 및 Set 메소드 (계속)			
특성	관리 도구의 올바른 값(기본값은 굵은체)	프로그램의 올바른 값	Set 메소드
WMQ_MQMD_MESSAGE_CONTEXT	• 기본값 MQOPEN API 호출 및 MQPMO 구조가 명시적 메시지 컨텍스트 옵션을 지정하지 않음 • SET_IDENTITY_CONTEXT MQOPEN API 호출이 메시지 컨텍스트 옵션 MQOO_SET_IDENTITY_CONTEXT를 지정하며, MQPMO 구조가 MQPMO_SET_IDENTITY_CONTEXT를 지정함 • SET_ALL_CONTEXT MQOPEN API 호출이 메시지 컨텍스트 옵션 MQOO_SET_ALL_CONTEXT를 지정하며, MQPMO 구조가 MQPMO_SET_ALL_CONTEXT를 지정함	• WMQ_MDCTX_DEFAULT • WMQ_MDCTX_SET_IDENTITY_CONTEXT • WMQ_MDCTX_SET_ALL_CONTEXT	setMQMDMessageContext

JMS 메시지 오브젝트 특성

JMS_IBM_MQMD가 접두부로 지정된 메시지 오브젝트 특성을 사용하면 해당되는 MQMD 필드를 설정하거나 읽을 수 있습니다.

메시지 송신

StrucId 및 Version을 제외한 모든 MQMD 필드가 표시됩니다. 이러한 특성은 MQMD 필드만 참조합니다. 여기서 특성은 MQMD 및 MQRFH2 헤더 모두에서 발생하며, MQRFH2의 버전은 설정되거나 추출되지 않습니다.

JMS_IBM_MQMD_BackoutCount를 제외한 이러한 특성을 설정할 수 있습니다.

JMS_IBM_MQMD_BackoutCount에 대해 설정된 값은 무시됩니다.

특성이 최대 길이를 보유하며 너무 긴 값을 제공하는 경우에는 값이 잘립니다.

특정 특성의 경우, 목적지 오브젝트에서 WMQ_MQMD_MESSAGE_CONTEXT 특성을 설정해야 할 수도 있습니다. 이 특성이 적용되려면 애플리케이션이 적합한 컨텍스트 권한으로 실행 중이어야 합니다.

WMQ_MQMD_MESSAGE_CONTEXT를 적합한 값으로 설정하지 않으면 특성 값이 무시됩니다.

WMQ_MQMD_MESSAGE_CONTEXT를 적합한 값으로 설정하지만 큐 관리자에 대해 충분한 컨텍스트 권한이 없는 경우에는 JMSException이 발생합니다. WMQ_MQMD_MESSAGE_CONTEXT의 특정 값을 요구하는 특성은 다음과 같습니다.

다음 특성은 WMQ_MQMD_MESSAGE_CONTEXT가 WMQ_MDCTX_SET_IDENTITY_CONTEXT 또는 WMQ_MDCTX_SET_ALL_CONTEXT로 설정되도록 요구합니다.

- JMS_IBM_MQMD_UserIdentifier
- JMS_IBM_MQMD_AccountingToken
- JMS_IBM_MQMD_ApplIdentityData

다음 특성은 WMQ_MQMD_MESSAGE_CONTEXT가 WMQ_MDCTX_SET_ALL_CONTEXT로 설정되도록 요구합니다.

- JMS_IBM_MQMD_PutApplType
- JMS_IBM_MQMD_PutApplName
- JMS_IBM_MQMD_PutDate
- JMS_IBM_MQMD_PutTime

메시지 수신

생성 애플리케이션이 설정한 실제 특성과는 무관하게 WMQ_MQMD_READ_ENABLED 특성이 true로 설정된 경우 이 모든 특성은 수신된 메시지에서 사용 가능합니다. JMS 스펙에 따라 우선 모든 특성이 선택 취소되지 않는 한 애플리케이션은 수신된 메시지의 특성을 수정할 수 없습니다. 수신된 메시지는 특성의 수정 없이 전달될 수 있습니다.



주의: WMQ_MQMD_READ_ENABLED 특성이 true로 설정된 목적지에서 애플리케이션이 메시지를 수신하고 WMQ_MQMD_WRITE_ENABLED가 true로 설정된 목적지에 이를 전달하는 경우, 이 결과에 따라 수신된 메시지의 모든 MQMD 필드 값은 전달된 메시지에 복사됩니다.

특성 표

다음 표에는 MQMD 필드를 표시하는 메시지 오브젝트의 특성이 나열되어 있습니다. 필드의 전체 설명과 이에 사용되는 값은 링크를 참조하십시오.

표 38. 특성 이름, 설명 및 유형			
특성	설명	Java 유형	전체 설명에 대한 링크
JMS_IBM_MQMD_Report	보고 메시지의 옵션	정수	보고서
JMS_IBM_MQMD_MsgType	메시지 유형	정수	MsgType
JMS_IBM_MQMD_Expiry	메시지 수명	정수	만기
JMS_IBM_MQMD_Feedback	피드백 또는 이유 코드	정수	피드백
JMS_IBM_MQMD_Encoding	메시지 데이터의 숫자 인코딩	정수	인코딩
JMS_IBM_MQMD_CodedCharSetId	메시지 데이터의 문자 세트 ID	정수	CodedCharSetId
JMS_IBM_MQMD_Format	메시지 데이터의 형식 이름	문자열	Format
JMS_IBM_MQMD_Priority ¹	메시지 우선순위	정수	우선순위
JMS_IBM_MQMD_Persistence	메시지 지속성	정수	persistence
JMS_IBM_MQMD_MsgId ²	메시지 ID	오브젝트 (byte[]) ⁴	MsgId
JMS_IBM_MQMD_CorrelId ³	상관 ID	오브젝트 (byte[]) ⁴	CorrelId
JMS_IBM_MQMD_BackoutCount	백아웃 카운터	정수	BackoutCount
JMS_IBM_MQMD_ReplyToQ	응답 큐의 이름	문자열	ReplyToQ
JMS_IBM_MQMD_ReplyToQMGr	응답 큐 관리자의 이름	문자열	ReplyToQMGr
JMS_IBM_MQMD_UserIdentifier	사용자 ID	문자열	UserIdentifier
JMS_IBM_MQMD_AccountingToken	계정 토큰	오브젝트 (byte[]) ⁴	AccountingToken
JMS_IBM_MQMD_ApplIdentityData	ID 관련 애플리케이션 데이터	문자열	ApplIdentityData
JMS_IBM_MQMD_PutApplType	메시지를 넣는 애플리케이션의 유형	정수	PutApplType

표 38. 특성 이름, 설명 및 유형 (계속)			
특성	설명	Java 유형	전체 설명에 대한 링크
JMS_IBM_MQMD_PutApplName	메시지를 넣는 애플리케이션의 이름	문자열	PutApplName
JMS_IBM_MQMD_PutDate	메시지를 넣은 날짜	문자열	PutDate
JMS_IBM_MQMD_PutTime	메시지를 넣은 시간	문자열	PutTime
JMS_IBM_MQMD_ApplOriginData	원본 관련 애플리케이션 데이터	문자열	ApplOriginData
JMS_IBM_MQMD_GroupId	그룹 ID	오브젝트 (byte[]) ⁴	GroupId
JMS_IBM_MQMD_MsgSeqNumber	그룹 내 논리 메시지의 순서 번호	정수	MsgSeqNumber
JMS_IBM_MQMD_Offset	논리 메시지의 시작에서 실제 메시지의 데이터의 오프셋	정수	offset
JMS_IBM_MQMD_MsgFlags	메시지 플래그	정수	MsgFlags
JMS_IBM_MQMD_OriginalLength	원래 메시지의 길이	정수	OriginalLength

-  **주의:** 0-9 범위에 없는 JMS_IBM_MQMD_Priority로 값이 지정되는 경우, 이는 JMS 스펙을 위반합니다.
-  **주의:** JMS 스펙은 메시지 ID가 JMS 제공자에 의해 설정되어야 하며 이는 고유하거나 넓이어야 함을 기술합니다. 값을 JMS_IBM_MQMD_MsgId로 지정되는 경우, 이 값은 JMSMessageID에 복사됩니다. 따라서 이는 JMS 제공자에 의해 설정되지 않으며 고유하지 않을 수 있습니다. 이는 JMS 스펙을 위반합니다.
-  **주의:** 'ID:' 문자열로 시작되는 JMS_IBM_MQMD_CorrelId로 값을 지정하는 경우, 이는 JMS 스펙을 위반합니다.
-  **주의:** 메시지에서 바이트 배열 특성의 사용은 JMS 스펙을 위반합니다.

'IBM MQ classes for JMS'를 사용하여 애플리케이션에서 'IBM MQ' 메시지 데이터에 액세스하기
 'IBM MQ classes for JMS'를 사용하여 애플리케이션 내에서 전체 'IBM MQ' 메시지 데이터에 액세스할 수 있습니다. 모든 데이터에 액세스하려면 메시지가 JMSBytesMessage여야 합니다. JMSBytesMessage의 본문에는 임의의 MQRFH2 헤더 및 기타 IBM MQ 헤더, 그리고 다음의 메시지 데이터가 포함됩니다.

JMSBytesMessage의 모든 메시지 본문 데이터를 수신하려면 목적지의 WMQ_MESSAGE_BODY 특성을 WMQ_MESSAGE_BODY_MQ로 설정하십시오.

WMQ_MESSAGE_BODY가 WMQ_MESSAGE_BODY_JMS 또는 WMQ_MESSAGE_BODY_UNSPECIFIED로 설정하는 경우, 메시지 본문은 JMS MQRFH2 헤더 없이 리턴되며 JMSBytesMessage의 특성은 RFH2에 설정된 특성을 반영합니다.

일부 애플리케이션은 이 주제에서 설명된 기능을 사용할 수 없습니다. 애플리케이션이 IBM MQ V6 큐 관리자에 연결되어 있거나 해당 PROVIDERVERSION이 6으로 설정된 경우에는 기능을 사용할 수 없습니다.

메시지 송신

메시지를 송신할 때 목적지 특성, WMQ_MESSAGE_BODY는 WMQ_TARGET_CLIENT에 우선합니다.

WMQ_MESSAGE_BODY가 'WMQ_MESSAGE_BODY_JMS'로 설정된 경우 'IBM MQ classes for JMS' JMSMessage' 속성 및 헤더 필드 설정에 따라 'MQRFH2' 헤더를 자동으로 생성합니다.

WMQ_MESSAGE_BODY가 WMQ_MESSAGE_BODY_MQ로 설정된 경우, 추가 헤더가 메시지 본문에 추가됩니다.

If WMQ_메시지_본문 is set to WMQ_MESSAGE_BODY_UNSPECIFIED, IBM MQ classes for JMS sends an MQRFH2 header, unless Wmq_target_client is set to WMQ_TARGET_DEST_MQ. 수신 시에 WMQ_TARGET_CLIENT를 WMQ_TARGET_DEST_MQ로 설정하면 MQRFH2가 메시지 본문에서 제거됩니다.

참고: JMSBytesMessage 및 JMSTextMessage는 MQRFH2를 요구하지 않는 반면, JMSStreamMessage, JMSMapMessage 및 JMSObjectMessage는 이를 요구합니다.

WMQ_MESSAGE_BODY_UNSPECIFIED는 WMQ_MESSAGE_BODY의 기본 설정이며, WMQ_TARGET_DEST_JMS는 WMQ_TARGET_CLIENT의 기본 설정입니다.

'JMSBytesMessage' 메시지를 보내면 'IBM MQ' 메시지가 작성될 때 'JMS' 메시지 본문에 대한 기본 설정을 재정의할 수 있습니다. 다음 특성을 사용하십시오.

- JMS_IBM_Format 또는 JMS_IBM_MQMD_Format: 이 속성은 'JMS' 메시지 본문을 시작하는 'IBM MQ' 헤더 또는 애플리케이션 페이로드의 형식을 지정합니다(앞의 WebSphere MQ 헤더가 없는 경우).
- JMS_IBM_Character_Set 또는 JMS_IBM_MQMD_CodedCharSetId: 이 속성은 'IBM MQ' 헤더 또는 'JMS' 메시지 본문을 시작하는 애플리케이션 페이로드의 'CCSID'를 지정합니다(앞의 WebSphere MQ 헤더가 없는 경우).
- JMS_IBM_Encoding 또는 'JMS_IBM_MQMD_인코딩': 이 속성은 앞의 'WebSphere MQ' 헤더가 없는 경우 'JMS' 메시지 본문을 시작하는 'IBM MQ' 헤더 또는 애플리케이션 페이로드의 인코딩을 지정합니다.

특성의 두 유형이 모두 지정된 경우, 목적지 특성 WMQ_MQMD_WRITE_ENABLED가 true로 설정되어 있는 한 JMS_IBM_MQMD_* 특성이 해당되는 JMS_IBM_* 특성을 대체합니다.

JMS_IBM_MQMD_* 및 JMS_IBM_*를 사용한 메시지 특성 설정 간의 실제 차이는 큼니다.

1. JMS_IBM_MQMD_* 특성은 IBM MQ JMS 제공자에 특정합니다.
2. JMS_IBM_MQMD_* 특성은 MQMD에서만 설정됩니다. JMS_IBM_* 속성은 메시지에 'MQRFH2' 'JMS' 헤더가 없는 경우에만 'MQMD'에 설정됩니다. 그렇지 않으면, 이는 JMS RFH2 헤더에 설정됩니다.
3. JMS_IBM_MQMD_* 특성은 JMSMessage에 쓰여진 숫자 및 텍스트의 인코딩에 영향을 주지 않습니다.

수신 애플리케이션은 메시지 본문에서 숫자 및 텍스트의 인코딩 및 문자 세트에 해당되는 MQMD.Encoding 및 MQMD.CodedCharSetId의 값을 가정할 수 있습니다. JMS_IBM_MQMD_* 특성이 사용되는 경우, 이의 수행은 송신 애플리케이션의 책임입니다. 메시지 본문에서 숫자 및 텍스트의 인코딩 및 문자 세트는 JMS_IBM_* 특성에 의해 설정됩니다.

240 페이지의 그림 39의 잘못 코드화된 스니펫은 MQMD.CodedCharSetId가 37이며 문자 세트 1208로 인코딩된 메시지를 송신합니다.

a. 잘못 인코딩된 메시지 송신

```
TextMessage tmo = session.createTextMessage();
((MQDestination) destination).setMessageBodyStyle
    (WMQConstants.WMQ_MESSAGE_BODY_MQ);
((MQDestination)destination).setMQMDWriteEnabled(true);
tmo.setIntProperty(WMQConstants.JMS_IBM_MQMD_CODEDCHARSETID, 37);
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 1208);
tmo.setText("String one");
producer.send(tmo);
```

b. MQMD.CodedCharSetId의 값으로 설정된 JMS_IBM_CHARACTER_SET의 값에 의존하여 메시지 수신:

```
TextMessage tmi = (TextMessage) cons.receive();
System.out.println("Message is \"" + tmi.getText() + "\"");
```

c. 결과 출력:

```
Message is "éÊË'>...??>?"
```

그림 39. 불일치하게 코드화된 MQMD 및 메시지 데이터

241 페이지의 그림 40에서 코드 스니펫 중 하나의 결과로 메시지는 자동 생성된 MQRFH2 헤더의 추가 없이 애플리케이션 페이로드가 포함되어 해당 본문에서 큐 또는 토픽에 놓여집니다.

1. WMQ_MESSAGE_BODY_MQ 설정:

```
((MQDestination) destination).setMessageBodyStyle
(WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

2. WMQ_TARGET_DEST_MQ 설정:

```
((MQDestination) destination).setMessageBodyStyle
(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);
((MQDestination) destination).
setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
```

그림 40. MQ 메시지 본문의 메시지 송신.

메시지 수신

WMQ_MESSAGE_BODY가 WMQ_MESSAGE_BODY_JMS로 설정된 경우, 인바운드 JMS 메시지 유형 및 본문은 수신된 WebSphere MQ 메시지의 콘텐츠에 의해 판별됩니다. 메시지 유형 및 본문은 MQRFH2 헤더 또는 MQMD(MQRFH2가 없는 경우)의 필드에 의해 판별됩니다.

WMQ_MESSAGE_BODY가 WMQ_MESSAGE_BODY_MQ로 설정된 경우, 인바운드 JMS 메시지 유형은 JMSBytesMessage입니다. JMS 메시지 본문은 기본 MQGET API 호출에 의해 리턴된 메시지 데이터입니다. 메시지 본문의 길이는 MQGET 호출에 의해 리턴된 길이입니다. 메시지 본문에서 데이터의 문자 세트 및 인코딩은 MQMD의 CodedCharSetId 및 Encoding 필드에 의해 판별됩니다. 메시지 본문에 있는 데이터의 형식은 MQMD의 Format 필드에 의해 판별됩니다.

WMQ_MESSAGE_BODY가 WMQ_MESSAGE_BODY_UNSPECIFIED(기본값)로 설정된 경우, IBM MQ classes for JMS는 이를 WMQ_MESSAGE_BODY_JMS로 설정합니다.

JMSBytesMessage를 수신하는 경우에는 다음 특성을 참조하여 이를 디코딩할 수 있습니다.

- JMS_IBM_Format 또는 JMS_IBM_MQMD_Format: 이 속성은 'JMS' 메시지 본문을 시작하는 'IBM MQ 헤더 또는 애플리케이션 페이로드의 형식을 지정합니다(앞의 WebSphere MQ 헤더가 없는 경우).
- JMS_IBM_Character_Set 또는 JMS_IBM_MQMD_CodedCharSetId: 이 속성은 'IBM MQ 헤더 또는 JMS 메시지 본문을 시작하는 애플리케이션 페이로드의 'CCSID'를 지정합니다(앞의 WebSphere MQ 헤더가 없는 경우).
- JMS_IBM_Encoding 또는 'JMS_IBM_MQMD_인코딩': 이 속성은 앞의 'WebSphere MQ 헤더가 없는 경우 JMS' 메시지 본문을 시작하는 'IBM MQ' 헤더 또는 애플리케이션 페이로드의 인코딩을 지정합니다.

다음 코드 스니펫의 결과는 JMSBytesMessage인 수신 메시지입니다. 수신된 MQMD의 형식 필드와 수신 메시지의 콘텐츠와는 무관하게, 메시지는 JMSBytesMessage입니다.

```
((MQDestination)destination).setMessageBodyStyle
(WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

목적지 특성 WMQ_MESSAGE_BODY

WMQ_MESSAGE_BODY는 JMS 애플리케이션이 IBM MQ 메시지의 MQRFH2 를 메시지 페이로드의 일부로 (즉, JMS 메시지 본문의 일부로) 처리하는지 여부를 판별합니다.

표 39. 특성 이름 및 설명		
특성	축약형	설명
WMQ_MESSAGE_BODY	MBODY	JMS 애플리케이션이 IBM MQ 메시지의 MQRFH2 를 메시지 페이로드의 일부로 (즉, JMS 메시지 본문의 일부로) 처리하는지 여부입니다.

표 40. 특성 이름, 값 및 Set 메소드			
특성	관리 도구의 올바른 값(기본값은 굵은체)	프로그램의 올바른 값	Set 메소드
WMQ_MESSAGE_BODY	• UNSPECIFIED 송신 시에 IBM MQ classes for JMS가 WMQ_TARGET_CLIENT의 값에 따라 MQRFH2 헤더를 생성 및 포함하거나 생성 및 포함하지 않습니다. 수신 시에 JMS 값으로 작동합니다. • JMS 송신 시에 IBM MQ classes for JMS가 MQRFH2 헤더를 자동 생성하며 이를 IBM MQ 메시지에 포함합니다. 수신 시에 IBM MQ classes for JMS가 MQRFH2(존재하는 경우)의 값에 따라 JMS 메시지 특성을 설정합니다. 이는 JMS 메시지 본문의 일부로서 MQRFH2를 제시하지 않습니다. • MQ 송신 시에 IBM MQ classes for JMS가 MQRFH2를 생성하지 않습니다. 수신 시에 IBM MQ classes for JMS가 MQRFH2를 JMS 메시지 본문의 일부로서 제시합니다.	• WMQ_MESSAGE_BODY_UNSPECIFIED • WMQ_MESSAGE_BODY_JMS • WMQ_MESSAGE_BODY_MQ	setMessageBodyStyle

JMS 지속 메시지

IBM MQ classes for JMS 애플리케이션은 **NonPersistentMessageClass** 큐 속성을 사용하여 JMS 지속 메시지에 대한 보다 나은 성능을 제공할 수 있으며, 이 경우 신뢰성이 약간 저하됩니다.

IBM MQ 큐에는 **NonPersistentMessageClass**라고 하는 속성이 있습니다. 이 속성의 값은 큐 관리자가 다시 시작될 때 큐에서 비지속 메시지가 제거되는지 여부를 판별합니다.

다음 매개변수 중 하나로 IBM MQ 스크립트(MQSC) 명령, DEFINE QLOCAL을 사용하여 로컬 큐의 속성을 설정할 수 있습니다.

NPMCLASS(NORMAL)

큐 관리자가 다시 시작될 때 큐에서 비지속 메시지가 제거됩니다. 이는 기본값입니다.

NPMCLASS(HIGH)

일시정지 또는 즉시 종료 이후 큐 관리자가 다시 시작할 때 큐에서 비지속 메시지가 제거되지 않습니다. 그러나 강제 종료 또는 장애 이후에는 비지속 메시지가 제거될 수 있습니다.

이 주제에서는 IBM MQ classes for JMS 애플리케이션이 이 큐 속성을 사용하여 JMS 지속 메시지에 대한 보다 나은 성능을 제공할 수 있는 방법을 설명합니다.

큐 또는 토픽 오브젝트의 PERSISTENCE 특성은 HIGH 값을 보유할 수 있습니다. 사용자가 IBM MQ JMS 관리 도구를 사용하여 이 값을 설정하거나, 애플리케이션이 WMQConstants.WMQ_PER_NPHIGH 값을 매개변수로서 전달하는 Destination.setPersistence() 메소드를 호출할 수 있습니다.

애플리케이션이 JMS 지속 메시지 또는 JMS 비지속 메시지를 PERSISTENCE 특성의 값이 HIGH인 목적지에 송신하며 기본 IBM MQ 큐가 NPMCLASS(HIGH)로 설정된 경우에는 해당 메시지를 IBM MQ 비지속 메시지로써 큐에 넣습니다. 목적지의 PERSISTENCE 특성이 HIGH 값을 보유하지 않거나 기본 큐가 NPMCLASS(NORMAL)로 설정된 경우에는 JMS 지속 메시지를 IBM MQ 지속 메시지로써 큐에 넣으며 JMS 비지속 메시지를 IBM MQ 비지속 메시지로써 큐에 넣습니다.

JMS 지속 메시지를 IBM MQ 비지속 메시지로써 큐에 넣으며 큐 관리자의 일시정지 또는 즉시 종료 이후 메시지가 제거되지 않도록 보장하려는 경우, 메시지가 라우팅될 수 있는 모든 큐는 NPMCLASS(HIGH)로 설정되어야 합니다. 발행/구독 도메인에서 이러한 큐에는 구독자 큐가 포함되어 있습니다. 이 구성을 적용하는 데 도움이 되도록, 애플리케이션이 PERSISTENCE 속성의 값이 HIGH이고 기본 IBM MQ 큐가 NPMCLASS(NORMAL)로 설정된 대상에 대한 메시지 소비자를 만들려고 하면 IBM MQ classes for JMS 에서 InvalidDestinationException 을 던집니다.

목적지의 PERSISTENCE 특성을 HIGH로 설정해도 메시지를 해당 목적지로부터 수신하는 방법에는 영향을 주지 않습니다. JMS 지속 메시지로써 송신된 메시지는 JMS 지속 메시지로써 수신되며, JMS 비지속 메시지로써 송신된 메시지는 JMS 비지속 메시지로써 수신됩니다.

애플리케이션이 PERSISTENCE 특성의 값이 HIGH인 목적지에 첫 번째 메시지를 송신하거나 애플리케이션이 PERSISTENCE 특성의 값이 HIGH인 목적지에 대해 첫 번째 메시지 이용자를 작성하는 경우, IBM MQ classes for JMS는 MQINQ 호출을 실행하여 NPMCLASS(HIGH)가 기본 IBM MQ 큐에서 설정되었는지 여부를 판별합니다. 따라서 애플리케이션에는 큐에 대해 조회할 권한이 있습니다. 또한 IBM MQ classes for JMS는 목적지가 삭제될 때까지 MQINQ 호출의 결과를 보존하며, 추가로 MQINQ 호출을 실행하지 않습니다. 따라서 애플리케이션이 아직 목적지를 사용 중인 동안 기본 큐에서 NPMCLASS 설정을 변경하는 경우, IBM MQ classes for JMS는 새 설정을 알지 못합니다.

JMS 지속 메시지를 IBM MQ 비지속 메시지로써 IBM MQ 큐에 넣도록 허용함으로써, 사용자는 일부 신뢰성을 희생하면서 성능을 높일 수 있습니다. JMS 지속 메시지의 최대 신뢰성이 필요한 경우에는 PERSISTENCE 특성의 값이 HIGH인 목적지로 메시지를 송신하지 마십시오.

JMS 계층은 SYSTEM.DEFAULT.MODEL.QUEUE 대신 SYSTEM.JMS.TEMPQ.MODEL을 사용할 수 있습니다. SYSTEM.DEFAULT.MODEL.QUEUE가 지속 메시지를 승인할 수 없으므로 SYSTEM.JMS.TEMPQ.MODEL은 지속 메시지를 승인하는 영구적 동적 큐를 작성합니다. 따라서 임시 큐를 사용하여 지속 메시지를 승인하려면 SYSTEM.JMS.TEMPQ.MODEL을 사용하거나 모델 큐를 직접 선택한 대체 큐로 변경해야 합니다.

IBM MQ classes for JMS에서 TLS 사용

IBM MQ classes for JMS 애플리케이션은 TLS(Transport Layer Security) 암호화를 사용할 수 있습니다. 이를 위해 그들은 (JSSE) Java Secure Socket Extension 공급자가 필요합니다.

TRANSPORT(CLIENT)를 사용한 IBM MQ classes for JMS 연결은 TLS 암호화를 지원합니다. TLS는 통신 암호화, 인증 및 메시지 무결성을 제공합니다. 이는 일반적으로 인트라넷 내의 또는 인터넷의 두 피어 간의 보안 통신에 사용됩니다.

IBM MQ classes for JMS TLS 암호화를 처리하기 위해 JSSE 사용되며, 따라서 제공자가 JSSE 필요합니다. Java SE v1.4Java Virtual Machine (JVM)에는 내장된 제공자가 JSSE 있습니다. 인증서를 관리하고 저장하는 방법의 세부사항은 제공자마다 다를 수 있습니다. 이에 대한 자세한 내용은 공급자의 JSSE 설명서를 참조하십시오.

이 섹션은 공급자가 JSSE 올바르게 설치 및 구성되었으며, 적절한 인증서가 설치되어 공급자가 JSSE 사용할 수 있도록 설정되었음을 전제로 합니다. 이제 JMSAdmin 도구를 사용하여 여러 관리 속성을 설정할 수 있습니다.

IBM MQ classes for JMS 애플리케이션이 클라이언트 채널 정의 테이블(CCDT)을 사용하여 큐 관리자에 연결하는 경우에는 262 페이지의 『IBM MQ classes for JMS에서 클라이언트 채널 정의 테이블 사용』의 내용을 참조하십시오.

관련 태스크

관리 도구를 사용하여 JMS 및 Jakarta Messaging 객체 구성

JMS 오브젝트의 TLS 특성

Transport Layer Security(TLS) 암호화는 속성을 **SSLCIPHERSUITE** 사용하여 활성화합니다. 그런 다음 몇 가지 다른 특성을 사용하여 TLS 암호화의 특성을 변경할 수 있습니다.

TLS 를 지정할 때 **TRANSPORT(CLIENT)**, 속성을 **SSLCIPHERSUITE** 사용하여 암호화된 통신을 활성화할 수 있습니다. 이 속성을 (JSSE) Java Secure Socket Extension 공급자가 제공하는 유효한 CipherSuite 로 설정하십시오. 이는 속성으로 **CHANNEL** 명명된 SVRCONN 채널에 지정된 CipherSpec 과 반드시 일치해야 합니다.

그러나 CipherSpec(SVRCONN 채널에 지정됨)과 CipherSuites(ConnectionFactory 오브젝트에 지정됨)는 다른 이름 지정 체계를 사용하여 동일한 TLS 암호화 알고리즘을 나타냅니다. CipherSpec 속성에 **SSLCIPHERSUITE** 인식된 이름이 지정된 경우, JMSAdmin 또는 JMSAdmin30 도구는 경고를 표시하고 CipherSpec 를 해당 CipherSuite 로 매핑합니다. 지원되는 CipherSuites 및 CipherSpecs, 에 대한 자세한 내용은 [TLS, CipherSpecs 및 CipherSuites](#) 을 IBM MQ classes for JMS 참조하십시오.

CipherSuite 를 사용하기 위해 제공자 IBMJavaJSSEFIPS (IBMJSSEFIPS)가 지원하는 연결이 필요한 경우, 연결 팩토리의 속성을 **SSLFIPSREQUIRED** YES 로 설정하십시오. 이 속성의 기본값은 NO 입니다. 이는 연결이 지원되는 모든 다중 사용자 프로토콜(CipherSuite)을 사용할 수 있음을 의미합니다. 속성이 설정되지 않은 **SSLCIPHERSUITE** 경우 해당 속성은 무시됩니다.

해당 **SSLPEERNAME** 속성은 채널 정의에서 설정할 수 있는 **SSLPEER** 매개변수의 형식과 일치합니다. 쉼표 또는 세미콜론으로 구분된 속성 이름-값 쌍의 목록입니다. 예를 들면, 다음과 같습니다.

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPPHERE)
```

이름과 값의 집합이 구별된 이름을 구성한다. 구별명(Distinguished Names)과 그 사용법에 대한 자세한 내용은 구별명([Distinguished IBM MQ Names](#))을 참조하십시오.

제공된 예에서는 연결 시 서버에 표시된 식별 인증서를 검사합니다. 연결이 성공하려면 인증서의 일반 이름(Common Name)이 QMGR.으로 시작해야 합니다 그리고 최소한 두 개의 조직 단위 이름이 있어야 하며, 첫 번째는 IBM 이고 두 번째는 WEBSPPHERE입니다. 검사는 대소문자를 구분하지 않습니다.

가 설정되지 **SSLPEERNAME** 않은 경우, 그러한 검사는 수행되지 않습니다. **SSLPEERNAME** 설정되지 **SSLCIPHERSUITE** 않은 경우 무시됩니다.

이 **SSLCRL** 속성은 하나 이상의 CRL(인증서 해지 목록) 서버를 지정합니다. 이 특성을 사용하려면 Java 2 v1.4에 JVM이 필요합니다. 이것은 쉼표로 구분된 양식 항목 목록입니다.

```
ldap://<hostname>:<port>]
```

선택적으로 뒤에 단일 /가 옵니다. LDAP 의 기본 포트인 389가 생략된 <port> 경우로 간주됩니다. 연결 시 서버에서 제공하는 TLS 인증서는 지정된 CRL 서버를 기준으로 검사됩니다. CRL 보안에 대한 자세한 내용은 ['해지된 인증서 작업'](#)을 참조하십시오.

가 설정되지 **SSLCRL** 않은 경우, 그러한 검사는 수행되지 않습니다. **SSLCRL** 설정되지 **SSLCIPHERSUITE** 않은 경우 무시됩니다.

해당 **SSLRESETCOUNT** 속성은 암호화에 사용되는 비밀 키가 재협상되기 전에 연결을 통해 송수신된 총 바이트 수를 나타냅니다. 송신된 바이트 수는 암호화 전의 바이트 수이며, 수신된 바이트 수는 복호화 후의 바이트 수입니다. 바이트 수에는 IBM MQ classes for JMS에 의해 송신되거나 수신된 제어 정보도 포함됩니다.

예를 들어, 4MB의 데이터가 전송된 후 비밀 키가 재협상되는 TLS 가 활성화된 MQI 채널을 통해 연결을 생성하는데 사용할 수 있는 `ConnectionFactory` 객체를 구성하려면, JMSAdmin 또는 JMSAdmin30: 에 다음 명령을 실행하십시오

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

값이 **SSLRESETCOUNT** 0인 경우(기본값), 비밀 키는 재협상되지 않습니다. 속성이 설정되지 않은 **SSLCIPHERSUITE** 경우 해당 **SSLRESETCOUNT** 속성은 무시됩니다.

TLS 를 /에서 IBM MQ classes for JMS활성화합니다.Jakarta Messaging ConnectionFactory 개체에 대해 TLS 암호화를 활성화하도록 **SSLCIPHERSUITE** 설정합니다.

ConnectionFactory 객체에서 TLS 암호화를 활성화하려면 해당 **SSLCIPHERSUITE** 속성을 (JSSE) Java Secure Socket Extension 공급자가 지원하는 CipherSuite 로 설정하십시오. 이는 대상 채널에서 설정된 CipherSpec과 일치해야 합니다. 그러나 CipherSuite는 CipherSpec과는 구분되므로, 다른 이름을 갖습니다. 253 페이지의 『IBM MQ classes for JMS의 TLS CipherSpecs 및 CipherSuites』 CipherSpecs 가 IBM MQ 지원하는 객체 유형을 해당 객체 유형의 동등한 JSSECipherSuites 로 매핑하는 테이블을 포함합니다. CipherSpecs 및 CipherSuites 사용에 대한 자세한 내용은 [CipherSpecs, CipherSuites 및 CipherSpecsIBM MQ 활성화 항목을](#) 참조하십시오.

다음과 같은 방법으로 **SSLCIPHERSUITE** 설정할 수 있습니다:

JMSAdmin 또는 JMSAdmin30 도구를 사용하여

예를 들어, TLS_RSA_WITH_AES_128_CBC_SHA CipherSpec 을 사용하여 TLS 활성화된 MQI 채널을 통해 연결을 생성하는 데 사용할 수 있는 ConnectionFactory 객체를 설정하려면 JMSAdmin 또는 JMSAdmin30:

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_WITH_AES_128_CBC_SHA)
```

어플리케이션에서

SSLCIPHERSUITE 응용 프로그램에서 MQConnectionFactory 객체의 `setSSLCipherSuite()` 메서드를 사용하여 설정할 수 있습니다.

편의를 위해, 속성에 **SSLCIPHERSUITE** `CipherSpec`가 지정된 경우 JMSAdmin 또는 `JMSAdmin30`는 `CipherSpec`를 적절한 `CipherSuite`에 매핑하려고 시도하며 경고를 발생시킵니다. 애플리케이션이 특성을 지정한 경우에는 이러한 맵핑 시도가 이루어지지 않습니다.

ConnectionFactory 객체의 특정 속성을 설정하여 클라이언트 연결 채널 정의를 생성하는 대신, 애플리케이션은 클라이언트 채널 정의 테이블(CCDT)에 저장된 클라이언트 연결 채널 정의를 사용할 수 있습니다. 자세한 정보는 262 페이지의 『IBM MQ classes for JMS에서 클라이언트 채널 정의 테이블 사용』의 내용을 참조하십시오.

관련 개념

JMS 오브젝트의 TLS 특성

Transport Layer Security(TLS) 암호화는 속성을 **SSLCIPHERSUITE** 사용하여 활성화합니다. 그런 다음 몇 가지 다른 특성을 사용하여 TLS 암호화의 특성을 변경할 수 있습니다.

관련 태스크

[JMSAdmin 및 JMS30Admin 을 사용한 관리 명령어 실행](#)

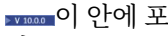
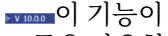
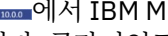
관련 참조

SSLCIPHERSUITE

/에서 연결에 인증된 *IBM MQ classes for JMSCipherSpecsFIPS* 만 사용되는지 여부를 지정합니다.*Jakarta Messaging*

CipherSuite 을 사용하기 위해 (JSSE) FIPSIBMJava Secure Socket Extension 제공자(IBMJSSEFIPS)가 지원하는 연결이 필요한 경우, 연결 팩토리의 속성을 **SSLFIPSREQUIRED** YES 로 설정하십시오.

참고:

-  AIX, Linux, 및 Windows에서는 IBM MQ 암호 IBM Crypto for C (ICC) 모듈을 GSKit 9 통해 규정 준수를 제공합니다 FIPS 140-3 . 해당 FIPS 140-3 모듈과 관련된 NIST 인증서는 <https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4755> 에서 확인할 수 있습니다.
-  Linux s390xFIPS 현재.에 대한 지원이 제공되지 않습니다. IBM MQ이 플랫폼에서 지원이 필요한 FIPS 고객은 이전 버전을 계속 사용해야 합니다. 해당 IBM Crypto for C (ICC) 암호화 모듈이 이 플랫폼에서 인증을 받으면, 향후 FixPack 에서 FIPS 140-3 지원이 활성화될 예정입니다.
-  이 안에 포함된 암호화 FIPS 140-3 모듈은 IBM Semeru Runtime 2024년 8월 NIST의 승인을 받았습니다. IBM Semeru Runtime 11+IBM MQ 10.0.0 TLS 를 FIPS 140-3 사용하여 및 IBM MQ classes for Java 의 IBM MQ classes for JMS 및 내에서 Java 8 클라이언트 연결 처리를 지원합니다. <https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4755> 해당 FIPS 140-3 모듈과 관련된 NIST 인증서는 에서 확인할 수 있습니다.
- 컨테이너 환경의 IBM MQ 경우, IBM MQ Operator3.2.0 및 9.4.0.0 이후 버전의 큐 매니저 컨테이너 이미지는 UBI 9를 기반으로 합니다. FIPS 140-3 컨테이너 내에서의 IBM MQ 규정 준수 여부는 현재 검토 중입니다.
 이 기능이 FIPS 활성화되어 있으면, 컨테이너 IBM MQ 제어 프로세스에서 인증된 OpenSSLFIPS 140-3 모듈을 사용합니다. NIST 인증에 대한 자세한 내용은 다음 링크에서 확인하실 수 있습니다: <https://access.redhat.com/compliance/fips>. IBM MQ 컨테이너 이미지에서 실행되는 큐 관리자는 기본 이미지 플랫폼 버전의 인증 수준과 FIPSIBM MQ 동일합니다.
-  에서 IBM MQ 10.0.0, 연결 팩토리 속성 은 에서 **SSLFIPSREQUIRED** 지원되지 IBM Semeru Runtime않습니다. 클라이언트 애플리케이션에 포함되어도 오류가 발생하지 않습니다. 속성이 설정되었지만 IBM Semeru RuntimeFIPS 속성들이 설정되지 않은 경우 새로운 JMSEException이 발생합니다. 기존 클라이언트 애플리케이션이 로 이동하여 Java 11+ 로 연결을 FIPS 기대하는 상황을 방지합니다.

참고: 이는 기존 사용자에게는 영향을 미치지 IBMJava 8 않으며, 오직 를 사용하는 고객에게만 영향을 IBM Semeru RuntimeJava 미칩니다.

해당 **SSLFIPSREQUIRED** 속성의 기본값은 NO 입니다. 이는 연결이 에서 지원하는 모든 IBM MQCipherSuite 을 사용할 수 있음을 의미합니다.

응용 프로그램이 하나 이상의 연결을 사용하는 경우, 응용 프로그램이 첫 번째 연결을 생성할 때 사용된 **SSLFIPSREQUIRED** 값이 이후 연결을 생성할 때 사용되는 값을 결정합니다. 이는 후속 연결 생성에 사용되는 연결 팩토리의 속성 **SSLFIPSREQUIRED** 값이 무시됨을 의미합니다. 다른 값을 사용하려면 애플리케이션을 **SSLFIPSREQUIRED** 다시 시작해야 합니다.

애플리케이션은 ConnectionFactory 오브젝트의 setSSLFipsRequired() 메소드를 호출하여 이 특성을 설정할 수 있습니다. CipherSuite가 설정되지 않은 경우에는 특성이 무시됩니다.

관련 태스크

[MQI 클라이언트에서 실행 시 인증된 CipherSpecs 만 FIPS 사용하도록 지정](#)

관련 참조

[SSLFIPSREQUIRED](#)

[Federal Information Processing Standards \(FIPS\)에 대해 AIX, Linux, and Windows](#)

/에서 IBM MQ classes for JMS SSLPEERNAME을 사용하여 고유 이름 패턴 지정Jakarta Messaging object **SSLPEERNAME** 속성을 사용하여 고유 식별자(DN) 패턴을 지정함으로써 애플리케이션이 JMS 올바른 큐 관리자에 연결되도록 합니다.

JMS 애플리케이션은 식별 이름(DN) 패턴을 지정하여 올바른 큐 관리자에 연결되는지 확인할 수 있습니다. 연결은 큐 관리자가 패턴과 일치하는 DN을 표시하는 경우에만 성공합니다. 이 패턴의 형식에 대한 자세한 정보는 관련 주제를 참조하십시오.

DN은 `ConnectionFactory` 객체의 `**SSLPEERNAME** setDN()` 속성을 사용하여 설정됩니다:

JMSAdmin 또는 JMSAdmin30 도구를 사용하여

예를 들어, 다음 명령은 ConnectionFactory 객체가 큐 관리자가 공통 이름(Common Name)을 다음과 같이 식별하도록 설정합니다 QMGR.: - 공통 이름은 문자로 시작해야 하며, - 최소 두 개의 조직 단위(Organizational WEBSPPHERE Unit) 이름을 가져야 합니다. - 첫 번째 조직 단위 이름은 이어야 IBM 하며, - 두 번째 조직 단위 이름은 이어야 합니다

```
ALTER CF(my.cf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPPHERE)
```

확인은 대소문자를 구분하며, 세미콜론을 쉼표 대신 사용할 수 있습니다.

어플리케이션에서

SSLPEERNAME 응용 프로그램에서 MQConnectionFactory 객체의 `**setSSLPeerName**` 메서드를 사용하여 설정할 수 있습니다.

이 특성이 설정되지 않으면 큐 관리자가 제공하는 식별 이름에서 확인이 수행되지 않습니다. CipherSuite가 설정되지 않은 경우에는 이 특성이 무시됩니다.

관련 개념

[식별 이름](#)

관련 태스크

[JMSAdmin 및 JMS30Admin 을 사용한 관리 명령어 실행](#)

관련 참조

[SSLPEERNAME](#)

/에 IBM MQ classes for JMS 있는 인증서 취소 목록 사용Jakarta Messaging

LDAP 서버 목록을 지정하여 인증서 해지 목록(CRL) 검사에 사용할 수 있도록 object **SSLCERTSTORES** 속성을 사용하십시오.

일반적으로는 인증서 폐기 목록(CRL)을 사용하여 더 이상 신뢰되지 않는 인증서를 식별합니다. 일반적으로 CRL은 LDAP 서버에서 호스팅됩니다. JMS는 Java 2 v1.4 이상에서 LDAP 서버가 CRL 확인을 위해 지정될 수 있도록

허용합니다. 다음 JMSAdmin 또는 JMSAdmin30 예제는 LDAP 서버(crl1.ibm.com:)에 호스팅된 CRL을 사용하여 JMS 지시합니다

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

참고: LDAP 서버에서 호스팅된 CRL에서 CertStore를 성공적으로 사용하려면 Java SDK(Software Development Kit)가 CRL과 호환 가능한지 확인하십시오. 일부 SDK에서는 LDAP v2에 대한 스키마를 정의하는 RFC 2587을 CRL이 준수하도록 요구합니다. 대부분의 LDAP v3 서버는 RFC 2256을 대신 사용합니다.

LDAP 서버가 기본 포트 389에서 실행 중이 아닌 경우, 호스트 이름에 콜론(:) 및 포트 번호를 추가하여 포트를 지정할 수 있습니다. 큐 관리자가 제시한 인증서가 crl1.ibm.com에서 호스팅된 CRL에 있으면 연결이 완료되지 않습니다. 단일 장애 지점을 피하기 위해, JMS에서는 공백 문자로 구분된 LDAP 서버의 목록을 제공하여 다중 LDAP 서버가 제공될 수 있도록 허용합니다. 다음은 예제입니다.

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

다중 LDAP 서버가 지정된 경우, JMS는 큐 관리자의 인증서가 성공적으로 확인될 수 있는 서버를 찾을 때까지 이를 하나씩 차례로 시도합니다. 각 서버에는 동일한 정보가 포함되어 있어야 합니다.

이 형식의 문자열은 MQConnectionFactory.setSSLCertStores() 메소드에서 애플리케이션에 의해 제공될 수 있습니다. 또는 애플리케이션이 하나 이상의 java.security.cert.CertStore 오브젝트를 작성하고 이를 적합한 Collection 오브젝트에 두며 이 Collection 오브젝트를 setSSLCertStores() 메소드에 제공할 수 있습니다. 이러한 방법으로 애플리케이션은 CRL 확인을 사용자 정의할 수 있습니다. CertStore 오브젝트의 구성 및 사용에 대한 세부사항은 JSSE 문서를 참조하십시오.

연결이 설정될 때 큐 관리자에 의해 표시되는 인증서는 다음과 같이 유효성 검증됩니다.

1. sslCertStores에 의해 식별되는 컬렉션 내의 첫 번째 CertStore 오브젝트는 CRL 서버를 식별하는 데 사용됩니다.
2. CRL 서버에 접속하려는 시도가 이루어집니다.
3. 시도가 성공하면 일치하는 인증서를 찾기 위해 서버가 검색됩니다.
 - a. 인증서가 폐기된 것으로 밝혀지면 검색 프로세스가 종료되며 이유 코드 MQRC_SSL_CERTIFICATE_REVOKED와 함께 연결 요청이 실패합니다.
 - b. 인증서를 찾지 못하는 경우에는 검색 프로세스가 종료되며 연결을 진행하도록 허용됩니다.
4. 서버에 접속하려는 시도가 실패하면 다음 CertStore 오브젝트를 사용하여 CRL 서버를 식별하며 프로세스가 2단계부터 반복됩니다.

컬렉션 내의 마지막 CertStore이거나 컬렉션에 CertStore 오브젝트가 없는 경우에는 검색 프로세스가 실패하고 이유 코드 MQRC_SSL_CERT_STORE_ERROR와 함께 연결 요청이 실패합니다.

컬렉션 오브젝트는 CertStores가 사용되는 순서를 판별합니다.

애플리케이션이 setSSLCertStores()를 사용하여 CertStore 오브젝트의 컬렉션을 설정하는 경우, MQConnectionFactory는 더 이상 JNDI 네임스페이스에 바인딩될 수 없습니다. 이를 수행하려고 시도하면 예외가 발생합니다. sslCertStores 특성이 설정되지 않은 경우에는 큐 관리자가 제공하는 인증서에서 폐기 확인이 수행되지 않습니다. CipherSuite가 설정되지 않은 경우에는 이 특성이 무시됩니다.

관련 태스크

[JMSAdmin 및 JMS30Admin 을 사용한 관리 명령어 실행](#)

관련 참조

SSLCRL

/에서 *IBM MQ classes for JMS* 비밀 키 재협상 *Jakarta Messaging*

객체 **SSLRESETCOUNT** 속성은 암호화에 사용되는 비밀 키가 재협상되기 전에 연결을 통해 송수신된 총 바이트 수를 나타냅니다.

송신된 바이트 수는 암호화 전의 바이트 수이며, 수신된 바이트 수는 복호화 후의 바이트 수입니다. 바이트 수에는 IBM MQ classes for JMS에 의해 송신되거나 수신된 제어 정보도 포함됩니다.

다음과 같은 방법으로 **SSLRESETCOUNT** 가능합니다:

JMSAdmin 또는 JMSAdmin30 도구를 사용하여

예를 들어, 4MB의 데이터가 전송된 후 비밀 키가 재협상되는 TLS 가 활성화된 MQI 채널을 통해 연결을 생성하는 데 사용할 수 있는 `ConnectionFactory` 객체를 구성하려면, JMSAdmin 또는 JMSAdmin30: 에 다음 명령을 실행하십시오

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

어플리케이션에서

애플리케이션은 ConnectionFactory 오브젝트의 setSSLResetCount() 메소드를 호출하여 이 특성을 설정할 수 있습니다.

이 특성의 값이 기본값인 0이면 비밀 키가 절대 재협상되지 않습니다. CipherSuite가 설정되지 않은 경우에는 특성이 무시됩니다.

관련 태스크

[JMSAdmin 및 JMS30Admin 을 사용한 관리 명령어 실행](#)

관련 참조

SSLRESETCOUNT

맞춤형 *SSLSocketFactory in IBM MQ classes for JMS/* 제공 *Jakarta Messaging*

애플리케이션은 큐 매니저에 대한 TLS 연결을 설정할 때 사용자 정의된 SSLSocketFactory 인스턴스를 제공할 수 있습니다. V 10.0.0 애플리케이션은 또한 HTTPS 엔드포인트에서 CCDT 또는 JWT를 가져올 때 사용자 정의된 SSLSocketFactory 를 제공할 수 있습니다.

TLS 연결의 특정 측면을 애플리케이션에 맞게 사용자 정의할 필요가 있을 수 있습니다. 예를 들어 다음과 같은 목적이 있을 수 있습니다:

- 암호화 하드웨어를 초기화합니다.
- 사용 중인 키스토어와 트러스트스토어를 변경하십시오.
- V 10.0.0 HTTPS 엔드포인트에서 CCDT 또는 JWT를 검색하고, JVM 내의 다른 HTTPS 연결에 영향을 주지 않으면서 신뢰 저장소(truststore), 키 저장소(keystore) 및 인증서 유효성 검사 로직을 제어할 수 있어야 합니다.

참고: 이 기능은 (JCA)을 Java EE Connector Architecture 통해 연결된 애플리케이션 서버에는 적용되지 않습니다. 와 같은 WebSphere Liberty 서버는 이미 연결을 필터링하는 메커니즘을 제공하기 때문입니다. CCDT 및 토큰 검색과 같은 개별 아웃바운드 HTTPS 연결에 대해 SSL 인증서 검증을 사용자 지정하려면, 문서의 WebSphere Liberty 'SSL 구성을 위한 아웃바운드 필터'에 설명된 대로 아웃바운드 필터를 사용하여 서버를 WebSphere Liberty 구성할 수 있습니다.

애플리케이션은 먼저 해당 환경에 맞게 커스터마이징된 애플리케이션 컨텍스트

(javax.net.ssl.SSLSocketFactory) 객체를 생성해야 합니다. 이 작업을 수행하는 방법에 대한 정보는 (JSSE) Java Secure Socket Extension 문서에서 확인하십시오. 사용자 정의 가능한 기능은 제공업체마다 다르기 때문입니다. 적당한 SSLSocketFactory 오브젝트를 확보한 후에는 MQConnectionFactory.setSSLSocketFactory() 메소드를 사용하여 사용자 정의된 SSLSocketFactory 오브젝트를 사용하도록 JMS를 구성하십시오.

애플리케이션이 setSSLSocketFactory() 메소드를 사용하여 사용자 정의된 SSLSocketFactory 오브젝트를 설정하는 경우, MQConnectionFactory object는 더 이상 JNDI 네임스페이스에 바인딩될 수 없습니다. 이를 수행하려고 시도하면 예외가 발생합니다. 특성이 설정되지 않은 경우에는 기본 SSLSocketFactory 오브젝트가 사용됩니다. 기본 `SSLSocketFactory` 객체의 동작에 대한 자세한 내용은 해당 JSSE 문서에서 확인하십시오. CipherSuite가 설정되지 않은 경우에는 이 특성이 무시됩니다.

중요: 자체 보안이 되지 않은 JNDI 네임스페이스에서 ConnectionFactory 오브젝트가 검색될 때 SSL 특성의 사용으로 보안이 보장된다고 가정하지 마십시오. 특히 JNDI의 표준 LDAP 구현은 안전하지 않습니다. 공격자는 LDAP 서버를 모방할 수 있으며, 모르는 사이에 JMS 애플리케이션이 올바르게 작동하지 않는 서버에 연결하도록 잘못 유도할 수 있습니다. 적당한 보안 배치가 적소에 마련되어 있으면 JNDI의 기타 구현(예: fscontext 구현)이 안전합니다.

V 10.0.0 JWT 검색을 위한 맞춤형 SSLSocketFactory 인스턴스 구성

사용자 정의 SSLSocketFactory 인스턴스를 제공하여, 애플리케이션이 HTTPS 엔드포인트에서 JWT 토큰을 안전하게 가져오는 방식을 제어할 수 있습니다. 이 사용자 정의 `SSLSocketFactory` 는 클라이언트 라이브러리가 IBM MQ classes for JMS `HTTPS` 연결을 통해 JWT를 가져올 때 수행할 검증 수준도 지정할 수 있습니다. 클

클라이언트가 완전한 검증을 수행해야 하는지, 호스트명 검사를 무시할 수 있는지, 또는 인증서 검증을 수행하지 않을 수 있는지 선택할 수 있습니다.

이 태스크 정보

클라이언트가 IBM MQ classes for JMS JVMSSL 설정을 사용하여 HTTPS 토큰 엔드포인트에서 토큰을 검색하는 경우, 이는 토큰 엔드포인트에 대한 인증서 처리를 분리하는 것을 방해합니다. 사용자 정의된 `SSLSocketFactory` 인스턴스가 제공되면, 이는 `JVM`의 기본 동작을 재정의합니다. SSL 설정. HTTPS 엔드포인트에서 JWT를 검색하기 위해 IBM MQ classes for JMS 라이브러리에서 별도의 사용자 지정 SSLSocketFactory 인스턴스를 제공하도록 속성을 **TOKEN_SSL_SOCKET_FACTORY** 사용함으로써, 애플리케이션에서 JMS 사용할 신뢰 저장소를 정의하고, 기본 JVM SSL 설정에 의존하지 않으며, 동일한 JVM에서 실행 중인 다른 애플리케이션에 영향을 주지 않고 HTTPS 처리를 사용자 지정할 수 있습니다.

JWT 토큰 서버 연결에 사용하려는 SSLSocketFactory 인스턴스는 큐 관리자 또는 CCDT 연결에 사용하는 인스턴스와 다를 수 있습니다.

인증서 유효성 검사는 사용자 정의 팩토리 내에서 정의된 로직에 따라 처리됩니다. 애플리케이션은 HTTPS 연결 중에 적용되는 유효성 검사 수준을 지정할 수 있습니다. **TOKEN_HTTPS_CERT_VALIDATION_POLICY** 인증서 유효성 검사 정책 속성을 지정하고 필수 옵션을 지정합니다. 옵션은 ANY, HOSTNAMECN (기본값) 또는 NONE 일 수 있습니다. 자세한 내용은 [을 참조하십시오 인증서 유효성 검사 정책 구성 IBM MQ](#).

인증서 유효성 검사 정책이 NONE 으로 설정된 경우, 애플리케이션은 모든 인증서를 신뢰하는 IBM MQ classes for JMS 라이브러리의 **HttpsSSLHandler** 제공하는 핸들러 구현을 사용합니다. 이 핸들러 구현은 모든 JWT HTTPS 연결에 적용되는 사용자 정의 JWT 인증 컨텍스트(SSLSocketFactory)를 제공함으로써 재정의됩니다.

사용자 정의된 SSLSocketFactory 인스턴스가 제공되지 않으면, 시스템은 기본적으로 's(JSSE)' 내장 인증서 Java Secure Socket Extension 검증 메커니즘을 사용합니다.

프로시저

1. 매장을 생성하고 맞춤 설정하세요.
애플리케이션은 SSLContext를 생성한 후 이를 통해 SSLCertificate(SSLSocketFactory)를 획득하는 것으로 시작합니다.
이 공장은 추후 JWT HTTPS 연결에 할당될 예정입니다.
2. 연결 IBM MQ classes for JMS 팩토리를 생성합니다.
이것은 인증 및 **SSLJmsConnectionFactory** 속성으로 구성됩니다.
3. 큐 관리자를 설정하십시오.
이것은 클라이언트가 토큰 검색 후 최종적으로 연결할 큐 관리자를 식별합니다.
4. JWT 매개변수를 구성하십시오.
이러한 속성은 클라이언트에게 JWT 토큰을 획득하는 방법을 지시합니다.
TOKEN_CLIENT_ID **TOKEN_CLIENT_SECRET** OAuth 자격 증명을 정의합니다.
TOKEN_ENDPOINT 토큰 검색에 사용되는 HTTPS 엔드포인트를 지정합니다.
5. JWT(HTTPS) 검색을 위해 특별히 맞춤화된 SSLSocketFactory 을 할당하십시오.
HTTPS 연결이 SSLSocketFactory 을 **TOKEN_ENDPOINT** 사용하는지 확인하고, 이는 JVM 전체의 SSL 설정에 영향을 미치지 않아야 합니다.
6. 인증서 유효성 검사 정책을 구성하십시오.
정책 값이 NONE 인 경우, 사용자 정의된 SSLSocketFactory 가 인증 방식을 재정의하지 않을 때 HTTPS 연결에 대한 호스트명 및 인증서 검사를 제어합니다. ANY 및 HOSTNAMECN에 대해서는 사용자 정의된 SSLSocketFactory's 존재 여부와 관계없이 정책이 적용됩니다.
7. 객체를 JMS 생성하고 연결을 시작합니다.
응용 프로그램은 이제 인증되었으며 메시지를 보내거나 받을 준비가 되었습니다.

예

다음 예시는 JWT 토큰(HTTPS) 엔드포인트용으로 맞춤 설정된 SSLSocketFactory 인스턴스를 구성하는 방법을 보여줍니다.

참고: 이 예제는 키스토어 및 트러스트스토어 위치를 포함한 기본적인 SSLSocketFactory 구현을 보여줍니다. 실제 운영 환경 요구사항에 맞게 수정하십시오.

```
public static void main(String[] args) {
    Connection connection = null;
    Session session = null;
    Destination destination = null;
    MessageProducer producer = null;

    // Construct and customize stores
    SSLContext sslContext = createSimpleSSLContext(); // Check method definition below
    SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();

    // Create a connection factory
    JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
    JmsConnectionFactory cf = ff.createConnectionFactory();

    // Set the queue manager name
    cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");

    // Set properties to get tokens
    cf.setStringProperty(WMQConstants.TOKEN_CLIENT_ID, "mykeycloakclient");
    cf.setStringProperty(WMQConstants.TOKEN_CLIENT_SECRET, "mykeycloaksecret");
    cf.setStringProperty(WMQConstants.TOKEN_ENDPOINT, "https://mykeycloak.server/oidc/token");

    // Set a custom sslSocketFactory instance for the https endpoint set in TOKEN_ENDPOINT
    // property above
    cf.setObjectProperty(WMQConstants.TOKEN_SSL_SOCKET_FACTORY, sslSocketFactory);

    // Set a level of certificate validation
    cf.setIntProperty(WMQConstants.TOKEN_HTTPS_CERT_VALIDATION_POLICY,
        MQConstants.MQ_HTTPSCERTVAL_ANY);

    // Create JMS objects
    connection = cf.createConnection();
    session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
    destination = session.createQueue("dev.queue.1");
    producer = session.createProducer(destination);

    // Start the connection
    connection.start();
}

// Sample method that constructs keystore and truststore
public static SSLContext createSimpleSSLContext() {
    try {
        KeyStore keyStore = KeyStore.getInstance("JKS");
        try (FileInputStream keyStoreStream = new FileInputStream("clientkey.jks")) { // Path to
            keyStore.load(keyStoreStream, "keystorepassword".toCharArray()); // Hardcoded password
            just for the sample
        }
        KeyStore trustStore = KeyStore.getInstance("JKS");
        try (FileInputStream trustStoreStream = new FileInputStream("clientkey.jks")) { // Path
            to truststore jks
            trustStore.load(trustStoreStream, "truststorepassword".toCharArray()); // Hardcoded
            password just for the sample
        }

        KeyManagerFactory kmf =
            KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
        kmf.init(keyStore, "keystorepassword".toCharArray()); // Hardcoded password just for the
        sample

        TrustManagerFactory tmf =
            TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
        tmf.init(trustStore);

        SSLContext context = SSLContext.getInstance("TLS"); // set protocol name
        context.init(kmf.getKeyManagers(), tmf.getTrustManagers(), new SecureRandom());
        return context;
    } catch (Exception e) {
        // handle exception
    }
}
```



```
    return null;
}
```

V 10.0.0 CCDT 파일 검색을 위한 맞춤형 *SSLSocketFactory* 구성

사용자 정의 *SSLSocketFactory* 인스턴스를 제공하여, 애플리케이션이 HTTPS 엔드포인트에서 CCDT 파일을 안전하게 가져오는 방식을 제어할 수 있습니다. 이 사용자 정의된 '*SSLSocketFactory*' 인스턴스는 라이브러리가 IBM MQ classes for JMS 파일을 가져올 때 수행할 유효성 검사 수준도 지정할 수 있습니다.

이 태스크 정보

클라이언트가 IBM MQ classes for JMS JVM SSL 설정을 기반으로 HTTPS 토큰 엔드포인트에서 파일을 검색하는 경우, 이는 CCDT 파일 엔드포인트에 대한 인증서 처리를 분리하는 것을 방해합니다. 사용자 정의된 '*SSLSocketFactory*' 인스턴스가 제공되면, 이는 'JVM'의 기본 동작을 재정의합니다. SSL 설정. HTTPS 엔드포인트에서 CCDT 파일을 검색하기 위해 IBM MQ classes for JMS 라이브러리에서 별도의 사용자 지정 *SSLSocketFactory* 인스턴스를 제공하도록 속성을 **WMQ_CCDT_SSL_SOCKET_FACTORY** 사용함으로써, 애플리케이션에서 JMS 사용할 신뢰 저장소를 정의하고, 기본 JVM SSL 설정에 의존하지 않으며, 동일한 환경에서 실행 중인 다른 애플리케이션에 영향을 주지 않고 HTTPS 처리를 사용자 지정할 수 있습니다.

CCDT 파일 서버 연결에 사용하려는 *SSLSocketFactory* 인스턴스는 큐 관리자나 JWT 연결에 사용하는 인스턴스와 다를 수 있습니다.

인증서 유효성 검사는 사용자 정의 팩토리 내에서 정의된 로직에 따라 처리됩니다. 애플리케이션은 HTTPS 연결 중에 적용되는 유효성 검사 수준을 지정할 수 있습니다. **WMQ_CCDT_HTTPS_CERT_VALIDATION_POLICY** 인증서 유효성 검사 정책 속성을 지정하고 필수 옵션을 지정합니다. 옵션은 ANY, HOSTNAMECN (기본값) 또는 NONE 일 수 있습니다. 자세한 내용은 [을 참조하십시오 인증서 유효성 검사 정책 구성 IBM MQ](#).

인증서 유효성 검사 정책이 NONE 으로 설정된 경우, 애플리케이션은 모든 인증서를 신뢰하는 IBM MQ classes for JMS 라이브러리의 **HttpsSSLHandler** 제공하는 핸들러 구현을 사용합니다. 이 핸들러 구현은 모든 JWT HTTPS 연결에 적용되는 사용자 정의 JWT 인증 컨텍스트(*SSLSocketFactory*)를 제공함으로써 재정의됩니다.

사용자 정의된 *SSLSocketFactory* 인스턴스가 제공되지 않으면, 시스템은 기본적으로 's(JSSE)' 내장 인증서 Java Secure Socket Extension 검증 메커니즘을 사용합니다.

프로시저

1. 매장을 생성하고 맞춤 설정하세요.

애플리케이션은 *SSLContext*를 생성한 후 이를 통해 *SSLCertificate*(*SSLSocketFactory*)를 획득하는 것으로 시작합니다.

이 '*SSL*' 컨텍스트는 나중에 '*SSLSocketFactory*' **WMQ_CCDT_SSL_SOCKET_FACTORY** 속성으로 설정할 '*'*를 생성하는 데 사용됩니다.

2. 연결 JMS 팩토리 생성

CCDT 검색 및 큐 관리자 연결에 필요한 구성을 포함하는 연결 JMS 팩토리가 생성됩니다.

3. 큐 관리자 이름을 설정합니다.

클라이언트는 속성을 **WMQ_QUEUE_MANAGER** 사용하여 토큰 검색 후 최종적으로 연결할 큐 관리자를 식별합니다.

4. CCDTURL을 설정하십시오.

HTTPS 속성을 **WMQ_CCDTURL** 사용하여 CCDT 파일을 검색할 엔드포인트를 설정하십시오.

5. CCDT에 사용자 정의 속성(*SSLSocketFactory*)을 할당하십시오.

SSLSocketFactory 를 속성을 **WMQ_CCDT_SSL_SOCKET_FACTORY** 사용하여 설정하십시오.

6. 인증서 유효성 검사 정책을 설정합니다.

certificate_validation_level 속성을 **WMQ_CCDT_HTTPS_CERT_VALIDATION_POLICY** 사용하여 인증서 유효성 검사 수준을 구성합니다.

정책 값이 NONE 인 경우, 사용자 정의된 *SSLSocketFactory* 가 인증 방식을 재정의하지 않을 때 HTTPS 연결에 대한 호스트명 및 인증서 검사를 제어합니다. ANY 및 HOSTNAMECN에 대해서는 사용자 정의된 *SSLSocketFactory*'s 존재 여부와 관계없이 정책이 적용됩니다.

7. 객체를 JMS 생성하고 연결을 시작합니다.

응용 프로그램은 이제 인증되었으며 메시지를 보내거나 받을 준비가 되었습니다.

예

다음 예제는 HTTPS 엔드포인트를 통해 CCDT용으로 사용자 정의된 SSLSocketFactory 인스턴스를 구성하는 방법을 보여줍니다.

참고: 이 예제는 키스토어 및 트러스트스토어 위치를 포함한 기본적인 SSLSocketFactory 구현을 보여줍니다. 실제 운영 환경 요구사항에 맞게 수정하십시오.

```
public static void main(String[] args) {
    Connection connection = null;
    Session session = null;
    Destination destination = null;
    MessageProducer producer = null;

    // Construct and customize stores
    SSLContext sslContext = createSimpleSSLContext(); // Check method definition below
    SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();

    // Create a connection factory
    JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
    JmsConnectionFactory cf = ff.createConnectionFactory();

    // Set the queue manager name
    cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");

    // Set CCDT url
    cf.setStringProperty(WMQConstants.WMQ_CCDTURL, "https://ccdturl.com/CCDT.json");

    // Set a custom sslSocketFactory instance for the https endpoint set in WMQ_CCDTURL property
    above
    cf.setObjectProperty(WMQConstants.WMQ_CCDT_SSL_SOCKET_FACTORY, sslSocketFactory);

    // Set a level of certificate validation
    cf.setIntProperty(WMQConstants.WMQ_CCDT_HTTPS_CERT_VALIDATION_POLICY,
        MQConstants.MQ_HTTPSCERTVAL_ANY);

    // Create JMS objects
    connection = cf.createConnection();
    session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
    destination = session.createQueue("dev.queue.1");
    producer = session.createProducer(destination);

    // Start the connection
    connection.start();
}

// Sample method that constructs keystore and truststore
public static SSLContext createSimpleSSLContext() {
    try {
        KeyStore keyStore = KeyStore.getInstance("JKS");
        try (FileInputStream keyStoreStream = new FileInputStream("clientkey.jks")) { // Path to
            keystore jks
            keyStore.load(keyStoreStream, "keystorepassword".toCharArray()); // Hardcoded password
            just for the sample
        }
        KeyStore trustStore = KeyStore.getInstance("JKS");
        try (FileInputStream trustStoreStream = new FileInputStream("clientkey.jks")) { // Path
            to truststore jks
            trustStore.load(trustStoreStream, "truststorepassword".toCharArray()); // Hardcoded
            password just for the sample
        }

        KeyManagerFactory kmf =
        KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
        kmf.init(keyStore, "keystorepassword".toCharArray()); // Hardcoded password just for the
        sample

        TrustManagerFactory tmf =
        TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
        tmf.init(trustStore);

        SSLContext context = SSLContext.getInstance("TLS"); // set protocol name
        context.init(kmf.getKeyManagers(), tmf.getTrustManagers(), new SecureRandom());
        return context;
    } catch (Exception e) {
```

```
// handle exception
}
return null;
}
```

/ 경로 *IBM MQ classes for JMS* 내 키스토어 또는 JSSE 트러스트스토어에 변경 사항 적용 *Jakarta Messaging* (JSSE) 키스토어 또는 Java Secure Socket Extension 트러스트스토어를 변경한 경우, 변경 사항이 적용되도록 하기 위해 특정 조치를 취해야 합니다.

키스토어 JSSE 또는 트러스트스토어의 내용을 변경하거나, 키스토어 또는 트러스트스토어 파일의 위치를 변경하는 경우, 당시 실행 *IBM MQ classes for JMS* 중인 애플리케이션은 변경 사항을 자동으로 반영하지 않습니다. 변경사항을 적용하려면 다음 조치를 수행해야 합니다.

- 애플리케이션이 모든 자체 연결을 닫고 연결 풀의 사용되지 않는 연결을 영구 삭제해야 합니다.
- 제공자가 JSSE 키스토어 및 트러스트스토어의 정보를 캐싱하는 경우, 이 정보를 새로 고쳐야 합니다.

이 조치가 수행된 후에는 애플리케이션이 자체 연결을 재작성할 수 있습니다.

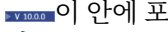
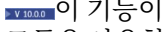
응용 프로그램의 설계 방식과 공급자가 JSSE 제공하는 기능에 따라, 응용 프로그램을 중지하고 다시 시작하지 않고도 이러한 작업을 수행할 수 있습니다. 그러나 애플리케이션을 중지하고 재시작하는 것이 가장 단순한 솔루션일 수 있습니다.

*IBM MQ classes for JMS*의 *TLS CipherSpecs* 및 *CipherSuites*

애플리케이션이 큐 매니저와 연결을 설정할 수 있는지 *IBM MQ classes for JMS* 여부는 MQI 채널의 서버 측에 지정된 *CipherSpec* 와 클라이언트 측에 지정된 *CipherSuite* 에 따라 달라집니다.

FIPS 준수

참고:

-  AIX, Linux, 및 Windows에서는 IBM MQ 암호 IBM Crypto for C (ICC) 모듈을 GSKit 9 통해 규정 준수를 제공합니다 FIPS 140-3 . 해당 FIPS 140-3 모듈과 관련된 NIST 인증서는 <https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4755> 에서 확인할 수 있습니다.
-  Linux s390xFIPS 현재.에 대한 지원이 제공되지 않습니다. IBM MQ이 플랫폼에서 지원이 필요한 FIPS 고객은 이전 버전을 계속 사용해야 합니다. 해당 IBM Crypto for C (ICC) 암호화 모듈이 이 플랫폼에서 인증을 받으면, 향후 FixPack 에서 FIPS 140-3 지원이 활성화될 예정입니다.
-  이 안에 포함된 암호화 FIPS 140-3 모듈은 IBM Semeru Runtime 2024년 8월 NIST의 승인을 받았습니다. IBM Semeru Runtime 11+IBM MQ 10.0.0 TLS 를 FIPS 140-3 사용하여 및 IBM MQ classes for Java 의 IBM MQ classes for JMS 및 내에서 Java 8 클라이언트 연결 처리를 지원합니다. <https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4755> 해당 FIPS 140-3 모듈과 관련된 NIST 인증서는 에서 확인할 수 있습니다.
- 컨테이너 환경의 IBM MQ 경우, IBM MQ Operator 3.2.0 및 9.4.0.0 이후 버전의 큐 매니저 컨테이너 이미지는 UBI 9를 기반으로 합니다. FIPS 140-3 컨테이너 내에서의 IBM MQ 규정 준수 여부는 현재 검토 중입니다.
-  이 기능이 FIPS 활성화되어 있으면, 컨테이너 IBM MQ 제어 프로세스에서 인증된 OpenSSL FIPS 140-3 모듈을 사용합니다. NIST 인증에 대한 자세한 내용은 다음 링크에서 확인하실 수 있습니다: <https://access.redhat.com/compliance/fips>. IBM MQ 컨테이너 이미지에서 실행되는 큐 관리자는 기본 이미지 플랫폼 버전의 인증 수준과 FIPS IBM MQ 동일합니다.

암호 모음이 준수 FIPS 140-2 (compliant)로 표시된 경우 애플리케이션이 준수 (compliance) FIPS 140-2 를 강제하도록 구성되지 않았다면 사용할 수 있습니다. 그러나 애플리케이션에 대해 준수 (FIPS 140-2 compliance)가 구성된 경우 (이 항목의 "암호 모음 및 FIPS 준수 구성" 섹션 참조), 호환 (FIPS 140-2 compatible)으로 표시된 암호 모음 (CipherSuites)만 구성할 수 있습니다. 다른 암호 모음 (CipherSuites)을 사용하려고 하면 오류가 발생합니다.

참고: 각각은 JRE 여러 암호화 보안 공급자를 가질 수 있으며, 각 공급자는 동일한 CipherSuite 의 구현을 제공할 수 있습니다. 그러나 모든 보안 업체가 FIPS 140-2 또는 FIPS 140-3 인증을 받은 것은 아닙니다. 특정 애플리케이션에 대해 FIPS 140-2 또는 FIPS 140-3 준수 요건이 적용되지 않는 경우, 인증되지 않은 보안 모듈 (CipherSuite)이 사용될 가능성이 있습니다. 인증되지 않은 구현의 경우, 해당 보안 모듈 (CipherSuite)이 이론적으로는 표준에서 요구하는 최소 보안 수준을 충족하더라도 FIPS 140-2 또는 FIPS 140-3을 준수하지 않을 수

있습니다. 258 페이지의 『애플리케이션에서 IBM MQ classes for JMS 암호화 모음 구성 및 FIPS 규정 준수』 애플리케이션에서 IBM MQJMS FIPS 140-2 또는 FIPS 140-3 적용을 구성하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

CipherSpecs 및 FIPS 140-2CipherSuites, 의 Suite-B 준수 관련 자세한 내용은 CipherSpecs 지정 항목을 참조하십시오. 미국과 관련된 정보도 알아야 Federal Information Processing Standards 할 수 있습니다.

CipherSuites 의 모든 기능을 활용하고 인증된 FIPS 140-2 환경 및/또는 Suite-B 규격에 부합하는 환경에서 운영하려면 적절한 JRE 장비가 필요합니다. IBM 254 페이지의 『IBM MQ에서 지원하는 CipherSpec 및 이와 동등한 CipherSuite』 Java 7 서비스 릴리스 4 픽스 팩 2 이상 버전은 IBM 에 나열된 JRECipherSuites 에 대한 적절한 지원을 제공합니다.

 IBM MQ 10.0.0TLS 를 사용하여 및 IBM MQ classes for Java 클라이언트 연결을 IBM MQ classes for JMS 처리하는 기능은 8.0.8.30, 11.0.24, 17.0.12, Java21.0.4 및 25.0.2 버전 이상에서 IBM Semeru Runtime 다음 플랫폼에 대한 FIPS 140-3 구현을 지원합니다:

-  AIX
-  Linux Linux for x86-64
-  Linux Linux on Power Systems - Little Endian
-  Windows

참고:

- 32비트 운영 체제는 지원되지 않습니다.
- Semeru 에서는 25가 FIPS 140-3 기본값이며, FIPS 140-2 이 옵션은 사용할 수 없습니다.
- 다른 런타임에 FIPS 140-2 대한 연결을 구성할 수 있습니다.

IBM MQ에서 지원하는 CipherSpec 및 이와 동등한 CipherSuite

다음 표는 지원되는 IBM MQ CipherSpecs 및 해당 CipherSuites 를 나열합니다. 이 표는 또한 통신에 사용되는 프로토콜과, 해당 통신 장치(CipherSuite)가 및 FIPS 140-3 FIPS 140-2 표준을 준수하는지 여부를 나타냅니다.

 사용되지 않는 CipherSpec 주제를 검토하여 다음 표에 나열된 CipherSpecs이 IBM MQ에서 더 이상 사용되지 않는지 확인하고, 그렇다면 CipherSpec이 더 이상 사용되지 않는 업데이트가 있는지 확인해야 합니다.

중요사항: 나열된 CipherSuites 는 (JRE)와 함께 IBMJava runtime environment 제공되는 것으로 IBM MQ 지원됩니다. 나열된 CipherSuites 에는 JavaJREOracle 에서 지원하는 항목들이 포함됩니다. 애플리케이션이 를 사용하도록 구성하는 OracleJavaJRE 방법에 대한 자세한 내용은 를 참조하십시오 애플리케이션이 또는 CipherSuiteJavaOracle IBMJava 매핑을 사용하도록 구성하기.

TLS1.3 암호화를 사용하려면 애플리케이션을 실행하는 JRE 시스템이 TLS1.3 를 지원해야 합니다.

암호화 Java 11방식 접두사는 상호 교환이 불가능하므로, 채널 및 클라이언트 정의 모두에 올바른 SSL_ 또는 TLS_ 접두사가 필요합니다.

참고: CipherSuites, 를 사용하려면 '제한 없는' 정책 파일을. JRE에 구성해야 합니다. SDK 또는 에서 정책 파일이 설정되는 방법에 대한 자세한 내용은 JRE 사용 중인 버전의 보안 참조: *IBM SDK, Java 기술 에디션* 에서 IBM SDK 보안 정책 파일 참조하십시오.

표 41. IBM MQ에서 지원하는 CipherSpec 및 이와 동등한 CipherSuite					
CipherSpec 257 페이지의 『1』	동등한 CipherSuite (IBM JRE)	동등한 CipherSuite (Oracle JRE)	프로토콜	FIPS 140-2 호환 가능	V10.0.0 FIPS 140-3 호환되는
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	V 10.0.0 TLS S_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA5	TLS 1.2	예	아니오
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	예	예
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	예	예
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	예	예
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	예	예
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	V 10.0.0 TLS S_ECDHE_ECDSA_WITH_NULL_SHA5	TLS 1.2	아니오	아니오
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	V 10.0.0 TLS S_ECDHE_ECDSA_WITH_RC4_128_SHA5	TLS 1.2	아니오	아니오
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	V 10.0.0 TLS S_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA5	TLS 1.2	예	아니오
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	예	예
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	예	예
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	예	예
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	예	예
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	V 10.0.0 TLS S_ECDHE_RSA_WITH_NULL_SHA5	TLS 1.2	아니오	아니오
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	V 10.0.0 TLS S_ECDHE_RSA_WITH_RC4_128_SHA5	TLS 1.2	아니오	아니오
TLS_RSA_WITH_3DES_EDE_CBC_SHA 257 페이지의 『2』 5 257 페이지의 『6』	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	아니오	아니오

표 41. IBM MQ에서 지원하는 CipherSpec 및 이와 동등한 CipherSuite (계속)

CipherSpec 257 페이지의 『1』	동등한 CipherSuite (IBM JRE)	동등한 CipherSuite (Oracle JRE)	프로토콜	FIPS 140-2 호환 가능	VIDEO FIPS 140-3 호환되는
TLS_RSA_WITH_AES_128_CBC_SHA 257 페이지의 『6』	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	아니 257 페이지의 『4』	아니 오
TLS_RSA_WITH_AES_128_CBC_SHA256 257 페이지의 『6』	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	아니 257 페이지의 『4』	아니 오
TLS_RSA_WITH_AES_128_GCM_SHA256 257 페이지의 『6』	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	아니 257 페이지의 『4』	아니 오
TLS_RSA_WITH_AES_256_CBC_SHA 257 페이지의 『6』	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	아니 257 페이지의 『4』	아니 오
TLS_RSA_WITH_AES_256_CBC_SHA256 257 페이지의 『6』	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	아니 257 페이지의 『4』	아니 오
TLS_RSA_WITH_AES_256_GCM_SHA384 257 페이지의 『6』	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	아니 257 페이지의 『4』	아니 오
TLS_RSA_WITH_DES_CBC_SHA 257 페이지의 『6』	SSL _RSA_WITH_DES_CBC_SHA 5	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	아니 오	아니 오
TLS_RSA_WITH_NULL_SHA 257 페이지의 『6』	SSL_RSA_WITH_NULL_SHA	TLS_RSA_WITH_NULL_SHA	TLS 1.2	아니 오	아니 오

표 41. IBM MQ에서 지원하는 CipherSpec 및 이와 동등한 CipherSuite (계속)					
CipherSpec 257 페이지의 『1』	동등한 CipherSuite (IBM JRE)	동등한 CipherSuite (Oracle JRE)	프로토콜	FIPS 140-2 호환 가능	V10.0.0 FIPS 140-3 호환 되는
TLS_RSA_WITH_RC4_128_SHA256 257 페이지의 『6』	SSL_RSA_WITH_RC4_128_SHA5	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	아니오	아니오
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	예	아니오
TLS_AES_128_GCM_SHA256 257 페이지의 『3』	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS 1.3	아니오	예
TLS_AES_256_GCM_SHA384 257 페이지의 『3』	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS 1.3	아니오	예
TLS_CHACHA20_POLY1305_SHA256 257 페이지의 『3』	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	아니오	아니오
TLS_AES_128_CCM_SHA256 257 페이지의 『3』	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS 1.3	아니오	아니오
TLS_AES_128_CCM_8_SHA256 257 페이지의 『3』	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS 1.3	아니오	아니오
임의 257 페이지의 『3』	*ANY	*ANY	다중	아니오	아니오
ANY_TLS13 257 페이지의 『3』	*TLS13	*TLS13	TLS 1.3	아니오	아니오
ANY_TLS12_OR_HIGHER 257 페이지의 『3』	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 이상	아니오	아니오
ANY_TLS13_OR_HIGHER 257 페이지의 『3』	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 이상	아니오	아니오

참고:

1. 이는 채널에 구성된 IBM MQ값으로, CCDT(바이너리 또는 JSON) 내에서도 포함됩니다.
2. **Deprecated** CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA는 더 이상 사용되지 않습니다. 그러나 연결이 오류로 종료되기 전까지 최대 32GB의 AMQ9288 데이터를 전송하는 데 여전히 사용할 수 있습니다. 이 오류를 방지하려면 3중 DES를 사용하지 않거나 이 CipherSpec을 사용할 때 비밀 키 재설정을 사용 가능하게 하십시오.
3. TLS1.3 암호화를 사용하려면 애플리케이션을 실행하는 (JRE) Java runtime environment 이 TLS1.3 를 지원해야 합니다.
4. 부터 IBM MQ 9.4.0, FIPS 모드에서 작동할 때는 RSA 키 교환에 대한 지원을 제거합니다 IBMJava 8JRE .
5. [V10.0.0](#) CipherSuites 는 IBM MQ 10.0.0 를 사용하는 IBM MQ 의 구성 IBM Semeru Runtime 25요소에서 Java 지원되지 않습니다. 이러한 구성 요소 중 하나를 사용하는 경우, 해당 구성 요소가. IBM Semeru Runtime 25NET 프레임워크에서 지원하는.NET 어셈블리 참조(CipherSuite)를 사용하도록 반드시 확인해야 합니다.
6. IBM Semeru Runtime 25TLS_RSA_* CipherSpecs 는 기본적으로 비활성화되어 있습니다.

애플리케이션에서 IBM MQ classes for JMS 암호화 모음 구성 및 FIPS 규정 준수

CipherSuite 를 사용하는 애플리케이션은 연결의 를 설정하기 위해 다음 IBM MQ classes for JMS 두 가지 방법 중 하나를 사용할 수 있습니다:

- ConnectionFactory 오브젝트의 setSSLCipherSuite 메소드를 호출합니다.
- 관리 IBM MQJMS 도구를 사용하여 ConnectionFactory 객체의 속성을 **SSLCIPHERSUITE** 설정하십시오.

해당 기능을 사용하는 애플리케이션은 다음 두 가지 방법 IBM MQ classes for JMS 중 하나로 규정 준수를 FIPS 140-2 강제할 수 있습니다:

- ConnectionFactory 오브젝트의 setSSLFipsRequired 메소드를 호출합니다.
- 관리 IBM MQJMS 도구를 사용하여 ConnectionFactory 객체의 속성을 **SSLFIPSREQUIRED** 설정하십시오.

V 10.0.0의 보안 공급자는 OpenJCEPlusFIPSFIPS 140-3 입니다. 다음 Java 속성을 제공함으로써 구현을 FIPS 140-3 활성화합니다:

```
-Dsemeru.fips=true -Dsemeru.customprofile=OpenJCEPlusFIPS
```

이렇게 하면 기본 보안 공급자가 OpenJCEPlusFIPS 로 설정됩니다. 254 페이지의 『IBM MQ에서 지원하는 CipherSpec 및 이와 동등한 CipherSuite』 연결은 해당 FIPS 140-3 구현에서 지원되는 CipherSuite 를 사용해야 합니다(참조).

참고:

- 클라이언트 연결이 사용하는 FIPS 버전을 확인하려면 시스템 속성 Java 을 com.ibm.fips.mode 쿼리하십시오. 이것은 또는 140-2 140-3 을 반환합니다.
- 로 마이그레이션할 때 기존 IBM MQ classes for JMS 및 IBM MQ classes for Java IBM MQ 10.0.0 클라이언트의 동작에는 아무런 변화가 없습니다. 그러나 FIPS 클라이언트를 에서 Java 8 로 마이그레이션하는 경우, 보안 FIPS 140-3 공급자를 활성화하기 위해 반드시 지정해야 하는 IBM Semeru Runtime 새로운 Java 속성을 반영하기 위해 클라이언트에 변경 사항을 적용해야 합니다.
- 해당 **SSLFIPSREQUIRED** 속성은 지원되지 IBM Semeru Runtime 않으며, FIPS 인증 연결을 생성하는 데 더 이상 필요하지 않습니다. 클라이언트가 IBM Semeru Runtime 11+ 런타임으로 연결 팩토리에 set을 **SSLFIPSREQUIRED** 설정하여 연결하지만 속성이 IBM Semeru RuntimeJava 지정되지 않은 경우, JMSSSL 구성 예외가 발생합니다. 이 예외는 적절한 보안 공급자가 사용 가능한 상태에서 인증된 FIPS 연결이 요청되었음을 나타냅니다. 자세한 정보는 JMS Exception Messages의 내용을 참조하십시오.
- CipherSuites 에서 Java 11 SSL_ 및 TLS_ 접두사를 상호 교환하여 사용할 수 있는 지원이 제거되었으므로 암호화 방식 정의가 올바르게 지정되어야 합니다.
- FIPS 140-3 IBM Semeru Runtime 32비트 클라이언트를 지원하지 않습니다. 32비트 클라이언트를 업데이트할 수 없는 경우, 오버라이드 파일을 제공하여 하드코딩된 연결 팩토리 속성을 재정의할 수 있습니다.

IBM Java 또는 Oracle Java CipherSuite �핑을 사용하도록 애플리케이션 구성

에서 IBM MQ 9.4.0, 암호(Cipher)는 암호화(CipherSpec) 또는 복호화(CipherSuite) 이름으로 정의될 수 있으며 에 의해 올바르게 IBM MQ 처리됩니다.

참고:  이전 버전의 에서 어떤 매핑을 사용할지 제어하던 시스템 com.ibm.mq.cfg.useIBMCipherMappings Java 속성은 IBM MQ 더 이상 필요하지 않아 에서 제품에서 IBM MQ 9.4.0 제거되었습니다.

상호 운용성 제한사항

사용 중인 프로토콜에 따라 특정 CipherSuite는 둘 이상의 IBM MQ CipherSpec과 호환 가능할 수 있습니다. 하지만 표 1에 지정된 TLS 버전을 사용하는 CipherSuite/CipherSpec 조합만 지원됩니다. CipherSuite 및 CipherSpec의 지원되지 않는 조합을 사용하려고 시도하면 해당되는 예외로 실패합니다. 이러한 CipherSuite/CipherSpec 조합을 사용한 설치의 지원되는 조합으로 이동되어야 합니다.

다음 표에는 이 제한사항이 적용되는 CipherSuite가 표시되어 있습니다.

표 42. CipherSuite 및 이의 지원 및 지원되지 않는 CipherSpec

CipherSuite	지원되는 TLS CipherSpec	지원되지 않는 SSL CipherSpec
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA A 259 페이지의 『1』	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

참고:

1. **Deprecated** 이 CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA는 더 이상 사용되지 않습니다. 그러나 AMQ9288 오류로 인해 연결이 종료되기 전까지는 이 CipherSpec을 사용하여 최대 32GB의 데이터를 전송할 수 있습니다. 이 오류를 방지하려면 3중 DES를 사용하지 않거나 이 CipherSpec을 사용할 때 비밀 키 재설정을 사용 가능하게 하십시오.

관련 개념

375 페이지의 『IBM MQ classes for Java의 TLS CipherSpecs 및 CipherSuites』

애플리케이션이 큐 매니저와 IBM MQ classes for Java 연결을 설정할 수 있는지 여부는 MQI 채널의 서버 측에 지정된 CipherSpec 와 클라이언트 측에 지정된 CipherSuite 에 따라 달라집니다.

채널 출력을 작성합니다. *Java IBM MQ classes for JMS*

채널 엑시트는 지정된 인터페이스를 구현하는 Java 클래스를 정의하여 작성됩니다.

보안 종료에 대한 소개를 보려면 [채널 보안 종료 프로그램 항목](#)부터 시작하십시오.

세 개의 인터페이스가 com.ibm.mq.exits 패키지에 정의됩니다.

- WMQSendExit: 송신 엑시트의 경우
- WMQReceiveExit: 수신 엑시트의 경우
- WMQSecurityExit: 보안 엑시트의 경우

다음의 샘플 코드는 세 인터페이스를 모두 구현하는 클래스를 정의합니다.

```
public class MyMQExits implements
    WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
    public MyMQExits(){
    }
    // This method implements the send exit interface
    public ByteBuffer channelSendExit(
        MQCXP channelExitParms,
        MQCD channelDefinition,
        ByteBuffer agentBuffer)
    {
        // Complete the body of the send exit here
    }
    // This method implements the receive exit interface
    public ByteBuffer channelReceiveExit(
        MQCXP channelExitParms,
        MQCD channelDefinition,
        ByteBuffer agentBuffer)
    {
        // Complete the body of the receive exit here
    }
    // This method implements the security exit interface
    public ByteBuffer channelSecurityExit(
        MQCXP channelExitParms,
        MQCD channelDefinition,
        ByteBuffer agentBuffer)
    {
        // Complete the body of the security exit here
    }
}
```

각 엑시트는 매개변수로서 MQCXP 오브젝트 및 MQCD 오브젝트를 수신합니다. 이러한 오브젝트는 절차적 인터페이스에 정의된 MQCXP 및 MQCD 구조를 표시합니다.

송신 엑시트가 호출된 경우, `agentBuffer` 매개변수에는 서버 큐 관리자에 송신될 예정인 데이터가 포함되어 있습니다. `agentBuffer.limit()` 표현식에서 데이터의 길이를 제공하므로 길이 매개변수는 필요하지 않습니다. 송신 엑시트는 서버 큐 관리자에 송신되는 데이터를 해당 값으로서 리턴합니다. 그러나 송신 엑시트가 송신 엑시트 시퀀스의 마지막 송신 엑시트가 아닌 경우에는 리턴된 데이터가 시퀀스의 다음 송신 엑시트로 대신 전달됩니다. 송신 엑시트는 `agentBuffer` 매개변수에서 수신된 데이터의 수정된 버전을 리턴하거나, 변경되지 않은 데이터를 리턴할 수 있습니다. 따라서 가장 단순한 가능한 엑시트 본문은 다음과 같습니다.

```
{ return agentBuffer; }
```

수신 엑시트가 호출되는 경우, `agentBuffer` 매개변수에는 서버 큐 관리자에서 수신된 데이터가 포함됩니다. 수신 엑시트는 IBM MQ classes for JMS에 의해 애플리케이션에 전달되는 데이터를 해당 값으로서 리턴합니다. 그러나 수신 엑시트가 수신 엑시트 시퀀스의 마지막 수신 엑시트가 아닌 경우에는 리턴된 데이터가 시퀀스의 다음 수신 엑시트로 대신 전달됩니다.

보안 엑시트가 호출될 때 `agentBuffer` 매개변수에는 연결의 서버 측에서 보안 엑시트의 보안 플로우에서 수신된 데이터가 포함되어 있습니다. 보안 엑시트는 보안 플로우에서 서버 보안 엑시트로 송신되는 데이터를 해당 값으로서 리턴합니다.

채널 엑시트는 백업 어레이가 있는 버퍼와 함께 호출됩니다. 최상의 성능을 위해 엑시트는 백업 어레이가 있는 버퍼를 리턴해야 합니다.

호출 시에 최대 32자의 사용자 데이터가 채널 엑시트에 전달될 수 있습니다. 엑시트는 MQCXP 오브젝트의 `getExitData()` 메소드를 호출하여 사용자 데이터에 액세스합니다. 엑시트가 `setExitData()` 메소드를 호출하여 사용자 데이터를 변경할 수 있지만, 사용자 데이터는 엑시트가 호출될 때마다 새로 고쳐집니다. 따라서 사용자 데이터에 대한 변경사항은 유실됩니다. 그러나 엑시트는 MQCXP 오브젝트의 엑시트 사용자 영역을 사용하여 하나의 호출에서 다음 호출로 데이터를 전달할 수 있습니다. 엑시트는 `getExitUserArea()` 메소드를 호출함으로써 참조에 의해 엑시트 사용자 영역에 액세스할 수 있습니다.

모든 엑시트 클래스에는 구성자가 있어야 합니다. 구성자는 이전 예제에서 표시된 대로 기본 구성자이거나 문자열 매개변수가 있는 구성자일 수 있습니다. 구성자는 클래스에 정의된 각 엑시트에 대한 엑시트 클래스의 인스턴스를 작성하기 위해 호출됩니다. 따라서 이전 예제에서는 `MyMQExits` 클래스의 인스턴스가 송신 엑시트에 대해 작성되고 다른 인스턴스가 수신 엑시트에 대해 작성되며 세 번째 인스턴스가 보안 엑시트에 대해 작성됩니다. 문자열 매개변수의 구성자가 호출된 경우, 매개변수에는 인스턴스가 작성되는 채널 엑시트에 전달되는 동일한 사용자 데이터가 포함됩니다. 엑시트에 기본 구성자 및 단일 매개변수 구성자가 모두 있는 경우에는 단일 매개변수 구성자가 우선합니다.

채널 엑시트 내에서 연결을 닫지 마십시오.

데이터가 연결의 서버 측에 송신된 경우, TLS 암호화는 채널 엑시트가 호출된 후에 수행됩니다. 이와 유사하게 데이터가 연결의 서버 측에서 수신된 경우, TLS 복호화는 채널 엑시트가 호출되기 전에 수행됩니다.

버전 IBM MQ classes for JMS 이전에는 채널 종료가 `MQSendExit`, IBM WebSphere MQ 7.0, `MQReceiveExit`, 및 `MQSecurityExit` 인터페이스를 사용하여 구현되었습니다. 이러한 인터페이스를 계속 사용할 수 있지만, 개선된 기능과 성능을 위해 새 인터페이스가 선호됩니다.

채널 엑시트를 사용하도록 IBM MQ classes for JMS 구성

IBM MQ classes for JMS 애플리케이션은 애플리케이션이 큐 관리자에 연결될 때 시작되는 MQI 채널에서 채널 보안, 송신 및 수신 엑시트를 사용할 수 있습니다. 해당 애플리케이션은 다음과 같이 작성된 종료 루틴을 사용할 수 있습니다. Java (32비트 또는 64비트) 또는 C/C++ (64비트만 해당). 또한 애플리케이션은 순차적으로 실행되는 송신 또는 수신 종료 루틴을 사용할 수도 있습니다.

다음 속성은 연결에서 JMS 사용되는 전송 종료점 또는 일련의 전송 종료점을 지정합니다:

- MQConnectionFactory 오브젝트의 **SENDEXIT** 특성.
- 리소스 IBM MQ 어댑터가 인바운드 통신에 사용하는 활성화 사양의 속성 **sendexit**,
- 리소스 IBM MQ 어댑터가 출력 통신에 사용하는 **sendexitConnectionFactory** 객체의 속성.

특성 값은 쉼표로 분리된 하나 이상의 항목으로 구성된 문자열입니다. 각 항목은 다음과 같은 방법 중 하나로 송신 엑시트를 식별합니다.

- 에서 작성된 전송 종료용 `WMQSendExitJava` 인터페이스를 구현하는 클래스의 이름.
- C 또는 C++로 작성된 송신 엑시트에 대한 `libraryName` 형식의 문자열(`entryPointName`).

유사한 방식으로 다음 특성은 연결에 사용되는 수신 엑시트 또는 수신 엑시트 순서를 지정합니다.

- MQConnectionFactory 오브젝트의 **RECEXIT** 특성.
- 리소스 IBM MQ 어댑터가 인바운드 통신에 사용하는 활성화 사양의 속성 **receiveexit**,
- 리소스 IBM MQ 어댑터가 출력 통신에 사용하는 **receiveexitConnectionFactory** 객체의 속성.

다음 특성은 연결에 사용되는 보안 엑시트를 지정합니다.

- MQConnectionFactory 오브젝트의 **SECEXIT** 특성.
- 리소스 IBM MQ 어댑터가 인바운드 통신에 사용하는 활성화 사양의 속성 **securityexit**,
- 리소스 IBM MQ 어댑터가 출력 통신에 사용하는 **securityexitConnectionFactory** 객체의 속성.

MQConnectionFactory, 의 경우, 관리 도구나 IBM MQJMS 를 사용하여 **SENDEXIT** IBM MQ Explorer, **RECEXIT** 및 **SECEXIT** 속성을 설정할 수 있습니다. 또는 애플리케이션이 **setSendExit()**, **setReceiveExit()** 및 **setSecurityExit()** 메소드를 호출하여 특성을 설정할 수 있습니다.

채널 엑시트는 자체 소유의 클래스 로더에 의해 로드됩니다. 채널 엑시트를 찾기 위해 클래스 로더는 지정된 순서로 다음 위치를 검색합니다.

1. 클라이언트 구성 IBM MQ 파일의 채널 스탠자 내 속성 **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** 또는 **JavaExitsClassPath** 속성으로 지정된 클래스 경로.
2. **Deprecated** Java 시스템 특성 **com.ibm.mq.exitClasspath**에서 지정하는 클래스 경로. 참고로, 이 특성은 이제 더 이상 사용되지 않습니다.
3. IBM MQ 엑시트 디렉토리(261 페이지의 표 43 참조). 클래스 로더는 우선 Java 아카이브(JAR) 파일에 패키징되지 않은 클래스 파일을 디렉토리에서 검색합니다. 채널 엑시트를 찾을 수 없는 경우, 클래스 로더는 디렉토리에서 JAR 파일을 검색합니다.

표 43. IBM MQ 엑시트 디렉토리	
플랫폼	디렉토리
 Linux  AIX AIX and Linux	/var/mqm/exits (32비트 Java 채널 종료) /var/mqm/exits64 (64비트 채널 종료)
 Windows Windows	install_data_dir\exits 여기서 <i>install_data_dir</i> 은 설치 도중 IBM MQ 데이터 파일에 대해 선택하는 디렉토리입니다. 기본 디렉토리는 C:\ProgramData\ IBM \ MQ입니다.

참고: 채널 엑시트가 둘 이상의 위치에 있으면, IBM MQ classes for JMS는 발견하는 첫 번째 인스턴스를 로드합니다.

클래스 로더의 상위는 IBM MQ classes for JMS를 로드하는 데 사용되는 클래스 로더입니다. 따라서 선행 위치에서 찾을 수 없는 경우에 상위 클래스 로더가 채널 엑시트를 로드할 수 있습니다. 그러나 애플리케이션 JEE 서버와 같은 환경에서 IBM MQ classes for JMS 를 사용할 때는 부모 클래스 로더의 선택에 영향을 미칠 수 없으므로, 애플리케이션 서버에서 시스템 **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** 속성 Java 을 설정하여 클래스 로더를 구성해야 합니다.

Java security manager가 사용 가능한 상태로 애플리케이션이 실행 중인 경우, 애플리케이션이 실행 중인 Java 런타임 환경에서 사용하는 정책 구성 파일에는 채널 엑시트 클래스를 로드할 수 있는 권한이 있어야 합니다. 이 작업을 수행하는 방법에 대한 정보는 [보안 Java 관리자에서 JMS 애플리케이션을 위한 IBM MQ 클래스 실행을 참조하십시오.](#)

버전 이전에 제공된, MQReceiveExit, 및 MQSecurityExit MQSendExit IBM WebSphere MQ 7.0 인터페이스는 여전히 지원됩니다. 이러한 인터페이스를 구현하는 채널 엑시트를 사용하는 경우 com.ibm.mq.jar이 (가) 클래스 경로에 있어야 합니다.

C로 채널 엑시트를 작성하는 방법에 대한 정보는 710 페이지의 『메시지 채널에 대한 채널 엑시트 프로그램』의 내용을 참조하십시오. 261 페이지의 표 43에 표시된 디렉토리에 C 또는 C++로 작성된 채널 엑시트 프로그램을 저장해야 합니다.

애플리케이션이 클라이언트 채널 정의 테이블(CCDT)을 사용하여 큐 관리자에 연결하는 경우에는 262 페이지의 『IBM MQ classes for JMS에서 클라이언트 채널 정의 테이블 사용』의 내용을 참조하십시오.

IBM MQ classes for JMS 사용 시에 채널 엑시트에 전달되는 사용자 데이터 지정 호출 시에 최대 32자의 사용자 데이터가 채널 엑시트에 전달될 수 있습니다.

MQConnectionFactory 오브젝트의 SENDXITINIT 특성은 호출 시에 각 송신 엑시트에 전달되는 사용자 데이터를 지정합니다. 특성 값은 쉼표로 분리된 하나 이상의 사용자 데이터 항목으로 구성된 문자열입니다. 문자열 내에서 사용자 데이터의 각 항목의 위치는 송신 엑시트의 시퀀스에서 사용자 데이터가 전달되는 송신 엑시트를 판별합니다. 예를 들어, 문자열에서 사용자 데이터의 첫 번째 항목은 송신 엑시트의 시퀀스에서 첫 번째 송신 엑시트에 전달됩니다.

IBM MQ JMS 관리 도구 또는 IBM MQ Explorer를 사용하여 SENDXITINIT 특성을 설정할 수 있습니다. 또는 애플리케이션이 setSendExitInit() 메소드를 호출하여 특성을 설정할 수 있습니다.

유사한 방법으로, ConnectionFactory 오브젝트의 RECEXITINIT 특성은 각 수신 엑시트에 전달된 사용자 데이터를 지정하며 SECXITINIT 특성은 보안 엑시트에 전달된 사용자 데이터를 지정합니다. IBM MQ JMS 관리 도구 또는 IBM MQ Explorer를 사용하여 이러한 특성을 설정할 수 있습니다. 또는 애플리케이션이 setReceiveExitInit() 및 setSecurityExitInit() 메소드를 호출하여 특성을 설정할 수 있습니다.

채널 엑시트에 전달되는 사용자 데이터를 지정할 때 다음 규칙을 참고하십시오.

- 문자열에서 사용자 데이터의 항목 수가 시퀀스의 엑시트 수를 초과하는 경우에는 사용자 데이터의 초과 항목이 무시됩니다.
- 문자열에서 사용자 데이터의 항목 수가 시퀀스의 엑시트 수 미만인 경우에는 사용자 데이터의 각 미지정 항목이 빈 문자열로 설정됩니다. 문자열 내의 두 개의 연속 쉼표 또는 문자열 맨 앞의 쉼표 역시 사용자 데이터의 미지정 항목을 표시합니다.

애플리케이션이 클라이언트 채널 정의 테이블(CCDT)을 사용하여 큐 관리자에 연결하는 경우, 클라이언트 연결 채널 정의에 지정된 사용자 데이터가 호출 시에 채널 엑시트에 전달됩니다. 클라이언트 채널 정의 테이블 사용에 대한 자세한 정보는 262 페이지의 『IBM MQ classes for JMS에서 클라이언트 채널 정의 테이블 사용』의 내용을 참조하십시오.

IBM MQ classes for JMS에서 클라이언트 채널 정의 테이블 사용

IBM MQ classes for JMS 애플리케이션은 클라이언트 채널 정의 테이블(CCDT)에 저장된 클라이언트 연결 채널 정의를 사용할 수 있습니다. 사용자는 CCDT를 사용하도록 ConnectionFactory 오브젝트를 구성합니다. 이의 사용에는 일부 제한사항이 있습니다.

ConnectionFactory 오브젝트의 특정 특성을 설정하여 클라이언트 연결 채널 정의를 작성하는 대안으로서, IBM MQ classes for JMS 애플리케이션은 클라이언트 채널 정의 테이블에 저장된 클라이언트 연결 채널 정의를 사용할 수 있습니다. 이러한 정의는 MQSC(IBM MQ Script) 명령 또는 IBM MQ PCF(Programmable Command Format) 명령으로 작성됩니다. 애플리케이션이 연결 오브젝트를 작성하는 경우, IBM MQ classes for JMS는 클라이언트 채널 정의 테이블에서 적당한 클라이언트 연결 채널 정의를 검색하며 채널 정의를 사용하여 MQI 채널을 시작합니다. 클라이언트 채널 정의 테이블 및 이를 구성하는 방법에 대한 자세한 정보는 클라이언트 채널 정의 테이블을 참조하십시오.

클라이언트 채널 정의 테이블을 사용하려면, 'ConnectionFactory' 객체의 'property' 속성인 'CCDTURL' 속성을 'URL' 객체로 설정해야 합니다. IBM MQ MQI 클라이언트 구성 파일에서 몇몇 다른 값이 사용되지만 IBM MQ classes for JMS는 IBM MQ MQI client 구성 파일 파일에서 CCDT에 대한 정보를 읽지 않습니다(적용되는 값은 98 페이지의 『Jakarta Messaging 구성 IBM MQ classes for JMS 파일』의 내용 참조). URL 오브젝트는 클라이언트 채널 정의 테이블이 포함된 파일의 이름과 위치를 식별하고 파일에 액세스하는 방법을 지정하는 URL(Uniform Resource Locator)을 캡슐화합니다. 해당 CCDTURL 속성은 관리 IBM MQJMS 도구를 사용하여 설정할 수 있으며, 애플리케이션은 'URL' 객체를 생성하고 'ConnectionFactory' 객체의 'setCCDTURL(' 메서드를 호출하여 해당 속성을 설정할 수 있습니다.

예를 들어, 파일에 클라이언트 채널 `ccdt1.tab` 정의 테이블이 포함되어 있고 애플리케이션이 실행 중인 동일한 시스템에 저장되어 있다면, 애플리케이션은 다음과 같은 방식으로 속성을 **CCDTURL** 설정할 수 있습니다:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
factory.setCCDTURL(chanTab1);
```

다른 예로, `ccdt2.tab` 파일에 클라이언트 채널 정의 테이블이 포함되어 있고 애플리케이션이 실행 중인 시스템과 다른 시스템에 저장된다고 가정하십시오. FTP 프로토콜을 사용하여 파일에 접근할 수 있다면, 애플리케이션은 다음과 같은 방식으로 속성을 **CCDTURL** 설정할 수 있습니다:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
factory.setCCDTURL(chanTab2);
```

ConnectionFactory 객체의 속성을 **CCDTURL** 설정하는 것 외에도, 동일한 객체의 속성은 **QMANAGER** 다음 값 중 하나로 설정되어야 합니다:

- 큐 관리자의 이름
- 별표(*) 및 큐 관리자 그룹의 이름

이러한 값은 MQI(Message Queue Interface)를 사용하는 클라이언트 애플리케이션에 의해 발행되는 MQCONN 호출의 **QMgrName** 매개변수에 대해 사용될 수 있는 동일한 값입니다. 따라서 이러한 값의 의미에 대한 자세한 정보는 **MQCONN**을 참조하십시오. 관리 도구나 IBM MQJMS를 사용하여 속성을 **QMANAGER** IBM MQ Explorer 설정할 수 있습니다. 또는 애플리케이션은 ConnectionFactory 오브젝트의 `setQueueManager()` 메소드를 호출하여 특성을 설정할 수 있습니다.

이 경우 애플리케이션은 'ConnectionFactory' 객체로부터 'Connection' 객체를 생성하고, IBM MQ classes for JMS 'CCDTURL' property로 식별되는 클라이언트 채널 정의 테이블에 접근한 다음, 'property'를 **QMANAGER** 사용하여 테이블에서 적합한 클라이언트 연결 채널 정의를 검색하고, 해당 채널 정의를 활용해 큐 매니저로 연결되는 'MQI' 채널을 시작합니다.

참고: 애플리케이션이 'createConnection()' 메소드를 호출할 때 'ConnectionFactory' 객체의 'CCDTURL' 및 'CHANNEL' 속성을 동시에 설정할 수 없습니다. 두 특성이 모두 설정되면 메소드에서 예외가 발생합니다. 해당 **CHANNELCCDTURL** 속성의 값이 null, 빈 문자열 또는 모두 공백 문자로 구성된 문자열이 아닌 경우, 해당 속성은 설정된 것으로 간주됩니다.

가 클라이언트 채널 정의 테이블에서 적합한 클라이언트 채널 정의 항목을 찾으면 IBM MQ classes for JMS, 해당 테이블에서 추출한 정보만을 사용하여 MQI 채널을 시작합니다. ConnectionFactory 오브젝트의 특성과 관련된 채널은 무시됩니다.

특히, TLS를 사용 중인 경우에는 다음과 같은 점에 유의하십시오.

- MQI 채널은 클라이언트 채널 정의 테이블에서 추출된 채널 정의가 IBM MQ classes for JMS에서 지원하는 CipherSpec의 이름을 지정하는 경우에만 TLS를 사용합니다.
- 또한 클라이언트 채널 정의 테이블에는 인증서 폐기 목록(CRL)을 보유하는 LDAP(Lightweight Directory Access Protocol) 서버의 위치에 대한 정보도 포함되어 있습니다. IBM MQ classes for JMS는 이 정보만을 사용하여 CRL을 보유하는 LDAP 서버에 액세스합니다.
- 클라이언트 채널 정의 테이블에는 OCSP 응답자의 위치도 포함할 수 있습니다. IBM MQ classes for JMS는 클라이언트 채널 정의 테이블 파일의 OCSP 정보를 사용할 수 없습니다. 그러나 Java 및 JMS 클라이언트 애플리케이션의 OCSP(Online Certificate Status Protocol) 절에서 설명한 대로 OCSP를 구성할 수 있습니다.

클라이언트 채널 정의 테이블에서 TLS 사용에 대한 자세한 정보는 [TLS 채널에서 확장 트랜잭션 클라이언트 사용](#)을 참조하십시오.

채널 엑시트를 사용하는 경우, 다음 사항도 참고하십시오.

- MQI 채널은 클라이언트 채널 정의 테이블에서 추출된 채널 정의에 의해 지정된 채널 엑시트 및 연관된 사용자 데이터만 사용합니다.
- 클라이언트 채널 정의 테이블에서 추출된 채널 정의는 Java로 작성된 채널 엑시트를 지정할 수 있습니다. 예를 들어, 클라이언트 연결 채널 정의를 생성하는 명령어의 **DEFINE CHANNEL SCYEXIT** 매개변수는 WMQSecurityExit 인터페이스를 구현하는 클래스의 이름을 지정할 수 있음을 의미합니다. 마찬가지로, **SENDEXIT** 매개변수는 WMQSendExit 인터페이스를 구현하는 클래스의 이름을 지정할 수 있으며, **RCVEXIT**

매개변수는 WMQReceiveExit 인터페이스를 구현하는 클래스의 이름을 지정할 수 있습니다. Java에서 채널 엑시트를 작성하는 방법에 대한 자세한 정보는 259 페이지의 『채널 출력을 작성합니다. Java IBM MQ classes for JMS』의 내용을 참조하십시오.

Java 이외의 언어로 작성된 채널 엑시트의 사용도 지원됩니다. 다른 언어로 작성된 채널 종료 **DEFINE CHANNEL** 명령어에서, **SENDEXIT**, 및 **RCVEXIT SCYEXIT** 매개변수를 지정하는 방법에 대한 정보는 **DEFINE CHANNEL**을 참조하십시오.

참고: V 10.0.0 SSLSocketFactory 에서 사용자 정의 설정을 'application.security. HTTPSIBM MQ 9.4.5' **WMQ_CCDT_SSL_SOCKET_FACTORY** 속성을 통해 추가할 수 있으며, 이를 통해 애플리케이션은 자체 구현을 통해 연결 보안 및 인증서 유효성 검사를 관리할 수 있습니다. 클라이언트 애플리케이션이 속성 **WMQ_CCDT_SSL_SOCKET_FACTORY** 제공하는 경우, 속성은 필수 **CCDTURL** 속성이 되며, 해당 속성이 누락된 경우 클라이언트 애플리케이션에 오류 메시지가 반환됩니다. 자세한 내용은 /의 IBM MQ classes for JMS 사용자 정의된 Jakarta MessagingSSLSocketFactory 제공을 참조하십시오.

자동 JMS 클라이언트 재연결

네트워크, 큐 관리자 또는 서버 장애 이후 자동으로 다시 연결하도록 JMS 클라이언트를 구성합니다.

일반적으로, 클라이언트 전송을 사용하여 독립형 IBM MQ classes for JMS 애플리케이션이 큐 관리자에 연결되고 일부 이유(예: 네트워크 가동 중단, 큐 관리자 장애 또는 큐 관리자 중지)로 인해 큐 관리자가 사용 불가능하게 되는 경우에는 다음 번에 애플리케이션이 큐 관리자와 통신을 시도할 때 IBM MQ classes for JMS에서 JMSException이 발생합니다. 애플리케이션은 JMSException을 발견하고 큐 관리자에 다시 연결을 시도해야 합니다. 사용자는 자동 클라이언트 다시 연결을 사용하여 애플리케이션의 디자인을 단순화할 수 있습니다. 큐 관리자가 사용 불가능하게 될 때 IBM MQ classes for JMS는 애플리케이션 대신 자동으로 큐 관리자에 대한 재연결을 시도합니다. 이는 애플리케이션에 다시 연결하기 위한 로직이 포함될 필요가 없음을 의미합니다.

자동 클라이언트 다시 연결의 이러한 구현은 Java Platform, Enterprise Edition 애플리케이션 서버 내에서 지원되지 않습니다. 대체 구현은 269 페이지의 『Java EE 환경에서 자동 클라이언트 다시 연결 사용』의 내용을 참조하십시오.

자동 JMS 클라이언트 재연결 사용

독립형 IBM MQ classes for JMS 애플리케이션이 CONNECTIONNAMELIST 또는 e CCDTURL 특성이 설정된 연결 팩토리를 사용하는 경우, 애플리케이션은 자동 클라이언트 다시 연결을 사용할 자격이 있습니다.

자동 클라이언트 다시 연결은 고가용성(HA) 구성의 일부인 큐 관리자를 포함한 큐 관리자에 다시 연결하는 데 사용할 수 있습니다. HA 구성에는 다중 인스턴스 큐 관리자, RDQM 큐 관리자 또는 IBM MQ 어플라이언스의 HA 큐 관리자가 포함됩니다.

IBM MQ classes for JMS에서 제공하는 자동 클라이언트 다시 연결 기능의 작동은 다음의 특성에 따라 다릅니다.

JMS 연결 팩토리 특성 TRANSPORT(단축 이름 TRAN)

TRANSPORT는 연결 팩토리를 사용하는 애플리케이션이 큐 관리자에 연결하는 방법을 지정합니다. 이 특성은 자동 클라이언트 다시 연결이 사용될 수 있도록 CLIENT 값으로 설정되어야 합니다. TRANSPORT 특성이 BIND, DIRECT 또는 DIRECTHTTP로 설정된 연결 팩토리를 사용하는 큐 관리자에 연결된 애플리케이션은 자동 클라이언트 다시 연결을 사용할 수 없습니다.

JMS 연결 팩토리 특성 QMANAGER(단축 이름 QMGR)

QMANAGER 특성은 연결 팩토리가 연결된 큐 관리자의 이름을 지정합니다.

JMS 연결 팩토리 특성 CONNECTIONNAMELIST(단축 이름 CRHOSTS)

CONNECTIONNAMELIST 특성은 심표로 분리된 목록이며, 여기서 각 항목에는 사용자가 CLIENT 전송을 사용 중일 때 QMANAGER 특성에 의해 지정된 큐 관리자에 연결하는 데 사용되는 호스트 이름 및 포트에 관한 정보가 포함되어 있습니다. 목록의 형식은 host name(port), host name(port)입니다.

JMS 연결 팩토리 특성 CCDTURL(단축 이름 CCDT)

CCDTURL 특성은 CCDT를 사용하여 큐 관리자에 연결할 때 IBM MQ classes for JMS가 사용하는 클라이언트 채널 정의 테이블을 지시합니다.

JMS 연결 팩토리 특성 CLIENTRECONNECTOPTIONS(단축 이름 CROPT)

CLIENTRECONNECTOPTIONS는 큐 관리자가 사용 가능해질 때 애플리케이션 대신 IBM MQ classes for JMS가 자동으로 큐 관리자에 연결을 시도하는지 여부를 제어합니다.

클라이언트 구성 파일의 채널 스탠자에 있는 DefRecon 속성

DefRecon 속성은 모든 애플리케이션의 자동 재연결을 허용하거나 자동 재연결을 위해 작성된 애플리케이션의 자동 재연결을 허용하지 않는 관리 옵션을 제공합니다.

자동 클라이언트 다시 연결은 애플리케이션이 큐 관리자에 성공적으로 연결되는 경우에만 사용 가능합니다.

애플리케이션이 CLIENT 전송을 사용하는 큐 관리자에 연결할 때 애플리케이션이 연결된 큐 관리자가 사용 불가능하게 되는 경우, IBM MQ classes for JMS는 연결 팩토리 특성 CLIENTRECONNECTOPTIONS의 값을 사용하여 자동 클라이언트 다시 연결을 사용하는지 여부를 판별합니다. 표 1은 CLIENTRECONNECTOPTIONS 특성의 가능한 값과 이 각각의 값에 대한 IBM MQ classes for JMS의 작동을 표시합니다.

표 44. 가능한 CLIENTRECONNECTOPTIONS 특성 값.	
CLIENTRECONNECTOPTIONS	IBM MQ classes for JMS의 작동
ANY	CONNECTIONNAMELIST가 설정되면, CONNECTIONNAMELIST 특성 값을 사용하여 호스트 이름 및 포트 조합에 대한 연결을 열고 임의의 큐 관리자에 연결합니다. 이 자동 클라이언트 다시 연결 옵션을 사용하려면 QMANAGER 특성이 기본값 또는 "*" 중 하나로 설정되어야 합니다. CCDTURL이 설정되면, CCDTURL 특성에서 지정하는 클라이언트 채널 정의 테이블을 열고 테이블의 항목을 선택한 후에 해당 항목을 사용하여 큐 관리자에 대한 클라이언트 연결 채널을 시작합니다. 이 자동 클라이언트 다시 연결 옵션을 사용하려면 QMANAGER 특성이 다음 중 하나로 설정되어야 합니다. • 별표(*) • 별표(*) 및 큐 관리자 그룹의 이름 • 비어 있는 문자열 또는 전체가 공백 문자인 문자열
ASDEF	DefRecon 값을 사용하여 자동 클라이언트 다시 연결이 사용 가능한지 여부를 판별합니다.
DISABLED	자동 클라이언트 다시 연결을 수행하지 않고 JMSException을 애플리케이션에 리턴합니다.
QMGR	클라이언트를 반드시 동일한 큐 관리자에 다시 연결하도록 지정합니다. 이 옵션은 동일한 큐 관리자의 다른 인스턴스에 다시 연결해야 하는 고가용성 솔루션에 사용해야 합니다. CONNECTIONNAMELIST가 설정되면, CONNECTIONNAMELIST가 설정되면, CONNECTIONNAMELIST 특성 값을 사용하여 호스트 이름 및 포트 조합에 대한 연결을 열고 임의의 큐 관리자에 연결합니다. CCDTURL이 설정되면, CCDTURL 특성에서 지정하는 클라이언트 채널 정의 테이블을 열고 QMANAGER 특성에서 지정하는 큐 관리자 이름과 일치하는 테이블의 항목을 찾은 후에 해당 항목을 사용하여 해당 큐 관리자에 대한 클라이언트 연결 채널을 시작합니다.

CONNECTIONNAMELIST가 설정되면, 자동 클라이언트 다시 연결을 수행하는 경우에 IBM MQ classes for JMS는 연결 팩토리 특성 CONNECTIONNAMELIST의 정보를 사용하여 재연결되는 시스템을 판별합니다.

IBM MQ classes for JMS는 초기에 CONNECTIONNAMELIST의 첫 번째 항목에 지정된 호스트 이름 및 포트를 사용하여 재연결을 시도합니다. 연결이 작성된 경우, IBM MQ classes for JMS는 QMANAGER 특성에 지정된 이름을 갖는 큐 관리자에 연결을 시도합니다. 큐 관리자에 대한 연결을 설정할 수 있는 경우, IBM MQ classes for JMS는 자동 클라이언트 다시 연결을 하기 전에 애플리케이션이 연 모든 IBM MQ 오브젝트를 다시 열고 이전과 같이 실행을 계속합니다.

CONNECTIONNAMELIST의 첫 번째 항목을 사용하여 필수 큐 관리자에 대한 연결을 설정할 수 없는 경우, IBM MQ classes for JMS는 CONNECTIONNAMELIST의 두 번째 항목을 시도합니다. 이 작업이 계속됩니다.

IBM MQ classes for JMS가 CONNECTIONNAMELIST의 모든 항목을 시도한 경우, 이는 다시 재연결을 시도하기 전에 일정 시간 동안 대기합니다. 새 재연결 시도를 수행하기 위해 IBM MQ classes for JMS는 CONNECTIONNAMELIST의 첫 번째 항목으로 시작합니다. 그리고 이는 재연결이 발생하거나 CONNECTIONNAMELIST의 끝에 도달할 때까지(이 시점에 IBM MQ classes for JMS는 다시 시도하기 전에 일정 기간을 대기함) CONNECTIONNAMELIST의 각 항목을 차례로 시도합니다.

CCDTURL이 설정되면, 자동 클라이언트 다시 연결을 수행하는 경우에 IBM MQ classes for JMS는 CCDTURL 특성에서 지정하는 클라이언트 채널 정의 테이블을 사용하여 다시 연결할 시스템을 판별합니다.

IBM MQ classes for JMS는 처음에 클라이언트 채널 정의 테이블을 구문 분석하고 QMANAGER 특성의 값과 일치하는 적당한 항목을 찾습니다. 항목을 찾으면 IBM MQ classes for JMS는 해당 항목을 사용하여 필수 큐 관리자에 다시 연결합니다. 큐 관리자에 대한 연결을 설정할 수 있는 경우, IBM MQ classes for JMS는 자동 클라이언트 다시 연결을 하기 전에 애플리케이션이 연 모든 IBM MQ 오브젝트를 다시 열고 이전과 같이 실행을 계속합니다.

필수 큐 관리자에 대한 연결을 설정할 수 없는 경우, IBM MQ classes for JMS는 클라이언트 채널 정의 테이블에서 다른 적당한 항목을 찾고 이를 사용하려고 시도합니다. 이 작업이 계속됩니다.

IBM MQ classes for JMS가 클라이언트 채널 정의 테이블에서 모든 적합한 항목을 시도한 경우, 이는 다시 재연결을 시도하기 전에 일정 시간 동안 대기합니다. 새 재연결 시도를 수행하기 위해, IBM MQ classes for JMS는 클라이언트 채널 정의 테이블을 다시 구문 분석하고 첫 번째 적당한 항목을 시도합니다. 그리고 이는 재연결이 발생하거나 클라이언트 채널 정의 테이블의 마지막 적당한 항목이 시도될 때까지(이 시점에 IBM MQ classes for JMS는 다시 시도하기 전에 일정 기간을 대기함) 클라이언트 채널 정의 테이블에서 각각의 적당한 항목을 차례로 시도합니다.

CONNECTIONNAMELIST 또는 CCDTURL을 사용하든지 간에, 이 자동 클라이언트 다시 연결 프로세스는 IBM MQ classes for JMS에서 QMANAGER 특성에 지정된 큐 관리자에 성공적으로 다시 연결될 때까지 계속됩니다.

기본적으로, 재연결 시도는 다음 간격으로 발생합니다.

- 첫 번째 시도는 1초의 초기 지연과 최대 250밀리초의 랜덤 요소 이후에 이루어집니다.
- 두 번째 시도는 첫 번째 시도가 실패한 이후 2초와 최대 500밀리초의 랜덤 간격에 이루어집니다.
- 세 번째 시도는 두 번째 시도가 실패한 이후 4초와 최대 1초의 랜덤 간격에 이루어집니다.
- 네 번째 시도는 세 번째 시도가 실패한 이후 8초와 최대 2초의 랜덤 간격에 이루어집니다.
- 다섯 번째 시도는 네 번째 시도가 실패한 이후 16초와 최대 4초의 랜덤 간격에 이루어집니다.
- 여섯 번째 시도와 모든 후속 시도는 이전 시도가 실패한 이후 25초와 최대 6초 250밀리초의 랜덤 간격에 이루어집니다.

재연결 시도는 부분적으로 고정되고 부분적으로 랜덤인 간격만큼 지연됩니다. 이는 더 이상 사용 가능하지 않은 큐 관리자에 연결된 모든 IBM MQ classes for JMS 애플리케이션의 동시 재연결을 방지하기 위한 것입니다.

큐 관리자가 복구하거나 대기 큐 관리자가 활성화되는 데 필요한 시간의 양을 보다 정확히 반영하기 위해 기본값을 늘려야 하는 경우, 클라이언트 구성 파일의 채널 스탠자의 ReconDelay 속성을 수정하십시오. 자세한 정보는 클라이언트 구성 파일의 CHANNELS 스탠자를 참조하십시오.

자동으로 다시 연결된 이후 IBM MQ classes for JMS 애플리케이션이 계속해서 올바르게 작동하는지 여부는 해당 디자인에 따라 다릅니다. 자동 재연결 기능을 사용할 수 있는 애플리케이션을 디자인하는 방법을 이해하려면 관련 주제를 읽으십시오.

큐 관리자를 더 이상 사용할 수 없음을 나타내는 이유 코드

자동 IBM MQ classes for JMS 재연결을 시도할 때 이유 코드에서 큐 관리자를 더 이상 사용할 수 없거나 큐 관리자에 도달할 수 없음을 나타냅니다.

264 페이지의 『자동 JMS 클라이언트 재연결』에서는 `JMSEException`에 대한 개요와 애플리케이션을 자동으로 재시작하는 방법을 제공하며, 264 페이지의 『자동 JMS 클라이언트 재연결 사용』의 정보에서는 자동 클라이언트 재연결을 위해 자세히 설명합니다.

다음 정보에서는 애플리케이션에서 확인해야 하는 IBM MQ 이유 코드를 나열합니다.

RC2009

`MQRC_CONNECTION_BROKEN`

RC2059

`MQRC_Q_MGR_NOT_AVAILABLE`

RC2161

`MQRC_Q_MGR_QUIESCING`

RC2162

`MQRC_Q_MGR_STOPPING`

RC2202

`MQRC_CONNECTION_QUIESCING`

RC2203

`MQRC_CONNECTION_STOPPING`

RC2223

`MQRC_Q_MGR_NOT_ACTIVE`

RC2279

`MQRC_CHANNEL_STOPPED_BY_USER`

RC2537

`MQRC_CHANNEL_NOT_AVAILABLE`

RC2538

`MQRC_HOST_NOT_AVAILABLE`

엔터프라이즈 애플리케이션에서 다시 발생하는 대부분의 `JMSEException`은 이유 코드를 포함하는 링크된 `MQException`을 포함합니다. 이전 목록에서 이유 코드에 대한 재시도 논리를 구현하려면 엔터프라이즈 애플리케이션은 다음 예제와 비슷한 코드를 사용하여 이 링크된 예외를 확인해야 합니다.

```
} catch (JMSEException ex) {
    Exception linkedEx = ex.getLinkedException();
    if (ex.getLinkedException() != null) {
        if (linkedEx instanceof MQException) {
            MQException mqException = (MQException) linkedEx;
            int reasonCode = mqException.reasonCode;
            // Handle the reason code accordingly
        }
    }
}
```

관련 개념

JMS용 IBM MQ 클래스

Java SE 및 Java EE 환경에서 자동 클라이언트 다시 연결 사용

IBM MQ 클라이언트 재연결을 사용하여 Java SE 및 Java EE 환경 내에서 다양한 고가용성(HA) 및 재해 복구(DR)를 용이하게 할 수 있습니다.

다양한 HA 및 DR 솔루션이 여러 플랫폼에 제공됩니다.

- ▶ **Multi** 다중 인스턴스 큐 관리자는 다양한 서버에서 구성되는 동일한 큐 관리자의 인스턴스입니다(다중 인스턴스 큐 관리자 참조). 큐 관리자의 한 인스턴스는 활성 인스턴스로 정의되고 다른 인스턴스는 대기 인스턴스로 정의됩니다. 활성 인스턴스가 실패하면 다중 인스턴스 큐 관리자가 대기 서버에서 자동으로 시작됩니다.

활성 및 대기 큐 관리자 모두 동일한 큐 관리자 식별자(QMID)를 가집니다. IBM MQ 다중 인스턴스 큐 관리자에 연결하는 클라이언트 애플리케이션은 자동 클라이언트 재연결 기능을 사용하여 큐 관리자의 대기 인스턴스에 자동으로 재연결되도록 구성할 수 있습니다.

- ▶ **Linux** 복제 데이터 큐 관리자(RDQM)는 Linux 플랫폼에서 사용 가능한 고가용성 솔루션입니다(RDQM 고가용성 참조). RDQM 구성은 각각 큐 관리자의 인스턴스가 있는 고가용성(HA) 그룹에서 구성된 세 서버로 구성됩니다. 하나의 인스턴스는 다른 두 인스턴스에 해당 데이터를 동시에 복제하는, 실행 중인 큐 관리자입니다. 이 큐 관리자를 실행하는 서버가 실패하면 큐 관리자의 다른 인스턴스가 시작되고 현재 데이터를 사용하여 작동합니다. 큐 관리자의 세 인스턴스는 부동 IP 주소를 공유하므로 클라이언트는 단일 IP 주소로만 구성해야 합니다. RDQM 큐 관리자에 연결하는 클라이언트 애플리케이션은 자동 클라이언트 다시 연결을 사용하여 큐 관리자의 대기 인스턴스에 자동으로 재연결하도록 구성될 수 있습니다.
- ▶ **MQ Appliance** HA 솔루션은 한 쌍의 IBM MQ 어플라이언스로도 제공될 수 있습니다(문서의 IBM MQ Appliance '고가용성 및 재해 복구' 참조). HA 큐 관리자는 어플라이언스 중 하나에서 실행되면서, 다른 어플라이언스에 있는 큐 관리자의 대기 인스턴스로 데이터를 동시에 복제합니다. 1차 어플라이언스가 실패할 경우 큐 관리자가 다른 어플라이언스에서 자동으로 시작되어 실행됩니다. 큐 관리자의 두 인스턴스는 부동 IP 주소를 공유하도록 구성할 수 있으므로 클라이언트는 단일 IP 주소로만 구성해야 합니다. IBM MQ Appliance의 HA 큐 관리자에 연결되는 클라이언트 애플리케이션은 자동 클라이언트 다시 연결을 사용하여 큐 관리자의 대기 인스턴스에 자동으로 다시 연결되도록 구성할 수 있습니다.

참고: Java EE 환경(예: WebSphere Application Server)에서는, IBM MQ classes for JMS에서 제공하는 기능을 사용하는 활성화 스펙을 통한 자동 클라이언트 재연결이 지원되지 않습니다. 활성화 스펙이 연결된 큐 관리자가 사용 불가능하게 되는 경우, IBM MQ 자원 어댑터는 활성화 스펙을 재연결하기 위한 자체 메커니즘을 제공합니다. 자세한 정보는 269 페이지의 『Java EE 환경에서 자동 클라이언트 다시 연결에 대한 지원』의 내용을 참조하십시오.

관련 개념

[다중 인스턴스 큐 관리자](#)

[자동 클라이언트 다시 연결](#)

관련 참조

[RDQM 고가용성](#)

Java SE 환경에서 자동 클라이언트 다시 연결 사용

실행 Java SE IBM MQ classes for JMS 중인 환경에서 애플리케이션은 연결 팩토리 속성을 통해 자동 클라이언트 재연결 **CLIENTRECONNECTOPTIONS** 기능을 사용할 수 있습니다.

연결 팩토리 특성 **CLIENTRECONNECTOPTIONS**는 두 개의 추가 연결 팩토리 특성인 **CONNECTIONNAMELIST** 및 **CCDTURL**을 사용하여 큐 관리자가 실행되는 서버에 연결하는 방법을 판별합니다.

CONNECTIONNAMELIST 특성

CONNECTIONNAMELIST 특성은 클라이언트 모드에서 큐 관리자에 연결하는 데 사용할 호스트 이름 및 포트 정보를 포함하는 심표로 구분된 목록입니다. 이 특성은 **QMANAGER** 및 **CHANNEL** 값과 함께 사용됩니다. 애플리케이션이 **CONNECTIONNAMELIST** 특성을 사용하여 클라이언트 연결을 작성할 때 IBM MQ classes for JMS는 목록 순서에 있는 각 호스트에 연결을 시도합니다. 첫 번째 큐 관리자 호스트를 사용할 수 없는 경우 IBM MQ classes for JMS가 목록에 있는 다음 호스트에 연결을 시도합니다. 연결을 작성하지 않고 연결 이름 목록의 끝에 도달한 경우 IBM MQ classes for JMS는 MQRC_QMGR_NOT_AVAILABLE IBM MQ 이유 코드를 발생시킵니다.

애플리케이션이 연결된 큐 관리자가 실패하면 해당 큐 관리자에 연결하기 위해 **CONNECTIONNAMELIST**를 사용한 모든 애플리케이션은 큐 관리자를 사용할 수 없다는 예외를 수신합니다. 애플리케이션은 예외를 발견하고 사용하고 있는 자원을 제거해야 합니다. 연결을 작성하려면 애플리케이션이 연결 팩토리를 사용해야 합니다. 연결 팩토리는 목록 순서의 각 호스트에 연결을 다시 시도하고 실패한 큐 관리자를 이제 사용할 수 없습니다. 연결 팩토리는 목록의 다른 호스트에 연결을 시도합니다.

CCDTURL 특성

CCDTURL 특성은 CCDT(Client Channel Definition Table)를 지시하는 URL(Uniform Resource Locator)를 포함하며 이 특성은 **QMANAGER** 특성과 함께 사용됩니다. CCDT는 IBM MQ 시스템에 정의된 큐 관리자에 연결하는 데 사용하는 클라이언트 채널의 목록을 포함합니다. IBM MQ classes for JMS에서 CCDT를 사용하는 방법에 대한

정보는 262 페이지의 『IBM MQ classes for JMS에서 클라이언트 채널 정의 테이블 사용』의 내용을 참조하십시오.

속성을 CLIENTRECONNECTOPTIONS 사용하여 클라이언트의 자동 재연결을 활성화합니다.

IBM MQ classes for JMS

CLIENTRECONNECTOPTIONS 특성이 IBM MQ classes for JMS 내에서 자동 클라이언트 다시 연결을 가능하게 하기 위해 사용됩니다. 이 특성의 가능한 값은 다음과 같습니다.

ASDEF

자동 클라이언트 재연결 작동은 IBM MQ 클라이언트 구성 파일(mqclient.ini)의 채널 스탠자에 지정된 기본값으로 정의됩니다.

사용 안함

자동 클라이언트 다시 연결을 사용할 수 없습니다.

QMGR

IBM MQ classes for JMS는 다음 옵션 중 하나를 사용하여 자신이 연결된 큐 관리자와 동일한 큐 관리자 ID를 갖는 큐 관리자에 연결을 시도합니다.

- **CHANNEL** 특성에 정의된 채널 및 **CONNECTIONNAMELIST** 특성
- **CCDTURL** 특성에 정의된 CCDT

모두

IBM MQ classes for JMS는 **CONNECTIONNAMELIST** 특성 또는 **CCDTURL** 중 하나를 사용하여 동일한 이름의 큐 관리자에 다시 연결을 시도합니다.

관련 정보

[클라이언트 구성 파일의 CHANNELS 스탠자](#)

[JMS용 IBM MQ 클래스 오브젝트의 특성](#)

Java EE 환경에서 자동 클라이언트 다시 연결 사용

Java EE (Java Platform, Enterprise Edition) 환경에 배치할 수 있는 IBM MQ 자원 어댑터 및 WebSphere Application Server IBM MQ 메시징 제공자는 IBM MQ classes for JMS를 사용하여 IBM MQ 큐 관리자와 통신합니다. IBM MQ 자원 어댑터 및 WebSphere Application Server IBM MQ 메시징 제공자는 활성화 스펙, WebSphere Application Server 리스너 포트 및 클라이언트 컨테이너 내에서 실행하는 애플리케이션이 큐 관리자에 자동으로 다시 연결할 수 있도록 하는 여러 메커니즘을 제공합니다. EJB(Enterprise JavaBean) 및 웹 기반 애플리케이션은 자체 재연결 로직을 구현해야 합니다.

참고: IBM MQ classes for JMS에 제공된 기능을 사용하는 활성화 스펙을 통한 자동 클라이언트 재연결은 지원되지 않습니다(264 페이지의 『자동 JMS 클라이언트 재연결』 참조). 활성화 스펙이 연결된 큐 관리자가 사용 불가능하게 되는 경우, IBM MQ 자원 어댑터는 활성화 스펙을 재연결하기 위한 자체 메커니즘을 제공합니다.

자원 어댑터가 제공하는 메커니즘은 다음을 통해 제어됩니다.

- IBM MQ 자원 어댑터 특성 **reconnectionRetryCount**.
- IBM MQ 자원 어댑터 특성 **reconnectionRetryInterval**.
- 활성화 스펙 특성 **connectionNameList**.

이러한 특성에 대한 자세한 정보는 407 페이지의 『ResourceAdapter 오브젝트 특성의 구성』의 내용을 참조하십시오.

메시지 구동 Bean 애플리케이션의 onMessage() 메소드 내에서 자동 클라이언트 다시 연결 사용 또는 Java Platform, Enterprise Edition 환경 내에서 실행 중인 기타 애플리케이션은 지원되지 않습니다. 연결 중인 큐 관리자가 사용 불가능하게 되는 경우, 애플리케이션은 자체 재연결 로직을 구현해야 합니다. 자세한 정보는 276 페이지의 『Java EE 애플리케이션에서 재연결 논리 구현』의 내용을 참조하십시오.

Java EE 환경에서 자동 클라이언트 다시 연결에 대한 지원

Java EE 환경(예: WebSphere Application Server)에서, IBM MQ 자원 어댑터 및 WebSphere Application Server IBM MQ 메시징 제공자는 애플리케이션이 큐 관리자에 자동으로 다시 연결할 수 있도록 하는 여러 메커니즘을 제공합니다. 그러나 일부 경우에는 이러한 지원이 제한적으로 제공됩니다.

Java EE 환경 및 WebSphere Application Server IBM MQ 메시징 제공자에 배치할 수 있는 IBM MQ 자원 어댑터는 IBM MQ classes for JMS 를 사용하여 IBM MQ 큐 관리자와 통신합니다.

다음 표에는 IBM MQ 자원 어댑터와 WebSphere Application Server IBM MQ 메시징 제공자가 자동 클라이언트 다시 연결을 지원하는 내용이 요약되어 있습니다.

표 45. Java EE 환경에서 자동 클라이언트 다시 연결에 대한 지원 요약				
자동 다시 연결에 대한 옵션	CONNECTIONNAMELIST 특성	CCDTURL 특성	CLIENTRECONNECTOPTIONS 특성	자동 클라이언트 다시 연결에 대한 대체 접근법
활성화 스펙	제한적으로 지원됨	제한적으로 지원됨	지원되지 않음	Java EE 환경과 활성화 스펙은 해당 자체 다시 연결 메커니즘을 제공함
WebSphere Application Server 리스너 포트	제한적으로 지원됨	제한적으로 지원됨	지원되지 않음	WebSphere Application Server 는 자체 재연결 메커니즘을 제공함
엔터프라이즈 Java Bean 및 웹 기반 애플리케이션	제한적으로 지원됨	제한적으로 지원됨	지원되지 않음	애플리케이션이 자체 재연결 논리를 구현해야 함
클라이언트 컨테이너 내에서 실행 중인 애플리케이션	지원됨	지원됨	지원됨	적용할 수 없음

Java EE 환경에 설치된 MDB(Message-driven bean) 애플리케이션(예: IBM MQ classes for JMS)은 IBM MQ 시스템에서 메시지를 처리하기 위해 활성화 스펙을 사용할 수 있습니다. 활성화 스펙은 IBM MQ 시스템에 도착한 메시지를 감지하여 처리를 위해 이를 MDB로 전달하는 데 사용됩니다. 메시지 구동 Bean은 **onMessage()** 메소드 내부에서 IBM MQ 시스템에 더 많은 연결을 작성할 수도 있습니다. 이러한 연결이 자동 클라이언트 다시 연결을 사용하는 방법에 대한 자세한 내용은 [Enterprise JavaBeans 및 웹 기반 애플리케이션](#)을 참조하십시오.

활성화 스펙

활성화 스펙의 경우 **CONNECTIONNAMELIST** 및 **CCDTURL** 특성이 제한적으로 지원되고 **CLIENTRECONNECTOPTIONS** 특성이 지원되지 않습니다.

Java EE 환경에 설치된 MDB(Message-driven bean) 애플리케이션(예: WebSphere Application Server)은 IBM MQ 시스템에서 메시지를 처리하기 위해 활성화 스펙을 사용할 수 있습니다.

활성화 스펙은 IBM MQ 시스템에 메시지 도착을 감지하여 처리를 위해 이를 MDB로 전달하는 데 사용됩니다. 이 절에서는 활성화 스펙이 IBM MQ 시스템을 모니터링하는 방법을 다룹니다.

MDB는 **onMessage()** 메소드 내부에서 IBM MQ 시스템에 추가로 연결할 수도 있습니다.

이러한 연결이 자동 클라이언트 다시 연결을 사용하는 방법에 대한 자세한 내용은 [273 페이지의 『엔터프라이즈 Java Bean 및 웹 기반 애플리케이션』](#)에 있습니다.

CONNECTIONNAMELIST 특성

시작할 때, 활성화 스펙은 다음을 사용하여 큐 관리자에 대한 연결을 시도합니다.

- **QMANAGER** 특성에 지정된 항목
- **CHANNEL** 특성에 지정된 채널
- **CONNECTIONNAMELIST**에서 첫 번째 항목의 호스트 이름 및 포트 정보

활성화 스펙이 목록의 첫 번째 항목을 사용하여 큐 관리자에 연결할 수 없는 경우 활성화 스펙은 두 번째 항목으로 이동합니다. 큐 관리자에 대한 연결이 작성될 때까지 또는 테이블의 끝에 도달할 때까지 계속해서 이동합니다.

활성화 스펙이 **CONNECTIONNAMELIST**의 모든 입력 항목을 사용해도 지정된 큐 관리자에 연결할 수 없으면 활성화 스펙이 중지되고 재시작되어야 합니다.

활성화 스펙이 실행되면 활성화 스펙이 IBM MQ 시스템에서 메시지를 가져오고 이 메시지를 처리를 위해 MDB에 전달합니다.

메시지가 처리되는 동안 큐 관리자가 실패하면 Java EE 환경이 실패를 감지하고 활성화 스펙에 다시 연결을 시도합니다.

활성화 스펙은 다시 연결을 시도할 때 이전과 마찬가지로 **CONNECTIONNAMELIST** 특성에 있는 정보를 사용합니다.

활성화 스펙이 **CONNECTIONNAMELIST**의 모든 항목을 시도했지만 여전히 큐 관리자에 연결할 수 없으면 다시 시도하기 전에 IBM MQ 자원 어댑터 특성 **reconnectionRetryInterval**에서 지정한 시간만큼 대기합니다.

IBM MQ 자원 어댑터 특성 **reconnectionRetryCount**는 활성화 스펙이 중지되고 수동 재시작을 요구하기 전에 허용되는 연속 재연결 시도 횟수를 정의합니다.

일단 활성화 스펙이 IBM MQ 시스템에 다시 연결되면 Java EE 환경은 필요한 트랜잭션 정리를 수행하고 처리를 위해 메시지를 MDB로 전달하는 것을 재개합니다.

트랜잭션 정리가 올바르게 작동하려면 Java EE 환경이 실패한 큐 관리자에 대한 로그에 액세스할 수 있어야 합니다.

XA 트랜잭션에 참여하는 트랜잭션 MDB와 함께 활성화 스펙을 사용 중이고 다중 인스턴스 큐 관리자에 연결 중인 경우, **CONNECTIONNAMELIST**에는 활성 및 대기 큐 관리자 인스턴스 둘 다에 대한 항목을 포함해야 합니다.

이는 실패 후에 환경이 다시 연결하는 큐 관리자에 관계없이, 환경이 트랜잭션 복구를 수행해야 하는 경우 Java EE 환경이 큐 관리자 로그에 액세스할 수 있음을 의미합니다.

트랜잭션 MDB가 독립형 큐 관리자와 함께 사용되는 경우, **CONNECTIONNAMELIST** 특성은 단일 항목을 포함해야 활성화 스펙이 항상 실패 후에 동일한 시스템에서 실행 중인 동일한 큐 관리자에 다시 연결할 수 있습니다.

CCDTURL 특성

시작할 때 활성화 스펙은 클라이언트 채널 정의 테이블 (CCDT)의 첫 번째 항목을 사용하여 **QMANAGER** 특성에 지정된 큐 관리자에 연결하려고 시도합니다.

활성화 스펙이 테이블에서 첫 번째 항목을 사용하여 큐 관리자에 연결할 수 없는 경우 활성화 스펙은 두 번째 항목으로 이동합니다. 큐 관리자에 대한 연결이 작성될 때까지 또는 테이블의 끝에 도달할 때까지 계속해서 이동합니다.

활성화 스펙이 CCDT의 모든 입력 항목을 사용해도 지정된 큐 관리자에 연결할 수 없으면 활성화 스펙이 중지되고 재시작되어야 합니다.

활성화 스펙이 실행되면 활성화 스펙이 IBM MQ 시스템에서 메시지를 가져오고 이 메시지를 처리를 위해 MDB에 전달합니다.

메시지가 처리되는 동안 큐 관리자가 실패하면 Java EE 환경이 실패를 감지하고 활성화 스펙에 다시 연결을 시도합니다.

활성화 스펙은 다시 연결을 시도할 때 이전과 마찬가지로 CCDT 특성에 있는 정보를 사용합니다.

활성화 스펙이 CCDT의 모든 항목을 시도했지만 여전히 큐 관리자에 연결할 수 없으면 다시 시도하기 전에 IBM MQ 자원 어댑터 특성 **reconnectionRetryInterval**에서 지정한 시간만큼 대기합니다.

IBM MQ 자원 어댑터 특성 **reconnectionRetryCount**는 활성화 스펙이 중지되고 수동 재시작을 요구하기 전에 허용되는 연속 재연결 시도 횟수를 정의합니다.

일단 활성화 스펙이 IBM MQ 시스템에 다시 연결되면 Java EE 환경은 필요한 트랜잭션 정리를 수행하고 처리를 위해 메시지를 MDB로 전달하는 것을 재개합니다.

트랜잭션 정리가 올바르게 작동하려면 Java EE 환경이 실패한 큐 관리자에 대한 로그에 액세스할 수 있어야 합니다.

XA 트랜잭션에 참여하는 트랜잭션 MDB와 함께 활성화 스펙을 사용 중이고 멀티 인스턴스 큐 관리자에 연결 중인 경우, CCDT는 활성 및 대기 큐 관리자 인스턴스 둘 다에 대한 항목을 포함해야 합니다.

이는 실패 후에 환경이 다시 연결하는 큐 관리자에 관계없이, 환경이 트랜잭션 복구를 수행해야 하는 경우 Java EE 환경이 큐 관리자 로그에 액세스할 수 있음을 의미합니다.

트랜잭션 MDB가 독립형 큐 관리자와 사용 중이면 활성화 스펙이 오류 발생 후에 동일한 시스템에서 실행 중인 동일한 큐 관리자로 항상 다시 연결되도록 CCDT에 단일 항목을 포함해야 합니다.

동일한 활성화 큐 관리자에 연결이 작성되도록 활성화 스펙과 함께 사용된 CCDT의 **AFFINITY** 특성에 기본값 **PREFERRED**를 설정했는지 확인하십시오.

CLIENTRECONNECTOPTIONS 특성

활성화 스펙은 자체 재연결 기능을 제공합니다. 제공된 기능을 통해 활성화 스펙이 연결된 큐 관리자가 실패할 경우 활성화 스펙이 IBM MQ 시스템에 자동으로 다시 연결할 수 있습니다.

이 때문에 IBM MQ classes for JMS에서 제공하는 자동 클라이언트 다시 연결 기능은 지원되지 않습니다.

Java EE에서 사용되는 모든 활성화 스펙에 대해 **CLIENTRECONNECTOPTIONS** 특성을 **DISABLED** 로 설정해야 합니다.

WebSphere Application Server 리스너 포트

WebSphere Application Server에 설치된 MDB(Message-driven bean) 애플리케이션은 IBM MQ 시스템에서 메시지를 처리하기 위해 리스너 포트를 사용할 수 있습니다.

리스너 포트는 IBM MQ 시스템에 메시지 도착을 감지하여 처리를 위해 이를 MDB로 전달하는 데 사용됩니다. 이 주제에서는 리스너 포트가 IBM MQ 시스템을 모니터링하는 방법을 설명합니다.

MDB는 onMessage() 메소드 내부에서 IBM MQ 시스템에 추가로 연결할 수도 있습니다.

이러한 연결에서 자동 클라이언트 다시 연결을 사용하는 방법에 대한 자세한 정보는 [273 페이지의 『엔터프라이즈 Java Bean 및 웹 기반 애플리케이션』](#)의 내용을 참조하십시오.

WebSphere Application Server 리스너 포트의 경우:

- **CONNECTIONNAMELIST**와 **CCDTURL**은 제한적으로 지원됩니다.
- **CLIENTRECONNECTOPTIONS**는 지원되지 않습니다.

CONNECTIONNAMELIST 특성

리스너 포트는 IBM MQ에 연결할 때 JMS 연결 풀을 사용하므로 연결 풀 사용의 영향을 받습니다. 자세한 정보는 [270 페이지의 『활성화 스펙』](#)의 내용을 참조하십시오.

사용 가능한 연결이 없고 이 연결 팩토리에서 아직 최대 연결 수가 작성되지 않은 경우 **CONNECTIONNAMELIST** 특성을 사용하여 IBM MQ에 대한 새 연결을 시도하고 작성합니다.

CONNECTIONNAMELIST의 모든 IBM MQ 시스템에 액세스할 수 없는 경우 리스너 포트가 중지됩니다.

그런 다음 리스너 포트는 메시지 리스너 서비스 사용자 정의 특성 **RECOVERY.RETRY.INTERVAL**에서 지정한 시간 동안 대기한 후 다시 연결을 시도합니다.

연결 시도 사이에 리턴된 연결이 있는 경우에 한하여 이 재연결은 연결 풀에 여유 연결이 있는지 확인을 시도합니다. 사용 가능한 연결이 여유 연결이 없으면 리스너 포트는 그 전처럼 **CONNECTIONNAMELIST**를 사용합니다.

일단 리스너 포트가 IBM MQ 시스템에 다시 연결되면 Java EE 환경은 필요한 트랜잭션 정리를 수행하고 처리를 위해 메시지를 MDB로 전달하는 것을 재개합니다.

트랜잭션 정리가 올바르게 작동하려면 Java EE 환경이 실패한 큐 관리자에 대한 로그에 액세스할 수 있어야 합니다.

XA 트랜잭션에 참여하는 트랜잭션 MDB와 함께 리스너 포트를 사용 중이고 **다중 인스턴스 큐 관리자**에 연결 중인 경우, **CONNECTIONNAMELIST**에는 활성 및 대기 큐 관리자 인스턴스 둘 다에 대한 항목을 포함해야 합니다.

이는 실패 후에 환경이 다시 연결하는 큐 관리자에 관계없이, 환경이 트랜잭션 복구를 수행해야 하는 경우 Java EE 환경이 큐 관리자 로그에 액세스할 수 있음을 의미합니다.

트랜잭션 MDB가 독립형 큐 관리자와 함께 사용되는 경우, **CONNECTIONNAMELIST** 특성은 단일 항목을 포함해야 활성화 스펙이 항상 실패 후에 동일한 시스템에서 실행 중인 동일한 큐 관리자에 다시 연결할 수 있습니다.

CCDTURL 특성

리스너 포트는 시작할 때 CCDT의 첫 번째 항목을 사용하여 **QMANAGER** 특성에 지정된 큐 관리자에 연결을 시도합니다.

리스너 포트가 테이블에서 첫 번째 항목을 사용하여 큐 관리자에 연결할 수 없는 경우 리스너 포트는 두 번째 항목으로 이동합니다. 큐 관리자에 대한 연결이 작성될 때까지 또는 테이블의 끝에 도달할 때까지 계속해서 이동합니다.

리스너 포트가 CCDT의 모든 항목을 사용해도 지정된 큐 관리자에 연결할 수 없으면 리스너 포트가 중지됩니다.

그런 다음 리스너 포트는 메시지 리스너 서비스 사용자 정의 특성 **RECOVERY.RETRY.INTERVAL**에서 지정한 시간 동안 대기한 후 다시 연결을 시도합니다.

이 재연결 시도는 이전처럼 CCDT의 모든 항목에 대해서 작동합니다.

일단 리스너 포트가 실행 중이면 IBM MQ 시스템에서 메시지를 얻고 처리를 위해 이를 MDB에 전달합니다.

메시지를 처리하는 동안 큐 관리자가 실패하면 Java EE 환경이 실패를 감지하고 리스너 포트에 재연결을 시도합니다. 리스너 포트는 재연결 시도를 수행할 때 CCDT의 정보를 사용합니다.

리스너 포트가 CCDT의 모든 항목을 시도했지만 여전히 큐 관리자에 연결할 수 없는 경우 포트는 다시 시도하기 전에 **RECOVERY.RETRY.INTERVAL** 특성에서 지정한 시간 동안 대기합니다.

메시지 리스너 포트 서비스 특성 **MAX.RECOVERY.RETRIES**는 리스너 포트가 중지되고 수동 재시작을 요구하기 전에 허용되는 연속 재연결 시도 횟수를 정의합니다.

일단 리스너 포트가 IBM MQ 시스템에 다시 연결되면 Java EE 환경은 필요한 트랜잭션 정리를 수행하고 처리를 위해 메시지를 MDB로 전달하는 것을 재개합니다.

트랜잭션 정리가 올바르게 작동하려면 Java EE 환경이 실패한 큐 관리자에 대한 로그에 액세스할 수 있어야 합니다.

XA 트랜잭션에 참여하는 트랜잭션 MDB와 함께 리스너 포트를 사용 중이고 멀티 인스턴스 큐 관리자에 연결 중인 경우, CCDT는 활성 및 대기 큐 관리자 인스턴스 둘 다에 대한 항목을 포함해야 합니다.

이는 실패 후에 환경이 다시 연결하는 큐 관리자에 관계없이, 환경이 트랜잭션 복구를 수행해야 하는 경우 Java EE 환경이 큐 관리자 로그에 액세스할 수 있음을 의미합니다.

트랜잭션 MDB가 독립형 큐 관리자와 사용 중이면 리스너 포트가 오류 발생 후에 동일한 시스템에서 실행 중인 동일한 큐 관리자로 항상 다시 연결되도록 CCDT에 단일 항목을 포함해야 합니다.

동일한 활성 큐 관리자에 연결이 작성되도록 리스너 포트와 함께 사용된 CCDT의 **AFFINITY** 특성에 기본값 **PREFERRED**를 설정했는지 확인하십시오.

CLIENTRECONNECTOPTIONS 특성

리스너 포트는 자체 재연결 기능을 제공합니다. 제공된 기능을 통해 리스너 포트가 연결된 큐 관리자가 실패할 경우 리스너 포트가 IBM MQ 시스템에 자동으로 다시 연결할 수 있습니다.

이 때문에 IBM MQ classes for JMS에서 제공하는 자동 클라이언트 다시 연결 기능은 지원되지 않습니다.

Java EE에서 사용되는 모든 리스너 포트에 대해 **CLIENTRECONNECTOPTIONS** 특성을 **DISABLED** 로 설정해야 합니다.

엔터프라이즈 Java Bean 및 웹 기반 애플리케이션

엔터프라이즈 자바 애플리케이션(JavaBean, EJB) 애플리케이션과 서블릿(Servlet)과 같이 웹 컨테이너 내에서 실행되는 애플리케이션은 연결 팩토리 JMS (connection factory)를 사용하여 큐 매니저(IBM MQ queue manager)에 대한 연결을 생성합니다.

다음 제한사항은 EJB 및 웹 기반 애플리케이션에 적용됩니다.

- **CONNECTIONNAMELIST**와 **CCDTURL**은 제한적으로 지원됩니다.

- **CLIENTRECONNECTOPTIONS**는 지원되지 않습니다.

CONNECTIONNAMELIST 특성

환경이 Java EE JMS 연결을 위한 연결 풀을 제공하는 경우, 이가 속성의 **CONNECTIONNAMELIST** 동작에 미치는 영향에 대한 정보는 [를 275 페이지의 『연결 풀에서 CONNECTIONNAMELIST 또는 CCDT 사용』](#) 참조하십시오.

Java EE 환경이 JMS 연결 풀을 제공하지 않으면 해당 애플리케이션은 속성을 **CONNECTIONNAMELIST** Java SE 애플리케이션과 동일한 방식으로 사용합니다.

XA 트랜잭션에 참여하는 트랜잭션 MDB와 함께 애플리케이션을 사용 중이고 다중 인스턴스 큐 관리자에 연결 중인 경우, **CONNECTIONNAMELIST**에는 활성 및 대기 큐 관리자 인스턴스 둘 다에 대한 항목을 포함해야 합니다.

이는 실패 후에 환경이 다시 연결하는 큐 관리자에 관계없이, 환경이 트랜잭션 복구를 수행해야 하는 경우 Java EE 환경이 큐 관리자 로그에 액세스할 수 있음을 의미합니다.

애플리케이션이 독립형 큐 관리자와 함께 사용되는 경우, 해당 **CONNECTIONNAMELIST** 속성은 단일 항목만 포함해야 합니다. 이는 장애 발생 후 애플리케이션이 항상 동일한 시스템에서 실행 중인 동일한 큐 관리자에 재연결 되도록 보장하기 위함입니다.

CCDTURL 특성

환경이 Java EE 연결을 JMS 위한 연결 풀을 제공하는 경우, 이가 속성의 **CCDTURL** 동작에 미치는 영향에 대한 정보는 [을 275 페이지의 『연결 풀에서 CONNECTIONNAMELIST 또는 CCDT 사용』](#) 참조하십시오.

Java EE 환경이 JMS 연결 풀을 제공하지 않으면 해당 애플리케이션은 속성을 **CCDTURL** Java SE 애플리케이션과 동일한 방식으로 사용합니다.

XA 트랜잭션에 참여하는 트랜잭션 MDB와 함께 애플리케이션을 사용 중이고 다중 인스턴스 큐 관리자에 연결 중인 경우, CCDT에는 활성 및 대기 큐 관리자 인스턴스 둘 다에 대한 항목을 포함해야 합니다.

이는 실패 후에 환경이 다시 연결하는 큐 관리자에 관계없이, 환경이 트랜잭션 복구를 수행해야 하는 경우 Java EE 환경이 큐 관리자 로그에 액세스할 수 있음을 의미합니다.

애플리케이션이 독립형 큐 관리자와 함께 사용 중이면 활성화 스펙이 오류 발생 후에 동일한 시스템에서 실행 중인 동일한 큐 관리자로 항상 다시 연결되도록 CCDT에 단일 항목을 포함해야 합니다.

CLIENTRECONNECTOPTIONS 특성

웹 컨테이너에서 실행되는 EJB 또는 애플리케이션이 사용하는 모든 JMS 연결 팩토리의 속성을 **CLIENTRECONNECTOPTIONS DISABLED** 로 설정해야 합니다.

사용 중인 큐 관리자가 실패하는 경우 자동으로 새 큐 관리자에 재연결해야 하는 애플리케이션은 자체 재연결 논리를 구현해야 합니다. 자세한 정보는 [276 페이지의 『Java EE 애플리케이션에서 재연결 논리 구현』](#)의 내용을 참조하십시오.

시나리오: WebSphere Application Server with IBM MQ

시나리오: WebSphere Application Server 리버티 프로파일과 IBM MQ

클라이언트 컨테이너 내에서 실행 중인 애플리케이션

일부 Java EE 환경 (예: WebSphere Application Server) 은 Java SE 애플리케이션을 실행하는 데 사용할 수 있는 클라이언트 컨테이너를 제공합니다.

이러한 환경 내에서 실행되는 애플리케이션은 JMS 연결 팩토리를 사용하여 IBM MQ 큐 관리자에 연결합니다.

클라이언트 컨테이너 내에서 실행 중인 애플리케이션의 경우:

- **CONNECTIONNAMELIST** 및 **CCDTURL**이 완전히 지원됩니다.
- **CLIENTRECONNECTOPTIONS**가 완전히 지원됩니다.

CONNECTIONNAMELIST 특성

환경이 Java EE JMS 연결을 위한 연결 풀을 제공하는 경우, 이가 속성의 **CONNECTIONNAMELIST** 동작에 미치는 영향에 대한 정보는 [플 275 페이지의 『연결 풀에서 CONNECTIONNAMELIST 또는 CCDT 사용』](#) 참조하십시오.

Java EE 환경이 JMS 연결 풀을 제공하지 않으면 해당 애플리케이션은 속성을 **CONNECTIONNAMELIST** Java SE 애플리케이션과 동일한 방식으로 사용합니다.

CCDTURL 특성

환경이 Java EE 연결을 JMS 위한 연결 풀을 제공하는 경우, 이가 속성의 **CCDTURL** 동작에 미치는 영향에 대한 정보는 [플 275 페이지의 『연결 풀에서 CONNECTIONNAMELIST 또는 CCDT 사용』](#) 참조하십시오.

Java EE 환경이 JMS 연결 풀을 제공하지 않으면 해당 애플리케이션은 속성을 **CCDTURL** Java SE 애플리케이션과 동일한 방식으로 사용합니다.

연결 풀에서 **CONNECTIONNAMELIST** 또는 **CCDT** 사용

일부 Java EE 환경(예: WebSphere Application Server)은 JMS 연결 풀 컨테이너를 제공하며, 이 컨테이너는 Java SE 애플리케이션을 실행하는 데 사용될 수 있습니다.

Java EE 환경에 정의된 연결 팩토리를 사용하여 연결을 작성하는 애플리케이션은 이 연결 팩토리에 대해 연결 풀에서 기존 여유 연결을 확보하거나, 연결 풀에 적합한 연결이 없으면 새 연결을 확보합니다.

이는 연결 팩토리가 **CONNECTIONNAMELIST** 또는 **CCDTURL** 특성이 정의되어 구성된 경우 영향을 미칠 수 있습니다.

연결 팩토리가 연결을 작성하기 위해 처음 사용되면 Java EE 환경이 **CONNECTIONNAMELIST** 또는 **CCDTURL**을 사용하여 IBM MQ 시스템에 대한 새 연결을 작성합니다. 이 연결이 더 이상 필요하지 않을 때 연결이 재사용될 수 있는 연결 풀로 돌아갑니다.

다른 항목이 연결 팩토리에서 연결을 작성하면 Java EE 환경은 새 연결을 작성하기 위해 **CONNECTIONNAMELIST** 또는 **CCDTURL** 특성을 사용하지 않고 연결 풀의 연결을 리턴합니다.

큐 관리자 인스턴스가 실패할 때 연결이 사용 중이면 연결이 제거됩니다. 하지만 연결 풀의 콘텐츠는 제거되지 않을 수도 있습니다. 이는 풀이 더 이상 실행되지 않는 큐 관리자에 대한 연결을 잠재적으로 포함할 수 있음을 의미합니다.

이 경우 다음 번에 연결 팩토리에서 연결을 작성하도록 요청되면 실패한 큐 관리자에 대한 연결이 리턴됩니다. 이 연결 사용을 위한 시도는 큐 관리자가 더 이상 실행 중이 아니므로 연결이 제거되어 실패합니다.

연결 풀이 비어 있는 경우에만 'Java EE 환경은 ' **CONNECTIONNAMELIST** ' 또는 ' **CCDTURL** ' 속성을 사용하여 'IBM MQ'에 대한 새 연결을 생성합니다.

CONNECTIONNAMELIST 및 **CCDT**가 JMS 연결을 작성하는 데 사용된 방법 때문에, 다양한 IBM MQ 시스템에 대한 연결을 포함하는 연결 풀을 보유하는 것도 가능합니다.

예를 들어 **CONNECTIONNAMELIST** 특성을 다음 값으로 설정하여 연결 팩토리를 구성했다고 가정합니다.

```
CONNECTIONNAMELIST = hostname1(port1), hostname2(port2)
```

애플리케이션이 이 연결 팩토리에서 독립형 큐 관리자로 연결을 작성하는 것을 처음 시도하고 시스템 `hostname1(port1)`에서 실행 중인 큐 관리자에 액세스할 수 없다고 가정합니다. 이는 애플리케이션이 `hostname2(port2)`에서 실행 중인 큐 관리자에 대한 연결을 종료함을 의미합니다.

또 다른 애플리케이션이 지금 따라 들어와서 동일한 연결 팩토리에서 JMS 연결을 작성합니다. 이제 ' `hostname1(port1)` '의 대기열 관리자를 사용할 수 있으므로 이 'IBM MQ' 시스템에 대한 새로운 'JMS' 연결이 생성되어 애플리케이션으로 반환됩니다.

두 애플리케이션이 완료되면 해당 JMS 연결을 닫고, 연결이 연결 풀로 리턴됩니다.

결과적으로 연결 팩토리에 대한 연결 풀에는 두 개의 JMS 연결이 있게 됩니다.

- `hostname1(port1)`에서 실행 중인 큐 관리자에 대한 하나의 연결

- hostname2(port2)에서 실행 중인 큐 관리자에 대한 하나의 연결

이는 트랜잭션 복구와 관련하여 문제를 일으킬 수 있습니다. Java EE 시스템에서 트랜잭션을 롤백해야 하는 경우 트랜잭션 로그에 대한 액세스 권한이 있는 큐 관리자에 연결할 수 있어야 합니다.

Java EE 애플리케이션에서 재연결 논리 구현

큐 관리자가 자체 재연결 논리 구현에 실패하는 경우 자동으로 다시 연결하려는 Enterprise JavaBeans 및 웹 기반 애플리케이션입니다.

다음 옵션에서는 이를 수행할 수 있는 방법에 대한 자세한 정보를 제공합니다.

애플리케이션의 실패를 허용

이 접근법은 애플리케이션 변경이 없어야 하지만 **CONNECTIONNAMELIST** 특성을 포함하도록 연결 팩토리 정의의 관리 재구성이 필요합니다. 그러나 이 접근법은 호출자가 실패를 적절하게 처리할 수 있어야 합니다. 참고로, 이는 연결 실패에 관련되지 않은 MQRC_Q_FULL과 같은 실패에 대해서도 필요합니다.

이 프로세스에 대한 예제 코드:

```
public class SimpleServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        try {
            // get connection factory/ queue
            InitialContext ic = new InitialContext();
            ConnectionFactory cf =
                (ConnectionFactory)ic.lookup("java:comp/env/jms/WMQCF");
            Queue q = (Queue) ic.lookup("java:comp/env/jms/WMQQueue");

            // send a message
            Connection c = cf.createConnection();
            Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
            MessageProducer p = s.createProducer(q);
            Message m = s.createTextMessage();
            p.send(m);

            // done, release the connection
            c.close();
        }
        catch (JMSEException je) {
            // process exception
        }
    }
}
```

앞의 코드는 이 서블릿이 사용 중인 연결 팩토리에 **CONNECTIONNAMELIST** 특성이 정의되어 있다고 가정합니다.

서블릿이 처음 처리할 때, 동일한 큐 관리자에 연결하는 다른 애플리케이션에서 폴링된 연결을 사용할 수 없다고 가정하여 **CONNECTIONNAMELIST** 속성을 사용하여 새 연결이 생성됩니다.

close() 호출 뒤에 연결이 해제될 때 이 연결은 폴로 돌아가고 다음 번에 서블릿이 실행될 때 다시 사용됩니다. 이는 **CONNECTIONNAMELIST**에 대한 참조 없이, CONNECTION_ERROR_OCCURRED 이벤트가 생성된 시점에서 연결 실패가 발생할 때까지 지속됩니다. 이 이벤트는 폴이 실패한 연결을 영구 삭제하도록 프롬프트합니다.

다음 번에 애플리케이션이 실행될 때 폴링된 연결을 사용할 수 없으며 처음으로 사용 가능한 큐 관리자에 연결하기 위해 **CONNECTIONNAMELIST**가 사용됩니다. 큐 관리자 장애 복구(예: 장애가 일시적 네트워크 오류가 아닌 경우)가 발생하면 서블릿이 백업 인스턴스에 연결합니다(사용 가능하면).

데이터베이스와 같은 기타 자원이 애플리케이션에 관련된 경우 애플리케이션 서버가 트랜잭션을 롤백해야 함을 표시하는 것이 적절할 수 있습니다.

애플리케이션 내의 다시 연결 처리

호출자가 서블릿에서 실패를 처리할 수 없으면 재연결을 애플리케이션 내에서 처리해야 합니다. 다음 예제에 표시된 대로 애플리케이션 내에서 다시 연결을 처리하려면 애플리케이션이 새 연결을 요청하여 JNDI에서 검색한 연결 팩토리를 캐시하고 JMSEException(예: JMSCMQ0001:WebSphere MQ call failed with

comPCODE '2' ('MQCC_FAILED') reason '2009' ('MQRC_CONNECTION_BROKEN'))을 처리할 수 있어야 합니다.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // get connection factory/ queue
    InitialContext ic = new InitialContext();
    ConnectionFactory cf = (ConnectionFactory)
        ic.lookup("java:comp/env/jms/WMQCF");
    Destination destination = (Destination) ic.lookup("java:comp/env/jms/WMQQueue");

    setupResources();

    // loop sending messages
    while (!sendComplete) {
        try {
            // create the next message to send
            msg.setText("message sent at "+new Date());
            // and send it
            producer.send(msg);
        }
        catch (JMSException je) {
            // drive reconnection
            setupResources();
        }
    }
}
```

다음 예제에서 `setupResources()`는 JMS 오브젝트를 작성하고 비즉각적 재연결을 처리하기 위해 휴면 및 재 시도 루프를 포함합니다. 실제로, 이 메소드는 여러 번의 재연결 시도를 방지합니다. 참고로, 쉽게 구별할 수 있도록 엑시트 조건을 예제에 표시하지 않았습니다.

```
private void setupResources() {

    boolean connected = false;
    while (!connected) {
        try {
            connection = cf.createConnection(); // cf cached from JNDI lookup
            session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
            msg = session.createTextMessage();
            producer = session.createProducer(destination); // destination cached from JNDI lookup
            // no exception? then we connected ok
            connected = true;
        }
        catch (JMSException je) {
            // sleep and then have another attempt
            try {Thread.sleep(30*1000);} catch (InterruptedException ie) {}
        }
    }
}
```

애플리케이션이 다시 연결을 관리할 때 이러한 자원이 기타 IBM MQ 큐 관리자 또는 기타 백엔드 서비스(예: 데이터베이스)인지 여부에 따라 애플리케이션이 기타 자원에 대해 보유한 연결을 해제합니다. 새 IBM MQ 큐 관리자 인스턴스에 다시 연결이 완료되면 이러한 연결을 다시 설정해야 합니다. 연결을 다시 설정하지 않으면 애플리케이션 서버 자원이 재연결 시도 동안 불필요하게 보유되고 다시 사용되는 시점에 제한시간 초과될 수도 있습니다.

WorkManager의 사용

처리 시간이 몇 십 초보다 큰 장시간 실행 애플리케이션(예: 배치 처리)에 대해 WebSphere Application Server WorkManager를 사용할 수 있습니다. WebSphere Application Server에 대한 코드 단편 예제는 다음과 같습니다.

```
public class BatchSenderServlet extends HttpServlet {

    private WorkManager workManager = null;
    private MessageSender sender; // background sender WorkImpl

    public void init() throws ServletException {
        InitialContext ctx = new InitialContext();
        workManager = (WorkManager)ctx.lookup("java:comp/env/wm/default");
        sender = new MessageSender(5000);
        workManager.startWork(sender);
    }

    public void destroy() {
```

```

        sender.halt();
    }

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        if (sender.isRunning()) {
            out.println(sender.getStatus());
        }
    }
}

```

여기서, web.xml에는 다음이 포함되어 있습니다.

```

<resource-ref>
    <description>WorkManager</description>
    <res-ref-name>wm/default</res-ref-name>
    <res-type>com.ibm.websphere.asynchbeans.WorkManager</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>

```

이제, 배치(batch)가 작업 인터페이스를 통해 구현됩니다.

```

import com.ibm.websphere.asynchbeans.Work;

public class MessageSender implements Work {

    public MessageSender(int messages) {numberOfMessages = messages;}

    public void run() {
        // get connection factory/ queue
        InitialContext ic = new InitialContext();
        ConnectionFactory cf = (ConnectionFactory)
            ic.lookup("java:comp/env/jms/WMQCF");
        Destination destination = (Destination) ic.lookup("jms/WMQQueue");

        setupResources();

        // loop sending messages
        while (!sendComplete) {
            try {
                // create the next message to send
                msg.setText("message sent at "+new Date());
                // and send it
                producer.send(msg);
                // are we finished?
                if (sendCount == numberOfMessages) {sendComplete = true;}
            }
            catch (JMSEException je) {
                // drive reconnection
                setupResources();
            }
        }

        public boolean isRunning() {return !sendComplete;}

        public void release() {sendComplete = true;}
    }
}

```

배치 처리를 실행하는 데 오래 걸리는 경우(예: 대형 메시지, 느린 네트워크, 또는 광범위한 데이터베이스 액세스 등, 특히 느린 장애 복구가 결합되었을 때) 서버는 다음 예제와 유사하게 스레드 정지 경고를 출력하기 시작합니다.

WSVR0605W: 스레드 "WorkManager.DefaultWorkManager: 0" (00000035)이 694061밀리초 동안 활성 상태였으며 정지될 수도 있습니다. 서버에 있는 정지될 수 있는 총 스레드 수는 1개입니다.

이러한 경고는 배치(batch) 크기를 줄이거나 정지 스레드 제한시간을 늘려서 피할 수 있습니다. 하지만 EJB에서 이러한 처리를 구현하거나(일괄 송신의 경우), MDB(이용 또는 이용 및 응답에 대해) 처리를 구현하는 경우에는 일반적으로 선호됩니다.

참고로, 애플리케이션 관리 재연결은 런타임 오류를 핸들하기 위한 일반적 해결책을 제공하지 않으며 애플리케이션은 연결 오류와 관련되지 않은 오류를 계속해서 핸들해야 합니다.

예를 들어 가득 찬 큐에 메시지를 넣으려고 하거나(2053 MQRC_Q_FULL), 올바르지 않은 보안 신임 정보를 사용하여 큐 관리자에 연결을 시도하는 경우(2035 MQRC_NOT_AUTHORIZED)가 있습니다.

애플리케이션은 장애 복구가 진행 중일 때 즉시 사용 가능한 인스턴스가 없으면 2059 MQRC_Q_MGR_NOT_AVAILABLE 오류도 핸들해야 합니다. 이는 재연결을 자동으로 시도하는 대신 JMS 예외가 발생하면 애플리케이션이 예외를 보고하여 처리될 수 있습니다.

IBM MQ classes for JMS 오브젝트 풀링

Java EE 외부에서 연결 풀의 양식을 사용하면 전체적인 최종 로드를 줄이는 데 도움이 됩니다(예: 프레임워크를 사용하거나 클라우드 환경에 배치되는 일부 독립형 애플리케이션에서, 또한 애플리케이션과 큐 관리자의 서버 통합 증가로 이어지는 QueueManagers로의 다수의 클라이언트 연결에서).

Java EE 프로그래밍 모델 내에는 사용 중인 다양한 오브젝트에 대해 잘 정의된 라이프 사이클이 있습니다. 메시지 구동 Bean(MDB)이 가장 제약적인 반면 서블릿은 제한이 적습니다. 그러므로 Java EE 서버에서 사용 가능한 풀링 옵션은 사용된 다양한 프로그래밍 모델에 적합합니다.

Java SE를 사용하면(또는 Spring 등의 다른 프레임워크를 사용하면) 프로그래밍 모델이 매우 탄력적입니다. 그러므로 단일 풀링 전략이 모두 적합하지는 않습니다. 풀링 양식을 수행할 수 있는 위치의 프레임워크가 되는지 고려해야 합니다(예: Spring).

사용할 풀링 전략은 애플리케이션이 실행 중인 환경에 따라 다릅니다.

Java EE 환경에서 오브젝트 풀링

Java EE 애플리케이션 서버는 메시지 주도 빈 애플리케이션, 엔터프라이즈 Java 빈 및 서블릿이 사용할 수 있는 연결 풀링 기능을 제공합니다.

WebSphere Application Server는 성능을 개선하기 위해 JMS 제공자에 대한 연결 풀을 유지보수합니다. 애플리케이션이 JMS 연결을 작성하면 애플리케이션 서버는 연결이 여유 연결 풀에 이미 있는지를 판별합니다. 있는 경우 연결이 애플리케이션에 리턴됩니다. 그렇지 않으면 새 연결이 작성됩니다.

279 페이지의 그림 41에서는 활성화 스펙과 리스너 포트가 JMS 연결을 구축하는 방법과 정상 모드에서 메시지의 목적지를 모니터하기 위해 해당 연결을 사용하는 방법을 보여줍니다.

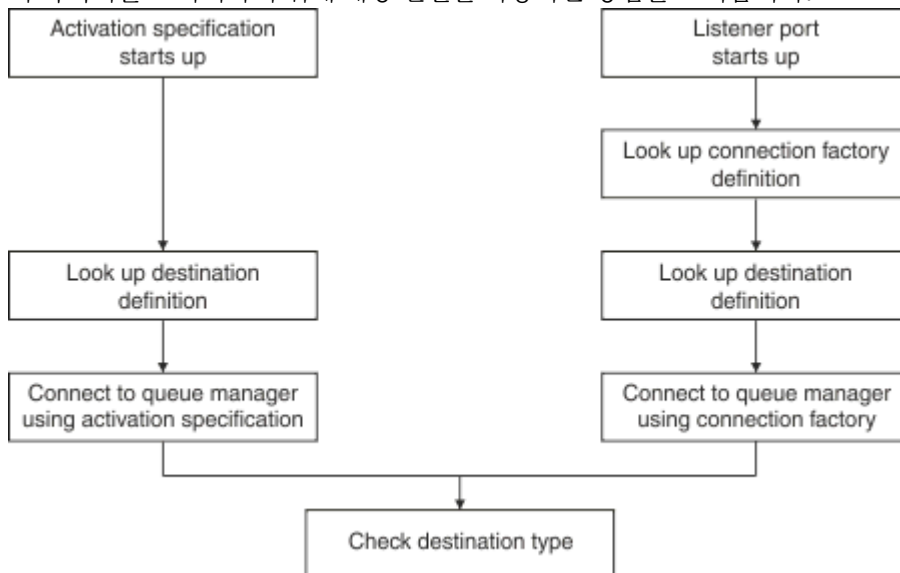


그림 41. 정상 모드

메시징 IBM MQ 공급자를 사용할 때, 아웃바운드 메시징을 수행하는 애플리케이션(예: 엔터프라이즈 Java 빈 및 서블릿)과 메시지 주도 빈 리스너 포트 구성 요소는 이러한 연결 풀을 활용할 수 있습니다.

IBM MQ메시징 제공자 활성화 스펙은 IBM MQ 자원 어댑터에서 제공하는 연결 풀링 기능을 사용합니다. 자세한 정보는 WebSphere MQ 자원 어댑터에 대한 특성 구성을 참조하십시오.

283 페이지의 『연결 풀 사용 예제』에서는 아웃바운드 메시징을 수행하는 애플리케이션과 리스너 포트가 JMS 연결을 작성할 때 여유 풀을 사용하는 방법을 설명합니다.

285 페이지의 『여유 연결 풀 유지보수 스레드』에서는 애플리케이션 또는 리스너 포트가 연결을 종료할 때 이러한 연결에 발생하는 사항을 설명합니다.

287 페이지의 『풀 유지보수 스레드 예제』에서는 JMS 연결이 시간 경과(stale)되는 것을 막기 위해 여유 연결 풀을 정리하는 방법을 설명합니다.

WebSphere Application Server는 연결 팩토리의 팩토리에서 작성될 수 있는 연결 수에 제한이 있으며 이러한 제한은 연결 팩토리의 *maximum connections* 특성에서 지정됩니다. 이 특성의 기본값은 10이며 한 번에 하나의 연결 팩토리에서 최대 10개의 연결을 작성할 수 있음을 의미합니다.

각 팩토리는 연관된 여유 연결 풀이 있습니다. 애플리케이션 서버가 시작되면 여유 연결 풀이 비어 있습니다. 팩토리에 대한 여유 풀에 존재할 수 있는 최대 연결 수는 최대 연결 수 특성에서도 지정됩니다.

팁: JMS 2.0에서 연결 팩토리를 사용하여 연결과 컨텍스트를 모두 작성할 수 있습니다. 결과적으로 연결과 컨텍스트가 둘 다 혼합되어 있는 연결 팩토리와 연결 풀이 연관될 수 있습니다. 연결 팩토리는 연결을 작성하거나 컨텍스트를 작성하는 데만 사용하는 것이 좋습니다. 그러면 해당 연결 팩토리의 연결 풀에 한 가지 유형의 오브젝트만 포함되므로 풀의 효율성이 향상됩니다.

WebSphere Application Server에서 연결 풀링이 작동하는 방법에 대한 정보는 [JMS 연결을 위한 연결 풀링 구성](#)을 참조하십시오. 다른 애플리케이션 서버의 경우 적절한 애플리케이션 서버 문서를 참조하십시오.

연결 풀이 사용되는 방법

각 JMS 연결 팩토리는 연관된 연결 풀이 있으며 연결 풀에는 0개 이상의 JMS 연결이 포함됩니다. 각 JMS 연결에는 연관된 JMS 세션 풀이 있으며 각 JMS 세션 풀은 0개 이상의 JMS 세션을 포함합니다.

280 페이지의 그림 42에서는 이러한 오브젝트 사이의 관계를 보여줍니다.

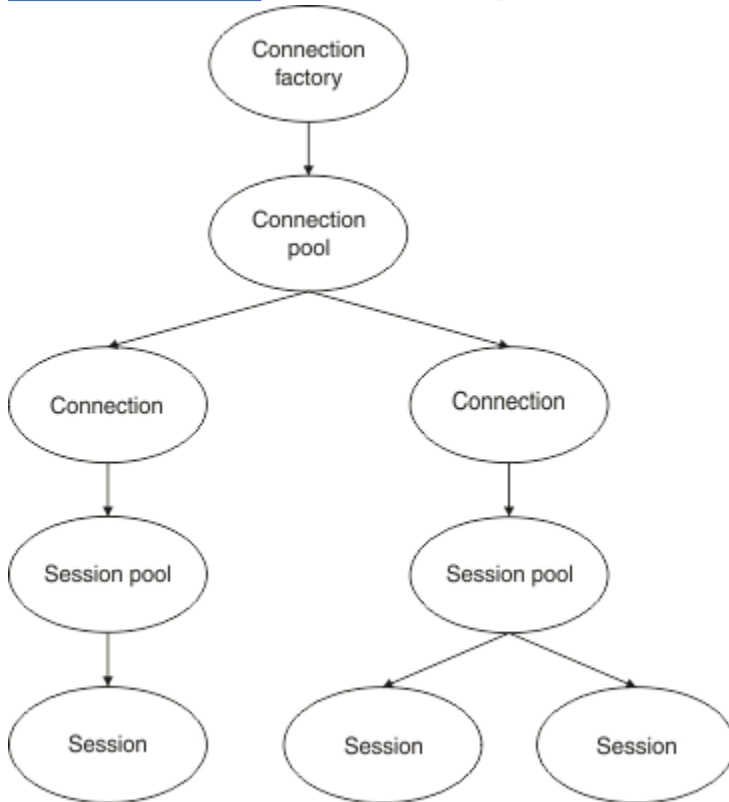


그림 42. 연결 풀 및 세션 풀

리스너 포트가 시작되거나 아웃바운드 메시징을 수행하려는 애플리케이션이 연결 팩토리를 작성하기 위해 팩토리를 사용하는 경우 포트 또는 애플리케이션이 다음 메소드 중 하나를 호출합니다.

- `connectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String, String)`
- `QueueConnectionFactory.createQueueConnection()`
- `QueueConnectionFactory.createQueueConnection(String, String)`

- **TopicConnectionFactory.createTopicConnection()**
- **TopicConnectionFactory.createTopicConnection(String, String)**

WebSphere Application Server 연결 관리자는 이 팩토리에 대한 여유 풀에서 연결을 확보하여 애플리케이션으로 리턴하려고 시도합니다.

풀에 여유 연결이 없고 이 팩토리에서 작성된 연결 수가 해당 팩토리의 *maximum connections* 특성에 지정된 한계에 도달하지 않은 경우 연결 관리자는 애플리케이션이 사용할 새 연결을 작성합니다.

그러나 애플리케이션이 연결 작성을 시도하지만 이 팩토리에서 작성된 연결 수가 이미 팩토리의 *maximum connections* 특성과 동일한 경우 애플리케이션은 연결이 사용 가능해질 때까지(여유 풀에 돌아올 때까지) 기다립니다.

애플리케이션이 대기하는 시간은 연결 풀의 *connection timeout* 특성에 지정되고 기본값은 180초입니다. 연결이 180초 내에 여유 풀에 돌아오면 연결 관리자가 즉시 풀에서 가져와서 애플리케이션에 전달합니다. 하지만 제한시간이 초과하면 *ConnectionWaitTimeoutException*이 발생합니다.

애플리케이션이 연결을 종료하면 다음을 호출하여 닫습니다.

- **Connection.close()**
- **QueueConnection.close()**
- **TopicConnection.close()**

연결은 계속해서 열려 있으며 여유 풀에 리턴되어 다른 애플리케이션이 다시 사용할 수 있게 됩니다. 따라서 애플리케이션 서버에서 애플리케이션이 실행 JMS 중이지 않더라도 와 WebSphere Application Server JMS 공급자 사이에 연결을 열어 둘 수 있습니다.

고급 연결 풀 특성

JMS 연결 풀의 작동을 제어하는 데 사용할 수 있는 여러 가지 고급 특성이 있습니다.

서지(surge) 보호

284 페이지의 『아웃바운드 메시징을 수행하는 애플리케이션이 연결 풀을 사용하는 방법』에서는 `connectionFactory.createConnection()`을 통합한 `sendMessage()` 메소드의 사용을 설명합니다.

`ejbCreate()` 메소드의 일부로 동일한 연결 팩토리에서 모두 JMS 연결을 작성 중인 50개의 EJB가 있는 상황을 고려해 봅시다.

이러한 Bean 모두가 동시에 작성되고 팩토리의 여유 연결 풀에 연결이 없으면 애플리케이션 서버는 동일한 JMS 제공자에 대해 동시에 50개의 JMS 연결을 작성하려고 시도합니다. 그 결과 WebSphere Application Server 및 JMS 제공자 모두에 심각한 로드가 발생합니다.

서지(surge) 보호 특성은 한 번에 연결 팩토리에서 작성할 수 있는 JMS 연결 수를 제한하고 추가 연결 작성에 시차를 두어 이러한 상황을 방지합니다.

다음 두 특성을 사용하여 한 번에 가능한 JMS 연결 수를 제한할 수 있습니다.

- 서지(Surge) 임계값
- 서지(Surge) 작성 간격

EJB 애플리케이션이 연결 팩토리에서 JMS 연결을 작성하려고 시도하면 연결 관리자가 얼마나 많은 연결이 작성되고 있는지 확인하는 검사를 수행합니다. 해당 숫자가 속성의 `surge threshold` 값보다 작거나 같으면 연결 관리자는 새 연결을 계속 열어갑니다.

그러나 생성되는 연결 수가 속성을 `surge threshold` 초과할 경우, 연결 관리자는 속성으로 `surge creation interval` 지정된 기간 동안 대기한 후 새 연결을 생성하고 엽니다.

스턱(Stuck) 연결

연결은 JMS 애플리케이션이 JMS 해당 연결을 사용하여 제공자에게 요청을 JMS 전송한 후, 제공자가 일정 시간 내에 응답하지 않을 경우 비활성 `stuck`상태로 간주됩니다.

WebSphere Application Server는 stuck JMS 연결을 감지할 수 있는 방법을 제공합니다. 이 기능을 사용하려면 세 개의 특성을 설정해야 합니다.

- 스택(Stuck) 시간 타이머
- 스택(Stuck) 시간
- 스택(stuck) 임계값

287 페이지의 『풀 유지보수 스레드 예제』에서는 풀 유지보수 스레드를 정기적으로 실행하고, 한동안 사용되지 않거나 너무 오래 존재해 온 연결이 있는지를 찾으면서 연결 팩토리의 여유 풀 콘텐츠를 검사하는 방법을 설명합니다.

스택(stuck) 연결을 감지하기 위해 애플리케이션 서버는 연결 팩토리에서 작성된 모든 활성 연결의 상태를 검사하는 스택(stuck) 연결 스레드를 관리하여 JMS 제공자로부터 응답을 대기 중인 연결이 있는지 확인합니다.

스택 연결 스레드가 실행되는 시점은 Stuck time timer 특성에 의해 결정됩니다. 이 특성의 기본값은 0이며 이는 스택(stuck) 연결 감지가 실행되지 않음을 의미합니다.

스레드가 응답을 대기 중인 연결을 발견하면 얼마나 오랫동안 대기했는지를 판별하고 Stuck time 특성의 값을 이 시간과 비교합니다.

JMS 제공자가 응답에 걸리는 시간이 Stuck time 특성에서 지정된 시간을 초과하는 경우, 애플리케이션 서버는 JMS 연결을 스택(stuck)으로 표시합니다.

예를 들어, 연결 팩토리의 속성이 Stuck time timer 10으로 설정되고, jms/CF1 속성이 Stuck time 15로 설정되어 있다고 가정해 보자.

스택(stuck) 연결 스레드는 10초마다 활성화되고 jms/CF1에서 작성된 연결 중에 IBM MQ의 응답을 15초 넘게 기다리고 있는 연결이 있는지를 검사합니다.

EJB가 를 사용하여 IBM MQ 에 대한 연결을 JMS 생성한 후, 해당 연결을 사용하여 jms/CF1를 호출하여 세션을 JMS 생성하려고 Connection.createSession() 시도한다고 가정합니다.

그러나 어떤 이유로 JMS 제공자가 요청에 응답하지 않고 있습니다. 시스템이 보류(freeze)되었거나 JMS 제공자에서 실행되는 프로세스가 교착 상태여서 새로운 작업이 처리되는 것을 방해할 수 있습니다.

EJB가 Connection.createSession()을 호출한 10초 후에 스택(stuck) 연결 타이머가 활성화되고 jms/CF1에서 작성된 활성 연결을 확인합니다.

예를 들어 호출된 c1과 같이 하나의 활성 연결이 있다고 가정합니다. 첫 번째 EJB는 c1으로 전송한 요청에 대한 응답을 10초 동안 기다립니다. 이 시간은 Stuck time의 값보다 적으므로 스택 연결 타이머가 이 연결을 무시하고 비활성으로 됩니다.

10초 후에 스택 연결 스레드는 다시 활성화되고 jms/CF1에 대한 활성화 연결을 확인합니다. 이전과 마찬가지로, c1 하나의 연결만 있다고 가정합니다.

처음 EJB가 createSession()을 호출한 후 20초가 되었으며 EJB가 여전히 응답을 대기 중입니다. 20초는 속성에 Stuck time 지정된 시간보다 길기 때문에, 연결 스레드가 해당 연결을 막힌 상태로 c1 표시합니다.

5초 후에 IBM MQ가 마침내 응답하면 첫 번째 EJB가 JMS 세션을 작성하고 연결이 다시 사용 중이 됩니다.

애플리케이션 서버는 스택인 연결 팩토리에서 작성된 JMS 연결 수를 계산합니다. 새 JMS 연결을 작성하기 위해 애플리케이션에서 해당 연결 팩토리를 사용하고 팩토리의 여유 풀에 여유 연결이 없으면 연결 관리자는 스택 연결 수를 Stuck threshold 특성 값과 비교합니다.

고착된 연결 수가 'stuck_connections' Stuck threshold 속성에 설정된 값보다 적을 경우, 연결 관리자는 새 연결을 생성하여 애플리케이션에 제공합니다.

그러나 막힌 연결 수가 속성의 Stuck threshold 값과 같을 경우, 애플리케이션은 리소스 예외를 받게 됩니다.

풀 파티션

WebSphere Application Server는 연결 팩토리에 대한 여유 풀 연결을 파티션할 수 있는 두 개의 특성을 제공합니다.

- Number of free pool partitions는 사용자가 여유 연결 풀을 나누려는 파티션 수를 애플리케이션 서버에 알려줍니다.
- Free pool distribution table size는 파티션이 색인화되는 방법을 결정합니다.

지원팀에서 IBM 변경을 요청하지 않는 한, 이 속성들은 기본값인 0으로 유지하십시오.

WebSphere Application Server에는 Number of shared partitions라는 하나의 추가 고급 연결 풀 특성이 있음을 참고하십시오. 이 특성은 공유 연결을 저장하는 데 사용하는 파티션 수를 지정합니다. 그러나 JMS 연결이 항상 공유되지 않으므로 이 특성은 적용되지 않습니다.

연결 풀 사용 예제

아웃바운드 메시지를 수행하는 애플리케이션과 MDB 리스너 포트 구성요소는 JMS 연결 풀을 사용합니다.

283 페이지의 그림 43에서는 WebSphere Application Server V7.5 및 V8.0에 대해 연결 풀이 작동하는 방법을 보여줍니다.

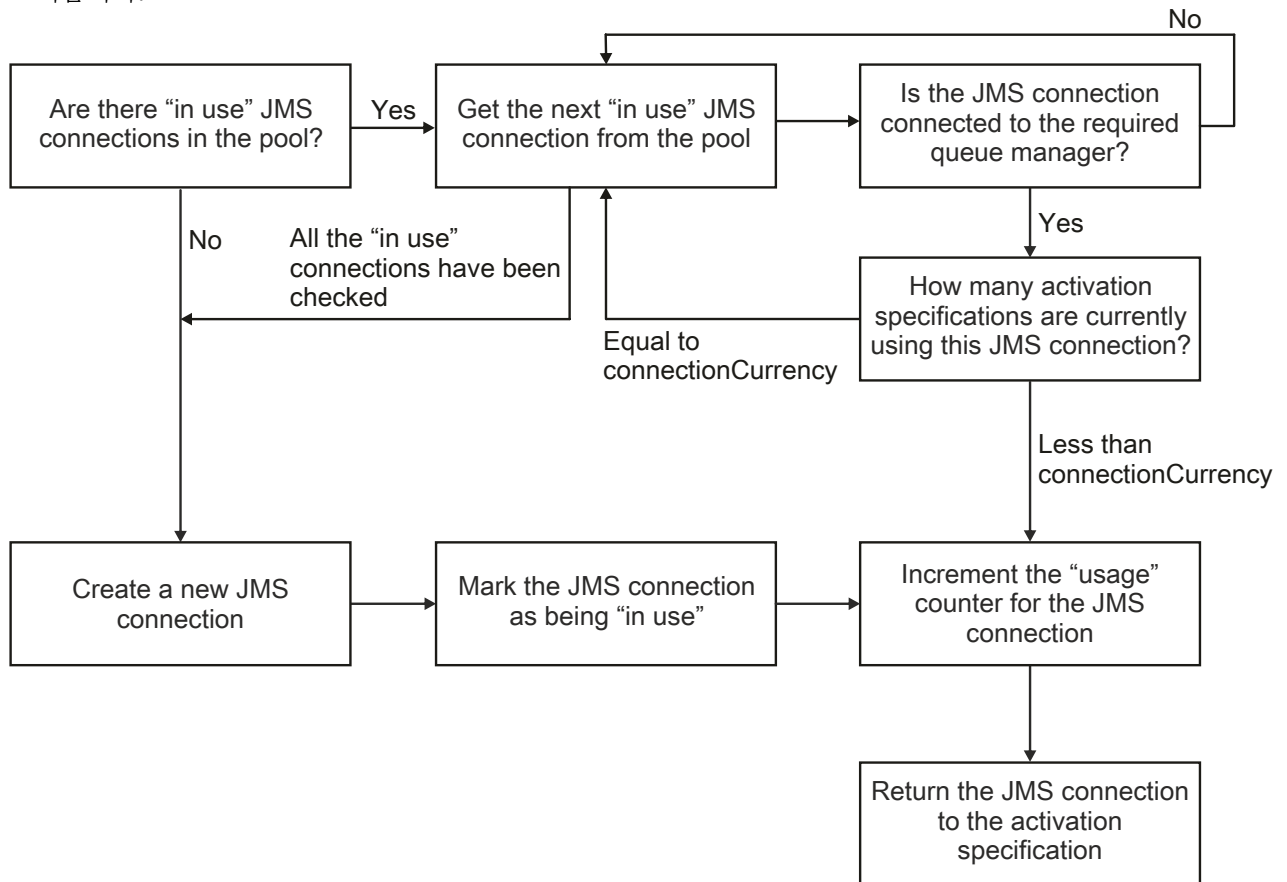


그림 43. WebSphere Application Server V7.5 및 V8.0 - 연결 풀이 작동하는 방법

284 페이지의 그림 44에서는 WebSphere Application Server V8.5에 대해 연결 풀이 작동하는 방법을 보여줍니다.

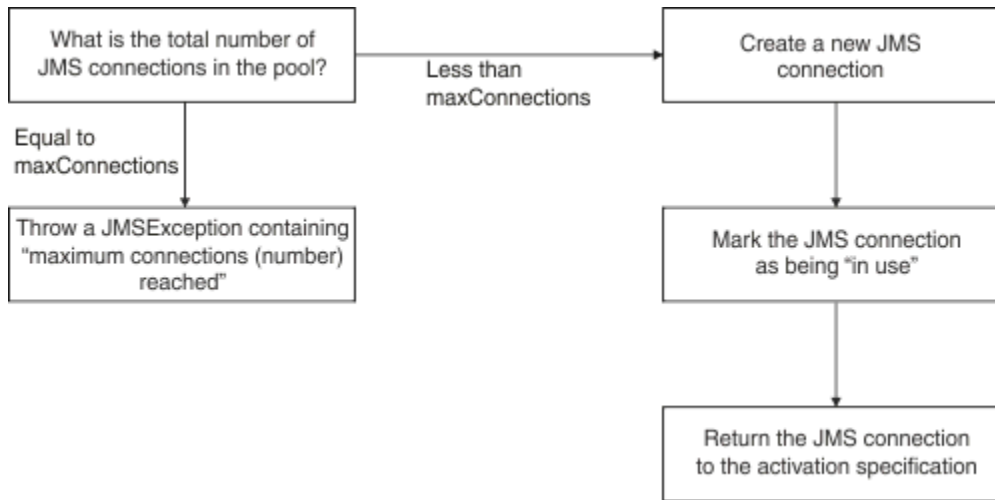


그림 44. WebSphere Application Server V8.5 - 연결 풀이 작동하는 방법

MDB 리스너 포트가 연결 풀을 사용하는 방법

네트워크 배포 WebSphere Application Server 시스템에 MDB가 배포되어 있고, 해당 MDB가 를 JMS 제공자로 IBM MQ 사용하고 있다고 가정합니다. *maximum connections* 특성을 2로 설정한 *.jms/CF1*이라는 연결 팩토리를 사용 중인 리스너 포트에 대해 MDB를 배치했으며 이는 한 번에 이 팩토리에서 두 개의 연결만 작성할 수 있음을 의미합니다.

리스너 포트가 시작될 때, 해당 포트는 연결 *.jms/CF1* 팩토리를 사용하여 에 IBM MQ 대한 연결을 생성하려고 시도합니다.

이를 수행하기 위해 포트가 연결 관리자에게 연결을 요청합니다. 이번이 *.jms/CF1* 연결 팩토리가 처음 사용되는 경우이므로 *.jms/CF1* 여유 연결 풀에는 연결이 없습니다. 따라서 연결 관리자가 예를 들어 *c1*이라는 새로운 연결을 작성합니다. 이 연결이 리스너 포트의 전체 주기 동안 존재함을 참고하십시오.

이제 WebSphere Application Server 관리 콘솔을 사용하여 사용자가 리스너 포트를 중지하는 상황을 고려해 보겠습니다. 이 경우 연결 관리자는 연결을 받아서 이를 여유 풀에 다시 넣습니다. 그러나 IBM MQ에 대한 연결은 여전히 열려 있습니다.

사용자가 리스너 포트를 다시 시작하면 포트는 다시 큐 관리자에 대한 연결을 연결 관리자에게 요청합니다. 여유 풀에 연결(*c1*)이 있으므로 연결 관리자가 풀에서 이 연결을 가져와서 리스너 포트가 사용할 수 있게 합니다.

이제, 두 번째 MDB가 애플리케이션 서버에 배치되어 있고 서로 다른 리스너 포트를 사용한다고 가정합니다.

그런 다음 역시 *.jms/CF1* 연결 팩토리를 사용하도록 구성된 세 번째 리스너 포트를 시작하려고 시도합니다. 세 번째 리스너 포트는 연결 관리자에게 연결을 요청하며 연결 관리자는 여유 풀에서 *.jms/CF1*을 찾아 비어 있음을 발견합니다. 그런 다음 *.jms/CF1* 팩토리에서 이미 작성된 연결 수를 확인합니다.

*.jms/CF1*에 대한 최대 연결 수 특성이 2로 설정되어 있고 이미 이 팩토리에서 2개의 연결을 작성했으므로 연결 팩토리는 연결이 사용 가능해질 때까지 180초(연결 제한시간 특성의 기본값)를 대기합니다.

하지만 사용자가 첫 번째 리스너 포트를 중지하면 연결 *c1*이 *.jms/CF1*에 대한 여유 풀에 넣어집니다. 연결 관리자는 이 연결을 검색하고 이를 세 번째 리스너에게 제공합니다.

이제 사용자가 첫 번째 리스너를 다시 시작하려고 하면 이 리스너는 첫 번째 리스너가 재시작할 수 있기 전에 다른 리스너 포트 중 하나가 중지되는 것을 기다려야 합니다. 180초 내에 실행 중인 리스너 포트 중 하나가 중지되지 않으면 첫 번째 리스너는 *ConnectionWaitTimeoutException* 오류를 수신하고 중지됩니다.

아웃바운드 메시징을 수행하는 애플리케이션이 연결 풀을 사용하는 방법

이 옵션에 대해 단일 EJB(예: *EJB1*)가 애플리케이션 서버에 설치되어 있다고 가정합니다. Bean은 다음 방법으로 호출된 *sendMessage()*를 구현합니다.

- 공장에서 IBM MQ 에 대한 `javax.jms.ConnectionFactory.createConnection()`.
- 연결에서 JMS 세션 작성
- 세션에서 메시지 생성자 작성
- 메시지 송신
- 생성자 닫기
- 세션 닫기
- `connection.close()`를 호출하여 연결 닫기.

팩토리 `javax.jms.ConnectionFactory`에 대한 여유 풀이 비어 있다고 가정합니다. EJB가 처음 호출될 때, 빈은 IBM MQ 팩토리에서 연결을 생성하려고 `javax.jms.ConnectionFactory.createConnection()` 시도합니다. 팩토리에 대한 여유 풀이 비어 있으므로 연결 관리자는 새 연결을 작성하고 이를 EJB1에 제공합니다.

메서드가 종료되기 직전에, 메서드는 `connection.close()`를 호출합니다. `c1`을 닫지 않고, 연결 관리자는 연결을 받아서 이를 `javax.jms.ConnectionFactory`에 대한 여유 풀에 넣습니다.

다음에 `sendMessage()` 호출될 때, `javax.jms.ConnectionFactory.createConnection()` 메서드는 애플리케이션으로 `c1` 반환됩니다.

첫 번째 인스턴스와 동시에 실행 중인 두 번째 EJB 인스턴스가 있다고 가정합니다. 두 인스턴스가 `sendMessage()`를 호출하면 `javax.jms.ConnectionFactory` 연결 팩토리에서 두 개의 연결이 작성됩니다.

이제 Bean의 세 번째 인스턴스가 작성되었다고 가정합니다. 세 번째 Bean이 `sendMessage()`를 호출하면 메서드는 `javax.jms.ConnectionFactory`에서 연결을 작성하기 위해 `javax.jms.ConnectionFactory.createConnection()`을 호출합니다.

하지만 현재 이 팩토리에 대한 최대 연결 값인 2개의 연결이 `javax.jms.ConnectionFactory`에서 작성되어 있습니다. 따라서 `createConnection()` 메소드는 연결이 사용 가능해질 때까지 180초(연결 제한시간 특성의 기본값)를 대기합니다.

그러나 첫 번째 EJB에 대한 `sendMessage()` 메소드가 `connection.close()`을 호출한 후 종료하면 사용 중인 연결 `c1`이 여유 연결 풀로 다시 돌아갑니다. 연결 관리자는 여유 풀에서 연결을 다시 가져와서 이를 세 번째 EJB에게 제공합니다. 해당 Bean에서 `javax.jms.ConnectionFactory.createConnection()`으로 호출이 리턴되고 `sendMessage()` 메소드가 완료됩니다.

동일한 연결 풀을 사용하는 MDB 리스너 포트 및 EJB

앞의 두 예제에서는 리스너 포트와 EJB가 각각 독립된 연결 풀을 사용하는 방법을 보여주었습니다. 하지만 동일한 애플리케이션 서버 내에서 실행 중이고 동일한 연결 팩토리를 사용하여 JMS 연결을 작성하는 리스너 포트와 EJB가 있을 수 있습니다.

이러한 상황의 영향도 고려해야 합니다.

기억할 사항의 핵심은 리스너 포트와 EJB 사이에 연결 팩토리가 공유된다는 점입니다.

예를 들어 동시에 실행 중인 리스너와 EJB가 있다고 가정합니다. 둘 다 `javax.jms.ConnectionFactory` 연결 팩토리를 사용하고 있으며 이는 해당 팩토리에 대한 최대 연결 수 특성에서 지정한 연결 한계에 도달했음을 의미합니다.

또 다른 리스너 포트 또는 EJB의 또 다른 인스턴스를 시작하려는 경우, 둘 중 하나는 `javax.jms.ConnectionFactory`에 대한 여유 연결 풀에 연결이 돌아올 때까지 기다려야 합니다.

여유 연결 풀 유지보수 스레드

각 여유 연결 풀과 연관된 풀 유지보수 스레드는 여유 풀을 모니터링하여 풀에 있는 연결이 여전히 유효한지 확인합니다.

풀 유지보수 스레드가 사용 가능한 풀의 연결을 버려야 한다고 결정하는 경우, 스레드는 IBM MQ에 대한 JMS 연결을 실제로 닫습니다.

풀 유지보수 스레드가 작동하는 방법

풀 유지보수 스레드의 작동은 연결 풀의 4개 특성 값에 의해 결정됩니다.

Aged timeout

연결이 계속 열려 있는 시간입니다.

Minimum connections

연결 관리자가 연결 팩토리의 여유 풀에 보관하는 최소 연결 수입니다.

Reap time

풀 유지보수 스레드가 실행되는 주기입니다.

Unused timeout

연결이 닫히기 전에 여유 풀에 남아 있는 시간입니다.

기본적으로 풀 유지보수된 스레드는 180초마다 실행됩니다. 연결 풀 **Reap time** 특성을 설정하여 변경할 수 있습니다.

유지보수 스레드는 풀의 각 연결을 확인하고 풀에 얼마나 오래 있었는지와 마지막으로 사용된 후 경과한 시간을 검사합니다.

연결 풀의 **Unused timeout** 특성 값보다 긴 기간 동안 연결이 사용되지 않은 경우, 유지보수 스레드는 현재 사용 가능한 풀에 있는 연결 수를 확인합니다. 해당 수가 다음과 같을 경우

- **Minimum connections** 값보다 클 경우 연결 관리자는 연결을 다룹니다.
- **Minimum connections** 값과 같을 경우 연결이 닫히지 않고 여유 풀에 남아 있습니다.

Minimum connections 특성의 기본값은 1입니다. 이는 성능상의 이유로 연결 관리자가 여유 풀에 최소한 하나의 연결을 보관하도록 시도함을 의미합니다.

Unused timeout 특성은 기본값이 1800초입니다. 기본적으로 연결이 여유 풀에 돌아오고 최소 1800초 동안 다시 사용되지 않으면 연결이 닫히며, 해당 연결을 닫은 경우 여유 풀에 최소 하나의 연결이 남아 있게 됩니다.

이러한 프로시저는 사용되지 않는 연결이 시간 경과되지(stale) 않게 하기 위한 것입니다. 이 기능을 끄려면 **Unused timeout** 특성을 0으로 설정하십시오.

연결이 여유 풀에 있고 작성 후 경과한 시간이 연결 풀에 대한 **Aged timeout** 특성 값보다 큰 경우, 마지막 사용 후 경과한 시간에 관계없이 연결이 닫힙니다.

기본적으로 **Aged timeout** 특성은 0으로 설정되며 이는 유지보수 스레드가 이 검사를 수행하지 않음을 의미합니다. **Aged timeout** 특성보다 오래된 연결은 얼마나 많은 연결이 여유 풀에 남아 있게 되는지에 관계없이 제거됩니다. **Minimum connections** 특성은 이러한 경우 영향을 주지 않는다는 점을 참고하십시오.

풀 유지보수 스레드 사용 안함

앞의 설명에서 풀 유지보수 스레드가 활성화될 때 많은 일을 한다는 것을 알 수 있습니다. 특히 연결 팩토리의 여유 풀에 다수의 연결이 있을 때 그렇습니다.

예를 들어 세 개의 JMS 연결 팩토리가 있고 각 팩토리에 대해 **Maximum connections** 특성을 10으로 설정했다고 가정합니다. 180초마다 세 개의 풀 유지보수 스레드가 활성화되고 각 연결 팩토리에 대해 여유 풀을 각각 스캔합니다. 여유 풀에 많은 연결이 있으면 유지보수 스레드가 해야 할 일이 많아지고 이는 성능에 상당한 영향을 줄 수 있습니다.

Reap time 특성을 0으로 설정하여 개별 여유 연결 풀에 대한 풀 유지보수 스레드를 사용 안함으로 설정할 수 있습니다.

유지보수 스레드를 사용 안함으로 설정한다는 것은 **Unused timeout**이 경과한 경우라도 연결을 절대 닫지 않음을 의미합니다. 그러나 **Aged timeout**을 초과하면 연결이 여전히 닫힙니다.

애플리케이션이 연결을 완료하면 연결 관리자는 연결이 존재한 기간을 확인하고 해당 기간이 **Aged timeout** 특성의 값보다 긴 경우 사용 가능한 풀로 리턴하지 않고 연결을 닫습니다.

유효 제한시간(Aged timeout)의 트랜잭션 영향

앞의 절에서 설명했듯이 **Aged timeout** 특성은 JMS 제공자에 대한 연결이 연결 관리자가 이를 닫기 전에 열린 상태로 남아 있는 시간을 지정합니다.

Aged timeout 특성의 기본값은 0이고 이는 연결이 너무 오래되어서 닫히는 경우가 없음을 의미합니다. **Aged timeout** 를 사용으로 설정하면 EJB 내부에서 JMS 를 사용할 때 트랜잭션에 영향을 줄 수 있으므로 **Aged timeout** 특성을 이 값으로 두어야 합니다.

JMS에서 트랜잭션의 단위는 JMS 세션이고 JMS 연결에서 작성됩니다. 이는 JMS 연결이 아닌, 트랜잭션에 등록되는 JMS 세션입니다.

애플리케이션 서버의 설계로 인해 해당 연결에서 작성된 JMS 세션이 트랜잭션에 포함되는 경우라도 **Aged timeout**이 초과되면 JMS 연결이 닫힐 수 있습니다.

JMS 연결을 닫으면 JMS 세션에서 진행 중인 트랜잭션 작업이 롤백됩니다(JMS 스펙 설명 참조). 그러나 애플리케이션 서버는 연결에서 작성된 JMS 세션이 더 이상 유효하지 않음을 인식하지 못합니다. 서버가 트랜잭션을 커밋하거나 롤백하기 위해 세션을 사용하려고 하면 `IllegalStateException`이 발생합니다.

중요사항: EJB내에서 **Aged timeout** 를 JMS 연결과 함께 사용하려면 JMS 조작을 수행하는 EJB 메소드가 종료되기 전에 JMS 작업이 JMS 세션에서 명시적으로 커밋되었는지 확인하십시오.

풀 유지보수 스레드 예제

엔터프라이즈 자바 애플리케이션(JavaBean, EJB) 예제를 통해 풀 유지 관리 스레드의 작동 방식을 이해하기 참고로, 여유 풀에 연결을 얻기 위한 방법이 필요한 것이므로 Message Driven Beans(MDBs)와 리스너 포트를 사용할 수도 있습니다.

`sendMessage()` 메소드에 대한 추가 세부사항은 284 페이지의 [『아웃바운드 메시징을 수행하는 애플리케이션이 연결 풀을 사용하는 방법』](#)의 내용을 참조하십시오.

다음 값으로 연결 팩토리를 구성했습니다.

- **Reap time**, 기본값 180초로 설정
- **Aged timeout**, 기본값 0초로 설정
- **Unused timeout** 300초로 설정

애플리케이션 서버가 시작된 후 `sendMessage()` 메소드가 호출됩니다.

메소드는 팩토리 `.jms/CF1`을 사용하는 `c1`이라는 연결을 작성합니다. 그런 다음 `connection.close()`를 호출하여 `c1`을 여유 풀에 넣습니다.

180초 후에 풀 유지보수 스레드가 시작되고 `.jms/CF1` 여유 연결 풀을 확인합니다. 풀에 여유 연결 `c1`이 있으므로 유지보수 스레드는 연결이 풀에 다시 넣어진 시점을 확인하고 현재 시간과 비교합니다.

연결이 여유 풀에 넣어진 후 180초가 지났으며 이 값은 `.jms/CF1`에 대한 **Unused timeout** 특성 값보다 작은 수입니다. 따라서 유지보수 스레드가 연결을 단독으로 남겨둡니다.

180초 후에 풀 유지보수 스레드가 다시 실행됩니다. 유지보수 스레드가 연결 `c1`을 찾고 해당 연결이 360초 동안 풀에 있었음을 판별합니다. 이 시간은 **Unused timeout**에 설정된 값보다 긴 시간이므로 연결 관리자가 연결을 닫습니다.

`sendMessage()` 메소드를 다시 실행하고 애플리케이션이 `connectionFactory.createConnection()`을 호출하면 연결 관리자는 연결 팩토리에 대한 연결 풀이 비어 있으므로 IBM MQ에 대해 새 연결을 작성합니다.

앞의 예제에서는 **Aged timeout** 특성이 0으로 설정될 때, 유지보수 스레드가 **Reap time** 및 **Unused timeout** 특성을 사용하여 연결이 시간 경과(stale)되는 것을 방법을 보여줍니다.

Aged timeout 특성의 작동 방법

다음 예제에서 아래와 같이 특성을 설정했다고 가정합니다.

- **Aged timeout** 특성을 300초로 설정
- **Unused timeout** 특성을 0으로 설정

`sendMessage()` 메소드를 호출하고 이 메소드는 `.jms/CF1` 연결 팩토리에서 연결을 작성하려고 시도합니다.

이 팩토리에 대한 여유 풀이 비어 있으므로 연결 관리자는 새로운 연결 `c1`을 작성하고 이를 애플리케이션에 리턴합니다. `sendMessage()`가 `connection.close()`를 호출하면 `c1`이 여유 연결 풀에 다시 넣어집니다.

180초 후, 풀 유지보수 스레드가 실행됩니다. 스레드는 여유 연결 풀에서 c1을 찾고 작성 이후 경과된 시간을 검사합니다. 연결이 180초 동안 존재했으면 이 시간이 **Aged timeout**의 값보다 작으므로 풀 유지보수 스레드는 연결을 단독으로 남겨두고 다시 휴면 상태가 됩니다.

60초 후에 `sendMessage()`가 다시 호출됩니다. 이번에 메소드가 `connectionFactory.createConnection()`을 호출하고 연결 관리자는 `jms/CF1`에 대한 여유 풀에 사용 가능한 c1 연결이 있음을 발견합니다. 연결 관리자는 풀에서 c1을 가져와서 해당 연결을 애플리케이션에 제공합니다.

`sendMessage()`가 종료되면 연결은 여유 풀로 리턴됩니다. 120초 후에 풀 유지보수 스레드가 휴면 상태에서 깨어나 120 `jms/CF1`에 대한 여유 풀 콘텐츠를 스캔하고 c1을 발견합니다.

연결이 불과 120초 전에 사용되었더라도 풀 유지보수 스레드는 연결이 총 360초 동안 존재해왔고 이는 **Aged timeout** 특성에 설정한 값인 300초보다 길기 때문에 연결을 닫습니다.

최소 연결 수 특성이 풀 유지보수 스레드에 영향을 미치는 방법

284 페이지의 『MDB 리스너 포트가 연결 풀을 사용하는 방법』 예제를 다시 사용하여, 애플리케이션 서버에 MDB 두 개가 배치되어 있고 각각 다른 리스너 포트를 사용한다고 가정합니다.

각 리스너 포트는 `jms/CF1` 연결 팩토리를 사용하도록 구성되어 있으며 사용자가 다음 특성으로 구성하였습니다.

- **Unused timeout** 특성을 120초로 설정
- **Reap time** 특성을 180초로 설정
- **Minimum connections** 특성을 1로 설정

첫 번째 리스너가 중지되고 이의 연결 c1이 여유 풀에 넣어진다고 가정합니다. 180초 후에 풀 유지보수 스레드가 휴면 상태에서 깨어나 `jms/CF1`에 대한 여유 풀 콘텐츠를 스캔하고 c1이 연결 팩토리에 대한 **Unused timeout** 특성 값보다 긴 시간 동안 여유 풀에 있었음을 발견합니다.

그러나 c1을 닫기 전에 풀 유지보수 스레드는 이 연결을 제거할 경우 풀에 남아 있게 될 연결 수를 먼저 확인합니다. c1이 여유 연결 풀에 있는 유일한 연결이므로 연결 관리자가 이를 닫지 않습니다. 이 연결을 닫게 되면 여유 풀에 남아 있는 연결 수가 **Minimum connections**에 설정된 값보다 작아지기 때문입니다.

이제 두 번째 리스너가 중지된다고 가정합니다. 여유 연결 풀은 이제 두 개의 여유 연결 c1 및 c2를 포함합니다.

180초 후에 풀 유지보수 스레드가 다시 실행됩니다. 이 시간까지 c1은 360초 동안 여유 연결 풀에 있었고, c2는 180초 동안 있었습니다.

풀 유지보수 스레드는 c1을 검사하고 **Unused timeout** 특성 값보다 오랫동안 풀에 있었음을 발견합니다.

그런 다음 스레드는 여유 풀에 있는 연결 수를 확인하고 이 수를 **Minimum connections** 특성 값과 비교합니다. 풀에 2개의 연결이 포함되어 있고 **Minimum connections**가 1로 설정되어 있으므로 연결 관리자가 c1을 닫습니다.

유지보수 스레드가 이제 c2를 확인합니다. 이 연결 역시 **Unused timeout** 특성 값보다 오래 여유 연결 풀에 있었습니다. 그러나 종료 c2 시 무료 연결 풀에 설정된 최소 연결 수보다 적은 연결이 남게 되므로, 연결 관리자는 이를 그대로 c2 둡니다.

JMS 연결 및 IBM MQ

'IBM MQ'를 'JMS' 제공업체로 사용하는 것에 대한 정보입니다.

바인딩 전송 사용

연결 팩토리가 바인딩 전송을 사용하도록 구성된 경우, 모든 'JMS 연결은 'hconn'와의 대화('IBM MQ'라고도 함)를 설정합니다. 대화는 프로세스간 통신(또는 공유 메모리)을 사용하여 큐 관리자와 통신합니다.

클라이언트 전송 사용

클라이언트 전송을 사용하도록 'IBM MQ 메시징 공급자 연결 팩토리가 구성된 경우, 해당 팩토리에서 생성된 모든 연결은 **hconn**)에서 'IBM MQ' 로의 새 대화를 설정합니다.

IBM MQ 메시징 제공자 정상 모드를 사용하여 큐 관리자에 연결하는 연결 팩토리의 경우 연결 팩토리에서 작성된 다중 JMS 연결이 IBM MQ와의 TCP/IP 연결을 공유하는 것이 가능합니다. 자세한 정보는 [291 페이지의 『IBM MQ classes for JMS에서 TCP/IP 연결 공유』](#)의 내용을 참조하십시오.

한 번에 JMS 연결에서 사용하는 최대 클라이언트 채널 수를 판별하려면 동일한 큐 관리자를 가리키는 모든 연결 팩토리에 대한 *Maximum connections* 특성 값을 더하십시오.

예를 들어 두 개의 연결 팩토리 `jms/CF1`과 `jms/CF2`를 가지고 있고 동일한 IBM MQ 채널을 사용하여 동일한 IBM MQ 큐 관리자에 연결하도록 구성되어 있다고 가정합니다.

이러한 팩토리는 기본 연결 풀 특성을 사용하며 이는 *Maximum connections*가 10으로 설정됨을 의미합니다. 모든 연결이 동시에 `jms/CF1`과 `jms/CF2`에서 사용되고 있다고 할 경우 애플리케이션 서버와 IBM MQ 사이에는 20개의 대화가 있게 됩니다.

연결 팩토리가 IBM MQ 메시징 제공자 정상 모드를 사용하여 큐 관리자에 연결할 경우 애플리케이션 서버와 이러한 연결 팩토리에 대한 큐 관리자 사이에 존재할 수 있는 최대 TCP/IP 연결 수는 다음과 같습니다.

```
20/the value of SHARECNV for the IBM MQ channel
```

연결 팩토리가 IBM MQ 메시징 제공자 마이그레이션 모드를 사용하여 연결하도록 구성되면 해당 연결 팩토리에 대한 애플리케이션 서버와 IBM MQ 사이의 최대 TCP/IP 연결 수는 20개(두 개의 팩토리에 대한 연결 풀에 있는 각 JMS 연결에 대해 하나씩)가 됩니다.

관련 개념

[83 페이지의 『사용 IBM MQ classes for JMS/Jakarta Messaging』](#)

IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 는 와 함께 제공되는 IBM MQ메시징 Java 공급자입니다. 이 메시징 공급자들은 및 Jakarta Messaging JMS 사양에서 정의된 인터페이스를 구현하는 것 외에도, 메시징 Java API에 두 가지 확장 기능을 추가합니다.

Java SE 환경에서 오브젝트 풀링

Java SE를 사용하면(또는 Spring 등의 다른 프레임워크를 사용하면) 프로그래밍 모델이 매우 탄력적입니다. 그러므로 단일 풀링 전략이 모두 적합하지는 않습니다. 풀링 양식을 수행할 수 있는 위치에 프레임워크가 있는지 고려해야 합니다(예: Spring).

그렇지 않으면 애플리케이션 논리가 이를 시작할 수 있습니다. 애플리케이션 자체가 얼마나 복잡한지 자문해 보십시오. 애플리케이션 및 연결성에서 메시징 시스템에 이르기까지 필요로 하는 사항을 이해하는 것이 가장 중요합니다. 애플리케이션은 기본 JMS API 주위에 자체 래퍼 코드 내에서도 마찬가지로 기록됩니다.

이는 매우 합리적인 접근법이고 복잡도를 숨길 수 있는 반면, 문제점을 유발할 수 있습니다. 예를 들어, 자주 호출되는 일반 `getMessage()` 메소드는 이용자를 열고 닫지 않아야 합니다.

고려해야 하는 사항:

- 애플리케이션이 IBM MQ에 대한 액세스를 필요로 하는 기간은 얼마입니까? 항상 또는 가끔씩.
- 메시지가 얼마나 자주 전송됩니까? 덜 자주 전송될수록 IBM MQ에 대한 단일 연결이 더 많이 공유될 수 있습니다.
- 연결 중단 예외는 일반적으로 풀링된 연결을 재작성하기 위해 필요한 부호입니다. 대상:
 - 사용 불가능한 보안 예외 또는 호스트
 - 전체 예외 큐잉
- 연결 중단 예외가 발생하면 풀의 다른 자유 연결에 어떤 일이 발생해야 합니까? 닫고 다시 작성해야 합니까?
- TLS가 사용 중이면, 예를 들어 단일 연결을 얼마나 오랜 시간 동안 열린 채 유지시킬 예정입니까?
- 풀링된 연결이 큐 관리자가 연결을 선택하고 이를 다시 추적할 수 있는 경우 자체적으로 식별하는 방법은 무엇입니까?

풀링을 위해 모든 JMS 오브젝트를 고려하고 이를 수행할 수 있을 때마다 해당 오브젝트를 풀링해야 합니다. 오브젝트에는 다음이 포함됩니다.

- JMS 연결
- 세션
- 컨텍스트
- 모든 다른 유형의 생성자 및 이용자

클라이언트 전송을 사용할 경우 JMS 연결, 세션 및 컨텍스트가 IBM MQ 큐 관리자와 통신할 때 소켓을 사용합니다. 이러한 오브젝트를 풀링하면 큐 관리자에 대해 수신되는 IBM MQ 연결(hConns)의 수가 절약되고 채널 인스턴스의 수도 줄어듭니다.

큐 관리자에 대한 바인딩 전송을 사용하면 네트워킹 계층 전체가 제거됩니다. 그러나 많은 애플리케이션이 보다 가용성이 높고 워크로드가 밸런싱된 구성을 제공하기 위해 클라이언트 전송을 사용합니다.

JMS 생성자 및 이용자는 큐 관리자에 목적지를 엽니다. 더 적은 수의 큐 또는 토픽이 열려 있고 애플리케이션의 다중 부분이 이러한 오브젝트를 사용 중인 경우, 이러한 풀링이 유용할 수 있습니다.

IBM MQ 퍼스펙티브에서 이 프로세스는 MQOPEN 및 MQCLOSE 조작의 순서를 저장합니다.

연결, 세션 및 컨텍스트

이러한 오브젝트는 모두 IBM MQ 연결 핸들을 큐 관리자로 캡슐화하며, ConnectionFactory에서 생성됩니다. 단일 연결 팩토리에서 작성된 연결 수 및 기타 오브젝트를 특정 수로 제한하려면 애플리케이션에 논리를 추가할 수 있습니다.

작성된 연결이 포함되도록 애플리케이션에서 단순 데이터 구조를 사용할 수 있습니다. 이러한 데이터 구조 중 하나를 사용해야 하는 애플리케이션 코드는 사용할 오브젝트를 체크아웃할 수 있습니다.

다음 요인을 고려하십시오.

- 풀에서 연결을 제거해야 하는 시점이 언제입니까? 일반적으로 연결에 대한 예외 리스너를 작성하십시오. 리스너가 예외를 처리하기 위해 호출된 경우, 연결 및 해당 연결로부터 작성된 세션을 재작성해야 합니다.
- CCDT가 워크로드 밸런싱을 위해 사용 중인 경우, 연결은 다른 큐 관리자로 이동할 수 있습니다. 이는 풀링 요구 사항에 적용할 수 있습니다.

JMS 스펙은 다중 스레드에 대해 동시에 세션 또는 컨텍스트에 대해 액세스되도록 하는 경우 프로그래밍 오류가 있음을 나타냄을 기억하십시오. IBM MQ JMS 코드는 스레드 처리 시 엄격하도록 시도를 수행합니다. 그러나 세션 또는 컨텍스트 오브젝트가 한 번에 하나의 스레드에 의해서만 사용되도록 보장하려면 논리를 애플리케이션에 추가해야 합니다.

생성자와 이용자

작성되는 각 생성자와 이용자는 큐 관리자에서 목적지를 엽니다. 동일한 목적지가 다양한 태스크에 대해 사용될 예정이면 이용자 또는 생성자 오브젝트가 열려 있을 수 있습니다. 모든 작업이 수행된 경우에만 오브젝트를 닫으십시오.

목적지 열기 및 닫기가 짧은 조작이더라도 자주 수행될 경우 처리 시간이 추가될 수 있습니다.

이러한 오브젝트 범위는 작성되는 세션 또는 컨텍스트 내에 있으므로 해당 범위 내에 보유되어야 합니다. 일반적으로 애플리케이션은 수행하기가 꽤 쉽도록 작성됩니다.

모니터링

애플리케이션이 오브젝트 풀을 모니터링하는 방법은 무엇입니까? 이에 대한 응답은 대개 구현된 풀링 솔루션의 복잡도에 의해 판별됩니다.

JavaEE 풀링 구현을 고려할 경우, 다음을 비롯한 많은 수의 옵션이 있습니다.

- 풀의 현재 크기
- 풀에서 오브젝트가 사용한 시간
- 풀 정리
- 연결 새로 고치기

재사용된 단일 세션이 큐 관리자에 표시되는 방법도 고려해야 합니다. 애플리케이션을 식별하기 위한 유용한 연결 팩토리 특성이 있습니다(예: appName).

83 페이지의 『사용 IBM MQ classes for JMS/Jakarta Messaging』

IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 는 와 함께 제공되는 IBM MQ메시징 Java 공급자입니다. 이 메시징 공급자들은 및 Jakarta Messaging JMS 사양에서 정의된 인터페이스를 구현하는 것 외에도, 메시징 Java API에 두 가지 확장 기능을 추가합니다.

IBM MQ classes for JMS에서 TCP/IP 연결 공유

단일 TCP/IP 연결을 공유하도록 MQI 채널의 다중 인스턴스를 작성할 수 있습니다.

동일한 Java 런타임 환경 내에서 실행되며 CLIENT 전송을 사용하여 큐 관리자에 연결하기 위해 또는 IBM MQ classes for JMS 리소스 IBM MQ 어댑터를 사용하는 애플리케이션은 채널 인스턴스를 공유하도록 구성할 수 있습니다.

SHARECNV 매개변수를 1보다 큰 값으로 설정하여 채널을 정의한 경우, 해당 수의 대화에서 채널 인스턴스를 공유할 수 있습니다. 연결 팩토리 또는 활성화 스펙을 사용으로 설정하여 이 기능을 사용하려면

SHARECNVALLOWED 특성을 YES로 설정하십시오.

JMS 애플리케이션이 작성한 모든 JMS 연결 및 JMS 세션은 큐 관리자와의 자체 대화를 작성합니다.

활성화 스펙이 시작될 때 IBM MQ 자원 어댑터는 사용할 활성화 스펙에 대해 큐 관리자와 대화를 시작합니다. 또한 활성화 스펙과 연관된 서버 세션 풀의 모든 서버 세션은 큐 관리자와 대화를 시작합니다.

해당 **SHARECNV** 속성은 연결 공유에 대한 최선의 노력 방식입니다. 따라서, 값이 **SHARECNV** 0보다 클 때 사용될 경우 IBM MQ classes for JMS, 새 연결 요청이 항상 이미 설정된 연결을 공유한다고 보장되지 않습니다.

TCP/IP 연결이 어떻게 공유되는지

TCP/IP 연결을 공유하는 데는 두 가지 방법이 있습니다:

글로벌 전략

이 전략은 TCP/IP 연결을 공유할 때의 기본 전략입니다. 모든 JMS 연결 또는 세션은 적합한 TCP/IP 연결을 통해 대화를 이용할 수 있습니다. 적합성은 호스트 주소, 포트 번호, 사용자 ID 및 비밀번호, 그리고 TLS / SSL 매개변수 등의 요소에 의해 결정됩니다.

TCP/IP 연결을 공유하는 이 방식은 사용 중인 채널 인스턴스의 수를 최소화하지만, 그 대가로 TCP/IP 연결의 전역 풀에 대한 액세스 경쟁이 발생합니다.

연결 전략

이 전략을 사용하면 채널 인스턴스는 관련 JMS 객체들 사이에서만 공유됩니다. 구체적으로, 연결이 생성되면 해당 JMS 연결을 위한 채널 인스턴스가 생성되며, 해당 채널 인스턴스 상의 추가 대화는 해당 JMS 연결로 생성된 JMS 세션에서만 이용할 수 있습니다.

SHARECNV 속성이 지정하는 것보다 더 많은 대화가 생성되면, 새 채널 인스턴스가 생성됩니다. 이 채널 인스턴스는 원래 JMSJMS 연결로 생성된 세션에서만 사용할 수 있습니다.

채널 인스턴스를 공유하는 이러한 접근 방식은 대화 경쟁을 줄여주지만, 그 대가로 훨씬 더 많은 채널 인스턴스가 필요할 수 있습니다.

채널 인스턴스 공유 전략 명시적 지정

기본적으로 애플리케이션이 재연결 불가능한 경우 GLOBAL 전략이 사용됩니다. 재연결 가능 애플리케이션은 항상 CONNECTION 전략을 사용합니다.

또는 를 IBM MQ classes for JMS 사용하는 IBM MQ classes for Jakarta Messaging애플리케이션의 경우, 재연결 불가능한 애플리케이션에 대해 애플리케이션 전체 수준에서 CONNECTION 전략을 활성화할 수 있습니다. 시스템 속성을 com.ibm.mq.jms.channel.sharing 값으로 설정하여 CONNECTION 전략을 활성화할 수 있습니다. 이 값은 대소문자를 구분하지 않으며, 이외의 값은 CONNECTION 무시됩니다.

다음 방법 중 하나로 시스템 com.ibm.mq.jms.channel.sharing 속성을 설정할 수 있습니다:

- ' "-D" ' 명령줄 옵션을 사용하여 ' JVM ' 초기화의 일부로 속성을 설정하십시오:

```
-Dcom.ibm.mq.jms.channel.sharing=CONNECTION
```


- 사용 전에 또는 IBM MQ classes for JMS 을 IBM MQ classes for Jakarta Messaging 사용하기 전에 속성을 설정하십시오. `System.setProperty()`

GLOBAL 공유 전략에 대한 채널 인스턴스 수 계산

다음 공식을 사용하여 애플리케이션이 작성한 채널 인스턴스의 최대 수를 판별할 수 있습니다.

활성화 스펙

$$\text{채널 인스턴스 수} = (\text{maxPoolDepth\_value} + 1) / \text{SHARECNV\_value}$$

여기서, `maxPoolDepth_value`는 `maxPoolDepth` 특성의 값이고 `SHARECNV_value`는 활성화 스펙에서 사용하는 채널의 `SHARECNV` 특성 값입니다.

기타 JMS 애플리케이션

$$\text{채널 인스턴스 수} = (\text{jms\_connections} + \text{jms\_sessions}) / \text{SHARECNV\_value}$$

`jms_connections`는 애플리케이션이 생성한 연결 수를, `jms_sessions`는 애플리케이션이 생성한 세션 JMS 수를, `SHARECNV_value`는 활성화 사양에서 사용되는 채널의 속성 `SHARECNV` 값을 나타냅니다.

CONNECTION 공유 전략에 대한 채널 인스턴스 수 계산

채널 인스턴스의 수는 애플리케이션 내 연결 JMS 간 세션 JMS 분배에 따라 달라집니다.

연결당 JMS 하나의 대화를 허용하고, 해당 JMS 연결 아래의 각 JMS 세션당 하나의 대화를 허용한 후, 값으로 `SHARECNV` 나누어 올림 처리합니다. 이 계산은 해당 JMS 연결에 필요한 채널 인스턴스를 제공합니다.

동일한 원칙이 활성화 사양에도 적용될 수 있습니다. 활성화 사양을 연결로 JMS 간주하고 속성을 `maxPoolDepth` 세션 JMS 수로 간주하십시오.

예

다음 예제는 공식을 이용함으로써 IBM MQ classes for JMS 또는 IBM MQ 자원 어댑터를 사용하여 애플리케이션이 큐 관리자에서 작성하는 채널 인스턴스의 수를 계산하는 방법을 표시합니다.

JMS 애플리케이션 예제

JMS 애플리케이션 연결은 CLIENT 전송을 사용하여 큐 관리자에 연결하며, 하나의 JMS 연결 및 세 개의 JMS 세션을 작성합니다. 애플리케이션이 큐 관리자에 연결하기 위해 사용 중인 채널에는 `SHARECNV` 특성의 값이 10으로 설정되어 있습니다. 애플리케이션이 실행 중일 때는 애플리케이션과 큐 관리자 간의 네 개의 대화와 하나의 채널 인스턴스가 있습니다. 네 개의 대화는 모두 채널 인스턴스를 공유합니다.

활성화 스펙 예제

활성화 스펙은 CLIENT 전송을 사용하여 큐 관리자에 연결됩니다. 활성화 스펙은 `maxPoolDepth` 특성을 10으로 설정하여 구성합니다. 활성화 스펙이 사용하도록 구성된 채널에는 `SHARECNV` 특성이 10으로 설정되어 있습니다. 활성화 스펙이 실행 중이며 10개의 메시지를 동시에 처리 중인 경우, 활성화 스펙 및 큐 관리자 간의 대화 수는 11개(서버 세션에 대해 10개의 대화 및 활성화 스펙에 대해 1개의 대화)입니다. 활성화 스펙에서 사용하는 채널 인스턴스의 수는 2개입니다.

활성화 스펙 예제

활성화 스펙은 CLIENT 전송을 사용하여 큐 관리자에 연결됩니다. 활성화 스펙은 `maxPoolDepth` 특성을 5로 설정하여 구성됩니다. 활성화 스펙이 사용하도록 구성된 채널에는 `SHARECNV` 특성이 0으로 설정되어 있습니다. 활성화 스펙이 실행 중이고 5개의 메시지를 동시에 처리하는 경우 활성화 스펙과 큐 관리자 간의 대화 수는 6(서버 세션에 대한 5개의 대화 및 활성화 스펙에 대한 1개의 대화)입니다. 활성화 사양에서 사용되는 채널 인스턴스 수는 6개입니다. 채널의 속성이 `SHARECNV` 0으로 설정되어 있으므로, 각 대화는 자체 채널 인스턴스를 사용하기 때문입니다.

관련 태스크

473 페이지의 『WebSphere Application Server에서 IBM MQ에 대해 작성된 TCP/IP 연결 수 판별』

공유 대화 기능을 사용하여 다중 대화에서 MQI 채널 인스턴스(TCI/IP 연결이라고도 함)를 공유할 수 있습니다.

IBM MQ classes for JMS에서 클라이언트 연결의 포트 범위 지정

LOCALADDRESS 특성을 사용하여 애플리케이션이 바인딩될 수 있는 포트의 범위를 지정할 수 있습니다.

IBM MQ classes for JMS 애플리케이션이 클라이언트 모드로 IBM MQ 큐 관리자에 연결을 시도하는 경우, 방화벽은 지정된 포트 또는 포트 범위에서 발생하는 해당 연결만 허용할 수 있습니다. 이러한 상황에서는 `ConnectionFactory`, `QueueConnectionFactory` 또는 `TopicConnectionFactory` 오브젝트의 `LOCALADDRESS` 특성을 사용하여 애플리케이션이 바인딩될 수 있는 포트, 포트 범위를 지정할 수 있습니다.

IBM MQ JMS 관리 도구를 사용하여 또는 JMS 애플리케이션에서 `setLocalAddress()` 메소드를 호출하여 `LOCALADDRESS` 특성을 설정할 수 있습니다. 다음은 애플리케이션 내에서 특성을 설정하는 예제입니다.

```
mqConnectionFactory.setLocalAddress("192.0.2.0(2000,3000)");
```

애플리케이션이 나중에 큐 관리자에 연결될 때 애플리케이션은 192.0.2.0(2000) - 192.0.2.0(3000) 범위의 로컬 IP 주소와 포트 번호에 바인딩됩니다.

둘 이상의 네트워크 인터페이스가 있는 시스템에서는 `LOCALADDRESS` 특성을 사용하여 연결에 사용되어야 하는 네트워크 인터페이스를 지정할 수도 있습니다.

브로커에 대한 실시간 연결의 경우, `LOCALADDRESS` 특성은 멀티캐스트가 사용될 때만 관련됩니다. 이 경우에는 연결에 사용되어야 하는 로컬 네트워크 인터페이스를 지정하기 위한 특성을 사용할 수 있지만, 특성 값에는 포트 번호 또는 포트 번호 범위가 포함되지 않아야 합니다.

포트의 범위를 제한하는 경우에는 연결 오류가 발생할 수 있습니다. 오류가 발생하는 경우, IBM MQ 이유 코드 `MQRC_Q_MGR_NOT_AVAILABLE` 및 다음 메시지가 포함된 임베드된 `MQException`과 함께 `JMSException`이 발생합니다.

Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions

지정된 범위의 모든 포트가 사용 중이거나 지정된 IP 주소, 호스트 이름 또는 포트 번호가 올바르지 않은 경우(예: 음수 포트 번호)에는 오류가 발생할 수 있습니다.

IBM MQ classes for JMS가 애플리케이션에 필요한 연결이 아닌 연결을 작성할 수 있으므로, 항상 포트 범위를 지정할 것을 고려하십시오. 일반적으로 애플리케이션이 작성한 모든 세션에는 하나의 포트가 필요하여, IBM MQ classes for JMS는 세 개나 네 개의 추가 포트를 요구할 수 있습니다. 연결 오류가 발생하면 포트의 범위를 늘리십시오.

기본적으로 IBM MQ classes for JMS에서 사용되는 연결 풀은 포트가 재사용될 수 있는 속도에 영향을 줄 수 있습니다. 결과적으로, 포트가 해제되는 동안 연결 오류가 발생할 수 있습니다.

IBM MQ classes for JMS의 채널 압축

IBM MQ classes for JMS 애플리케이션은 IBM MQ 기능을 사용하여 메시지 헤더 또는 데이터를 압축할 수 있습니다.

IBM MQ 채널에서 이동되는 데이터를 압축하면 채널의 성능이 개선되고 네트워크 트래픽이 감소될 수 있습니다. IBM MQ에서 제공하는 기능을 사용하면 메시지 채널 및 MQI 채널에서 이동되는 데이터를 압축할 수 있습니다. 두 가지 채널 유형에서는 서로 간에 독립적으로 헤더 데이터 및 메시지 데이터를 압축할 수 있습니다. 기본적으로 채널에서 데이터는 압축되지 않습니다.

IBM MQ classes for JMS 애플리케이션은 `java.util.Collection` 오브젝트를 작성하여 연결에서 헤더 또는 메시지 데이터를 압축하는 데 사용될 수 있는 기술을 지정합니다. 각 압축 기술은 컬렉션의 정수 오브젝트이며, 애플리케이션이 컬렉션에 압축 기술을 추가하는 순서는 애플리케이션이 연결을 작성할 때 압축 기술이 큐 관리자와 협상되는 순서입니다. 그리고 애플리케이션은 `setHdrCompList()` 메소드(헤더 데이터의 경우) 또는 `setMsgCompList()` 메소드(메시지 데이터의 경우)를 호출하여 컬렉션을 `ConnectionFactory` 오브젝트에 전달할 수 있습니다. 애플리케이션이 준비되면 이는 연결을 작성합니다.

다음 코드 단편은 설명된 접근 방식을 예시합니다. 첫 번째 코드 단편은 헤더 데이터 압축을 구현하는 방법을 보여줍니다.

```
Collection headerComp = new Vector();
headerComp.add(new Integer(WMQConstants.WMQ_COMPHDR_SYSTEM));
.
.
.
((MQConnectionFactory) cf).setHdrCompList(headerComp);
.
.
.
connection = cf.createConnection();
```

두 번째 코드 단편은 메시지 데이터 압축을 구현하는 방법을 표시합니다.

```
Collection msgComp = new Vector();
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_RLE));
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_ZLIBHIGH));
.
.
.
((MQConnectionFactory) cf).setMsgCompList(msgComp);
.
.
.
connection = cf.createConnection();
```

두 번째 예제에서는 연결이 생성될 때 압축 기술이 RLE, ZLIBHIGH 순서로 협상됩니다. 선택된 압축 기술은 Connection 오브젝트의 수명 중에는 변경될 수 없습니다. 연결에서 압축을 사용하려면, Connection 오브젝트를 작성하기 전에 setHdrCompList() 및 setMsgCompList() 메소드를 호출해야 합니다.

IBM MQ classes for JMS에서 비동기로 메시지 넣기

일반적으로, 애플리케이션이 목적지에 메시지를 송신할 때 애플리케이션은 큐 관리자가 요청이 처리되었는지 확인할 때까지 대기해야 합니다. 비동기 메시지 넣기를 대신 선택함으로써 일부 상황에서 메시징 성능을 개선할 수 있습니다. 애플리케이션이 메시지를 비동기로 넣는 경우, 큐 관리자는 각 호출의 성공 또는 실패를 리턴하지 않지만 사용자가 대신 주기적으로 오류를 확인할 수 있습니다.

큐 관리자가 메시지를 안전하게 수신했는지 여부를 판별함이 없이 목적지가 애플리케이션에 제어를 리턴하는지 여부는 다음 특성에 달려 있습니다.

JMS 목적지 특성 **PUTASYNCALLOWED**(단축 이름 - **PAALD**).

JMS 목적지가 나타내는 기본 큐 또는 토픽에서 이 옵션을 허용하는 경우, PUTASYNCALLOWED는 JMS 애플리케이션이 메시지를 비동기적으로 넣을 수 있는지 여부를 제어합니다.

IBM MQ 큐 또는 토픽 특성 **DEFPRESP**(기본 Put 응답 유형).

DEFPRESP는 큐에 메시지를 넣거나 토픽에 메시지를 발행하는 애플리케이션이 비동기 넣기 기능을 활용할 수 있는지 여부를 지정합니다.

다음 표에서는 PUTASYNCALLOWED 및 DEFPRESP 특성의 가능한 값과 비동기 넣기 기능을 사용하는 데 필요한 값 조합을 보여줍니다.

표 46. 메시지를 목적지에 비동기적으로 넣을지 여부를 판별하기 위해 PUTASYNCALLOWED 및 DEFPRESP 특성을 결합하는 방법			
IBM MQ 큐 특성	PUTASYNCALLOWED = NO	PUTASYNCALLOWED = YES	비동기 넣기 기능 사용
DEFPRESP=SYNC	비동기 넣기 기능 사용 안함	비동기 넣기 기능 사용	PUTASYNCALLOWED = AS_DEST 또는 AS_Q_DEF 또는 AS_T_DEF
DEFPRESP=ASYN	비동기 넣기 기능 사용 안함	비동기 넣기 기능 사용	PUTASYNCALLOWED = AS_DEST 또는 AS_Q_DEF 또는 AS_T_DEF

-JMS 대상 IBM MQ 속성을 표에 표시된 대로 "NO" 또는 "YES"로 지정하여 동작을 변경할 수 있지만, 다음 설정 값을 사용하여 전체 Java 가상 **SystemProperty** 머신에 대해 재정의할 수도 있습니다: JVM

```
com.ibm.mq.cfg.Channels.Put1DefaultAlwaysSync=Y
```

트랜잭션 세션에서 송신된 메시지의 경우, 애플리케이션은 최종적으로 commit() 호출 시에 큐 관리자가 메시지를 안전하게 수신했는지 여부를 판별합니다.

애플리케이션이 트랜잭션 세션 내에서 지속 메시지를 송신하며 하나 이상의 메시지를 안전하게 수신하지 않은 경우에는 트랜잭션이 커밋에 실패하며 예외가 생성됩니다. 그러나 애플리케이션이 트랜잭션 세션 내에서 비지속 메시지를 송신하며 하나 이상의 메시지를 안전하게 수신하지 않은 경우에는 트랜잭션이 성공적으로 커밋됩니다. 애플리케이션은 비지속 메시지가 안전하게 도착하지 않았다는 피드백을 수신하지 않습니다.

트랜잭션되지 않은 세션에서 송신된 비지속 메시지의 경우, *ConnectionFactory* 오브젝트의 *SENDCHECKCOUNT* 특성은 IBM MQ classes for JMS에서 큐 관리자가 메시지를 안전하게 수신했는지를 확인하기 전에 송신되는 메시지의 수를 지정합니다.

확인을 통해 하나 이상의 메시지가 안전하게 수신되지 않았으며 애플리케이션이 연결에서 예외 리스너를 등록했음을 발견하는 경우, IBM MQ classes for JMS는 예외 리스너의 *onException()* 메소드를 호출하여 JMS 예외를 애플리케이션에 전달합니다.

JMS 예외는 JMSWMQ0028의 오류 코드를 보유하며 이 코드는 다음 메시지를 표시합니다.

```
At least one asynchronous put message failed or gave a warning.
```

JMS 예외에는 추가 세부사항을 제공하는 링크된 예외도 있습니다. *SENDCHECKCOUNT* 특성의 기본값은 0이며, 이는 해당 확인이 이루어지지 않음을 의미합니다.

이 최적화는 클라이언트 모드로 큐 관리자에 연결하며 빠른 속도로 연속해서 메시지 시퀀스를 송신해야 하는 애플리케이션에 가장 유용하지만, 송신된 각 메시지에 대해 큐 관리자의 즉시 피드백이 필요하지 않습니다. 그러나 애플리케이션은 바인딩 모드로 큐 관리자에 연결된 경우에도 계속해서 이 최적화를 사용할 수 있습니다. 단, 예상되는 성능상의 장점은 그다지 크지 않습니다.

참고: 미확인 ID를 사용하여 **MessageProducer** 트랜잭션 하위에서 메시지를 전송하는 경우, 기본적으로 메시지는 비동기적 삽입 메커니즘을 통해 큐에 배치됩니다.

이는 API가 JMS 다음 구문을 사용하여 대상(Destination)을 **MessageProducer** 지정하지 않고도 생성할 수 있도록 허용하기 때문에 발생할 수 있습니다:

```
javax.jms.MessageProducer messageProducer = javax.jms.Session.createProducer(null);
messageProducer.send(Destination destination, Message message, int deliveryMode, int priority, long timeToLive);
```

이 시나리오에서는 메시지가 전송될 때 목적지가 JMS 제공되며, 메시지 객체가 **MessageProducer** 생성될 때 미리 제공되는 것이 아닙니다. API IBM MQ 측면에서 이는 메시지를 큐에 넣기 위해 MQPUT1가 발행되는 결과를 초래합니다.

이 작업을 동기화 지점 IBM MQ (syncpoint) 아래에서 수행하는 경우, 즉 (자바스전 용어로 JMS) 트랜잭션 아래에 메시지를 배치하는 경우(트랜잭션 JMS 세션 사용 또는 API를 통해 XASession IBM MQ classes for JMS)는 비동기적 삽입(asynchronous put)으로 전환됩니다.

*IBM MQ classes for JMS*에서 미리 읽기 사용

IBM MQ에서 제공하는 미리 읽기 기능을 사용하면 애플리케이션에서 요청하기 전에 트랜잭션 외부에서 수신된 비지속 메시지가 IBM MQ classes for JMS에 송신될 수 있습니다. IBM MQ classes for JMS는 내부 버퍼에 메시지를 저장하며, 애플리케이션에서 요청할 때 애플리케이션에 메시지를 전달합니다.

MessageConsumers 또는 **MessageListeners**를 사용하여 트랜잭션 외부의 대상에서 메시지를 수신하는 IBM MQ classes for JMS 애플리케이션은 미리 읽기 기능을 사용할 수 있습니다. 미리 읽기를 사용하면 이러한 오브젝트를 사용하는 애플리케이션이 메시지를 수신할 때 성능 개선의 장점을 얻을 수 있습니다.

MessageConsumers 또는 **MessageListeners**를 사용하는 애플리케이션이 미리 읽기를 사용할 수 있는지 여부는 다음 특성에 달려 있습니다.

JMS 목적지 특성 READAHEADALLOWED(단축 이름 - RAALD).

JMS 목적지가 표시하는 기본 큐 또는 토픽이 이 옵션을 허용하는 경우, READAHEADALLOWED는 트랜잭션 외부에서 비지속 메시지를 가져오거나 찾아볼 때 JMS 애플리케이션이 미리 읽기를 사용할 수 있는지 여부를 제어합니다.

IBM MQ 큐 또는 토픽 특성 DEFREADA(기본 미리 읽기).

DEFREADA는 트랜잭션 외부에서 비지속 메시지를 수신하거나 찾아보는 애플리케이션이 미리 읽기를 사용할 수 있는지 여부를 지정합니다.

다음 표에서는 READAHEADALLOWED 및 DEFREADA 특성의 가능한 값과 미리 읽기 기능을 사용하는 데 필요한 값 조합을 보여줍니다.

표 47. 트랜잭션 외부에서 비지속 메시지를 수신하거나 찾아볼 때 미리 읽기를 사용할지 여부를 판별하기 위해 *READAHEADALLOWED* 및 *DEFREADA* 특성을 조합하는 방법

IBM MQ 큐 특성	READAHEADALLOWED = YES	READAHEADALLOWED = NO	AS_DEST 또는 AS_Q_DEF 또는 AS_T_DEF
DEFREADA = NO	미리 읽기 기능 사용	미리 읽기 기능 사용 안함	미리 읽기 기능 사용 안함
DEFREADA = YES	미리 읽기 기능 사용	미리 읽기 기능 사용 안함	미리 읽기 기능 사용
DEFREADA = DISABLED	미리 읽기 기능 사용 안함	미리 읽기 기능 사용 안함	미리 읽기 기능 사용 안함

미리 읽기 기능을 사용하는 경우, 애플리케이션이 `MessageConsumer` 또는 `MessageListener`를 작성할 때 IBM MQ classes for JMS는 `MessageConsumer` 또는 `MessageListener`가 모니터링 중인 목적지의 내부 버퍼를 작성합니다. 각 `MessageConsumer` 또는 `MessageListener`마다 하나의 내부 버퍼가 있습니다. 큐 관리자는 애플리케이션이 다음 메소드 중 하나를 호출할 때 IBM MQ classes for JMS에 비지속 메시지의 송신을 시작합니다.

- `MessageConsumer.receive()`
- `MessageConsumer.receive(long timeout)`
- `MessageConsumer.receiveNowait()`
- `Session.setMessageListener(MessageListener listener)`

IBM MQ classes for JMS는 애플리케이션이 작성한 메소드 호출을 통해 첫 메시지를 다시 애플리케이션에 자동으로 리턴합니다. 기타 비지속 메시지는 목적지에 대해 작성된 내부 버퍼에서 IBM MQ classes for JMS에 의해 저장됩니다. 애플리케이션이 다음 메시지의 처리를 요청하는 경우, IBM MQ classes for JMS는 내부 버퍼에서 다음 메시지를 리턴합니다.

IBM MQ classes for JMS는 내부 버퍼가 비어 있을 때 큐 관리자로부터 비지속 메시지를 추가로 요청합니다.

IBM MQ classes for JMS에 의해 사용되는 내부 버퍼는 `MessageListener`가 연관된 JMS 세션 또는 `MessageConsumer`를 애플리케이션이 닫을 때 삭제됩니다.

`MessageConsumers`의 경우, 내부 버퍼의 미처리 메시지는 유실됩니다.

`MessageListeners`를 사용하는 경우, 내부 버퍼의 메시지에 발생하는 상황은 JMS 목적지 특성 *READAHEADCLOSEPOLICY*(단축 이름 - *RACP*)에 달려 있습니다. 특성의 기본값은 *DELIVER_ALL*입니다. 이는 내부 버퍼의 모든 메시지가 애플리케이션에 전달될 때까지 `MessageListener`를 작성하는 데 사용된 JMS 세션이 닫히지 않음을 의미합니다. 특성이 *DELIVER_CURRENT*로 설정된 경우, JMS 세션은 현재 메시지가 애플리케이션에 의해 처리되었으며 내부 버퍼의 모든 잔여 메시지가 제거된 후에 닫힙니다.

IBM MQ classes for JMS의 보유된 발행

IBM MQ classes for JMS 클라이언트는 보유된 발행을 사용하도록 구성될 수 있습니다.

발행자는 토픽에서 관심을 등록하는 향후 구독자에게 송신이 가능하도록 발행의 사본을 보유해야 하는지를 지정할 수 있습니다. 이 작업은 정수 특성 *JMS_IBM_RETAIN*을 값 1로 설정하여 IBM MQ classes for JMS에서 수행됩니다. `com.ibm.msg.client.jms.JmsConstants` 인터페이스에서 이러한 값에 대해 상수가 정의되었습니다. 예를 들어, 메시지 *msg*를 작성한 경우에 이를 보유된 발행으로 설정하려면 다음 코드를 사용하십시오.

```
// set as a retained publication
msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION);
```

이제 메시지를 정상으로 송신할 수 있습니다. *JMS_IBM_RETAIN*를 수신된 메시지에서 조회할 수도 있습니다. 따라서 수신된 메시지가 보유된 발행인지 여부를 조회할 수 있습니다.

애플리케이션에서 JMS 연결 유지 동작 JMS 사용자 지정

큐 매니저가 채널 연결 해제 간격(**DISCINT**)을 강제 적용할 때 발생할 수 있는 예기치 않은 연결 해제를 방지하기 위해 IBM MQ 10.0.0, 연결의 JMS 연결 유지 동작을 사용자 정의할 수 있습니다.

사용 시 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging:

- 응용 프로그램에 의해 생성된 모든 연결 JMS 또는 Jakarta Messaging 연결은 큐 관리자에 대한 연결 핸들을 가집니다.
- 연결에서 생성된 각 세션 JMS Jakarta Messaging 또는 각 세션도 자체 연결 핸들을 가집니다.
- 애플리케이션이 생성한 모든 JMSContext에는 두 개의 연결 핸들이 있습니다. JMSContexts는 연결(Jakarta Messaging connection) JMS 또는 세션(session)을 캡슐화하므로, 연결 핸들 중 하나는 연결용이고 다른 하나는 세션용입니다.

애플리케이션과 큐 관리자 간의 모든 통신은 또는 Jakarta Messaging JMS 세션과 연관된 연결 핸들을 통해 이루어집니다. 연결 핸들에서 또는 Jakarta Messaging JMS 연결에 대한 활동이 거의 없습니다.

채널의 속성이 **DISCINT** 0보다 큰 값으로 설정된 상태에서 또는 Jakarta Messaging JMS 연결이 큐 관리자에 연결된 경우, 큐 관리자는 활동 부족으로 인해 해당 채널을 분리할 수 있습니다. 이로 인해 애플리케이션이 예기치 않게 이유 코드 2009(IBM MQ MQRC_CONNECTION_BROKEN)을 포함하는 JMSExceptions를 보고하게 됩니다.

부터 IBM MQ 10.0.0, 또는 Jakarta Messaging JMS 연결에서 사용되는 채널 인스턴스가 열린 상태로 유지되고로 인해 달하지 않도록 보장하는 **DISCINT** 새로운 메커니즘이 제공됩니다. 이는 애플리케이션의 안정성을 향상시킵니다.

기본 동작

기본적으로 및 IBM MQ classes for JMS 은 각 JMS IBM MQ classes for Jakarta Messaging 연결에 대해 주기적으로 MQSTAT API 호출을 발행하여 모든 연관된 JMS 세션을 활성 상태로 유지합니다.

- 간격은 채널 속성에 의해 **HBINT** 결정됩니다.
- 만약 **HBINT**=0, 기본 간격으로 300초가 사용됩니다.

이 유지 흐름은 큐 관리자가 채널 연결 해제 간격(**DISCINT**)을 강제할 때 발생할 수 있는 예기치 않은 연결 해제를 방지하는 데 도움이 됩니다.

간격 맞춤 설정

기본 간격을 재정의하려면 다음 Java 시스템 속성을 설정하여 애플리케이션을 실행하십시오:

```
-Dcom.ibm.mq.jms.tuning.connectionKeepAlive=<value>
```

여기서 <value> 는 다음과 같이 설정됩니다:

- 필요한 차단 간격을 초 단위로 정의하는 양의 정수.
- 권장: 채널 **DISCINT** 값의 절반으로 설정하십시오.
- 애플리케이션이 서로 다른 **DISCINT** 값을 가진 여러 채널을 통해 큐 관리자에 연결하는 경우, 해당 채널들에 설정된 값 중 가장 작은 **DISCINT** 값을 확인하고 해당 값의 정확히 절반으로 속성을 설정하십시오.
- 값이 -1 이면 연결 유지 흐름이 JMS 비활성화됩니다.
- **HBINT** 0 또는 -1을 제외한 다른 음수 값은 무시되며, 구간은 에서 도출됩니다.

적용성

이 기능은(및 이후 IBM MQ 10.0.0 버전)에서만 IBM MQ classes for JMS 사용할 수 있으며, 또는 이후 IBM MQ 9.0 버전의 IBM MQ classes for Jakarta Messaging 큐 관리자에 연결할 때만 적용됩니다.

생존 확인 호출 검증

검증 또는 문제 해결을 위해 다음 Java 시스템 속성을 설정하여 애플리케이션을 실행하면 MQSTAT API 호출의 로깅을 활성화할 수 있습니다:

```
-Dcom.ibm.mq.jms.tuning.connectionKeepAlive.logMQSTAT=true
```

- true로 설정하면 각 MQSTAT 호출이 파일에 mqjms.log 기록됩니다.

- 기본값은 false 이며, 이는 로깅을 비활성화합니다.

다음 예제는 파일에 mqjms.log 기록되는 로그 메시지를 보여줍니다:

```
2026년 8월 25일 오후 10:11:54 IST[ JmsConnectionKeepAliveThread [connection
name=9.30.51.244(1414 ),queue manager=QM1,channel name=DEV.APP.SVRCONN,HBINT=40, MQ Build
Level=9.4.4.0(p944-jdsc-L250805 )]] com.ibm.mq.jmqi.remote.api.JmsConnectionKeepAliveThread
MQ 연결 ID '414D5143514D3120202020202020A00CAC68B00300F0' 를 가진 JMS 연결에 대해 ' QM1 ' 큐 관리자
로 전송된 JMS 연결 유지 흐름.
```

설명:

모든 JMS 세션이 활성 상태로 유지되도록 하기 위해 JMS 연결에 대해 지정된 큐 관리자에 MQSTAT API 호출 형식의 JMS 연결 유지 플로우가 전송되었습니다. JMS 연결에 대한 다음 키프얼라이브 흐름은 '40'초 후에 전송될 수 있습니다.

ACTION:

없음

IBM MQ classes for JMS에서 XA 지원

JMS는 JEE 컨테이너 내에서 지원되는 트랜잭션 관리자에서 바인딩 및 클라이언트 모드로 XA-호환 트랜잭션을 지원합니다.

애플리케이션 서버 환경에서 XA 기능이 필요하면 애플리케이션을 적절하게 구성해야 합니다. 분배 트랜잭션을 사용하도록 애플리케이션을 구성하는 방법에 대한 정보는 애플리케이션 서버의 자체 문서를 참조하십시오.

IBM MQ 큐 관리자는 JMS의 트랜잭션 관리자로서 작동할 수 없습니다.

메시지 JMS 전달 지연

버전 JMS 2.0 1.0.0 이상에서는 메시지 전송 시 전달 지연을 지정할 수 있습니다. 지정된 전달 지연이 경과된 이 후가 아니면 큐 관리자가 메시지를 전달하지 않습니다.

응용 프로그램은 메시지를 보낼 때 또는 `MessageProducer.setDeliveryDelay(long deliveryDelay)` `JMSProducer.setDeliveryDelay(long deliveryDelay)` 을 사용하여 밀리초 단위의 전달 지연을 지정할 수 있습니다. 이 값은 메시지가 송신된 시간에 추가되며 기타 애플리케이션이 해당 메시지를 가져올 수 있는 가장 빠른 시간을 제공합니다.

전달 지연은 단일 내부 스테이징 큐를 사용하여 구현됩니다. 0이 아닌 전달 지연을 보유한 메시지는 대상 큐에 대한 정보와 전달 지연을 표시하는 헤더와 함께 이 큐에 놓입니다. 전달 지연 프로세서라고 하는 큐 관리자의 컴포넌트는 스테이징 큐의 메시지를 모니터링합니다. 메시지의 전달 지연이 완료되면 메시지가 스테이징 큐를 떠나며 대상 큐에 놓여집니다.

메시징 클라이언트

전달 지연의 IBM MQ 구현은 JMS 클라이언트를 사용 중일 때만 사용 가능합니다. IBM MQ에서 전달 지연을 사용 중이면 다음 제한사항이 적용됩니다. 이러한 제한 사항은 와 `MessageProducers` 에 동등하게 `JMSProducers` 적용되지만, 의 `JMSRuntimeExceptions` 경우 예외가 `JMSProducers` 발생합니다.

- 큐 매니저에 연결된 시점이 보다 이른 IBM MQ 8.0 경우, 를 0이 아닌 `MessageProducer.setDeliveryDelay` 값으로 호출하려는 모든 시도는 `MQRC_FUNCTION_NOT_SUPPORTED` 메시지와 함께 `JMSException` 를 반환합니다.
- `MQBND_BIND_NOT_FIXED` 이외의 `DEFBIND` 값을 갖는 클러스터 목적지에는 전달 지연이 지원되지 않습니다. `MessageProducer`에 0이 아닌 전달 지연이 설정되어 있으며 이 요구사항을 충족하지 않는 목적지에 송신을 시도한 경우, 호출 결과로 `MQRC_OPTIONS_ERROR` 메시지와 함께 `JMSException`이 발생합니다.
- 이전에 지정된 0이 아닌 전달 지연 미만인 잔존 시간 값의 설정을 시도(또는 그 역으로 시도)하면 `MQRC_EXPIRY_ERROR` 메시지와 함께 `JMSException`이 발생합니다. 이 확인은 선택된 정확한 조작 세트에 따라 `setTimeToLive` 또는 `setDeliveryDelay` 또는 `send` 메소드의 호출에서 수행됩니다.
- 보유된 발행 및 전달 지연의 사용은 지원되지 않습니다. 해당 메시지가 `msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION)` 을 사용하여 보유됨으로 표시된 경우 전달 지연으로 메시지를 발행하려는 시도하면 `MQRC_OPTIONS_ERROR` 메시지와 함께 `JMSException`이 발생합니다.
- 배달 지연 및 메시지 그룹화는 지원되지 않으며, 이 조합을 사용하려는 시도는 `MQRC_OPTIONS_ERROR` 메시지와 함께 `JMSException` 실패합니다.

전달 지연으로 메시지를 송신하는 데 실패하면 클라이언트에서 적합한 오류 메시지(예: 큐가 가득 참)와 함께 `JMSEException`을 전달합니다. 일부 상황에서, 오류 메시지는 대상 목적지 또는 스테이징 큐 또는 둘 모두에 적용될 수 있습니다.

참고: IBM MQ는 작업 단위가 커밋되지 않은 경우에도 작업 단위로 메시지를 넣는 애플리케이션이 동일한 메시지를 다시 가져올 수 있도록 허용합니다. 작업 단위가 커밋될 때까지 메시지가 스테이징 큐에 놓이지 않으며 따라서 대상 목적지에 송신되지 않으므로, 이 기술은 전달 지연에서 작동하지 않습니다.

권한 부여

IBM MQ는 애플리케이션이 0이 아닌 전달 지연으로 메시지를 송신할 때 원래 대상 목적지에서 권한 검사를 수행합니다. 애플리케이션이 권한 부여되지 않으면 송신에 실패합니다. 메시지의 전달 지연이 완료되었음을 감지하는 경우, 큐 관리자는 대상 큐를 엽니다. 권한 검사는 이 시점에는 수행되지 않습니다.

SYSTEM.DDELAY.LOCAL.QUEUE

시스템 큐, `SYSTEM.DDELAY.LOCAL.QUEUE`가 전달 지연을 구현하는 데 사용됩니다.

- ▶ **Multi** 다중 플랫폼에는 기본적으로 `SYSTEM.DDELAY.LOCAL.QUEUE`가 있습니다. 해당 `MAXMSGL` 및 `MAXDEPTH` 속성이 예상된 로드에서 충분하도록 시스템 큐가 대체되어야 합니다.
- ▶ **z/OS** IBM MQ for z/OS에서, `SYSTEM.DDELAY.LOCAL.QUEUE`는 로컬 및 공유 큐 모두에 전달 지연으로 송신된 메시지의 스테이징 큐로서 사용됩니다. z/OS에서는 큐가 작성되어야 하며 해당 `MAXMSGL` 및 `MAXDEPTH` 속성이 예상 로드에서 충분할 수 있도록 정의되어야 합니다.

이 큐가 작성될 때 이는 가급적 적은 수의 사용자가 이에 액세스할 수 있도록 보호되어야 합니다. 큐에 대한 액세스는 유지보수 및 모니터링 용도로만 사용되어야 합니다.

0이 아닌 전달 지연으로 JMS 애플리케이션에 의해 메시지가 송신되는 경우, 이는 새 메시지 ID로 이 큐에 놓여집니다. 원래 메시지 ID는 메시지의 상관 ID에 놓여집니다. 이 상관 ID는 애플리케이션이 필요 시에 스테이징 큐에서 메시지를 검색할 수 있도록 허용합니다(예: 대규모 전달 지연이 실수로 사용된 경우).

z/OS에 대한 고려사항

▶ z/OS

시스템이 z/OS에서 실행 중인 경우, 전달 지연을 사용하려면 고려해야 할 추가 고려사항이 있습니다.

전달 지연이 사용되는 경우에는 시스템 큐 `SYSTEM.DDELAY.LOCAL.QUEUE`가 정의되어야 합니다. 이는 예상된 로드에서 충분하며 `INDXTYPE(NONE)` 및 `MSGDLVSQ(FIFO)`가 지정된 스토리지 클래스로 정의되어야 합니다. 시스템 큐의 샘플 정의가 `CSQ4INSG JCL`에서 주석 처리되어 제공됩니다.

공유 큐

전달 지연은 메시지를 공유 큐에 송신하기 위해 지원됩니다. 그러나 대상 큐가 공유되는지 여부와는 무관하게 사용되는 하나의 개인용 스테이징 큐만 존재합니다. 해당 개인 큐를 소유하는 큐 관리자는 지연이 완료될 때 지연 메시지를 대상 공유 큐에 송신할 수 있도록 실행 중이어야 합니다.

참고: 비지속 메시지를 전달 지연으로 공유 큐에 넣고 스테이징 큐를 소유하는 큐 관리자가 종료되는 경우에는 원래 메시지가 유실됩니다. 따라서 공유 큐에 전달 지연으로 송신된 비지속 메시지는 공유 큐에 전달 지연 없이 송신된 비지속 메시지보다 유실될 가능성이 높습니다.

대상 목적지 분석

메시지가 큐에 송신된 경우, 분석은 두 번 구동됩니다. 즉, 스테이징 큐에서 메시지를 꺼내와서 이를 대상 큐에 송신할 때 한 번은 JMS 애플리케이션에 의해 구동되며 다른 한 번은 큐 관리자에 의해 구동됩니다.

JMS 애플리케이션이 송신 메소드를 호출할 때 구독에 대한 대상 구독은 일치됩니다.

메시지가 큐 정의에 따라 지속성 및 우선순위로 송신되는 경우, 값은 첫 번째 분석에서 설정되며 두 번째에서는 설정되지 않습니다.

만기 간격

전달 지연은 만료 속성의 동작을 보존합니다. **MQMD.Expiry** 예를 들어, 만료 간격 20,000ms 및 지연 간격 5,000ms로 메시지를 JMS 애플리케이션에서 넣고 10,000ms의 경과 시간 이후 가져오는 경우, **MQMD.expiry** 필드의 값이 약 5초일 수 있습니다. 이 값은 메시지를 넣은 시간에서 이를 가져온 시간까지 15초가 경과되었음을 표시합니다.

스테이징 큐에 있는 동안 메시지가 만료되고 **MQRO_EXPIRATION_*** 옵션 중 하나가 설정된 경우, 생성된 보고서는 애플리케이션이 송신한 원래 메시지용이며 전달 지연 정보를 포함하기 위해 사용된 헤더는 제거됩니다.

전달 지연 프로세서 중지 및 시작

z/OS z/OS에서 전달 지연 프로세서는 큐 관리자 **MSTR** 주소 공간에 통합됩니다. 큐 관리자가 시작될 때 전달 지연 프로세서도 시작됩니다. 스테이징 큐가 사용 가능한 경우, 이는 큐를 열고 처리를 위해 메시지가 이에 도착할 때까지 대기합니다. 스테이징 큐가 정의되지 않았거나 가져오기에 사용할 수 없거나 다른 오류가 발생하는 경우에는 전달 지연 프로세서가 종료됩니다. 스테이징 큐가 나중에 정의되었거나 가져오기를 사용하도록 변경된 경우에는 전달 지연 프로세서가 다시 시작됩니다. 배송 지연 프로세서가 다른 이유로 종료된 경우, 스테이징 큐의 속성을 **PUT ENABLED**에서 **DISABLED**로 변경한 후 다시 **ENABLED**로 되돌리면 재시작할 수 있습니다. 어떤 이유론든 전달 지연 프로세서를 중지해야 하는 경우에는 스테이징 큐의 **PUT** 속성을 **DISABLED**로 설정하십시오.

Multi 다중 플랫폼에서는 지연 프로세서가 큐 관리자와 함께 시작되며, 복구 가능한 장애가 발생하는 경우 자동으로 다시 시작됩니다.

대상 큐에 넣기 실패

일단 지연이 완료될 때 지연 메시지를 대상 큐에 넣을 수 없는 경우, 메시지는 보고서 옵션에 표시된 대로 처리됩니다. 이는 버려지거나 데드-레터 큐에 송신됩니다. 이 조치가 실패하는 경우에는 나중에 메시지 넣기가 시도됩니다. 조치가 성공하는 경우에는 예외 보고서가 생성되며 지정된 큐에 송신됩니다(보고서가 요청된 경우). 보고 메시지를 송신할 수 없는 경우, 보고 메시지가 데드-레터 큐에 송신됩니다. 데드-레터 큐에 보고서를 송신할 수 없으며 메시지가 지속인 경우에는 모든 변경사항이 버려지며 원래 메시지는 롤백되고 나중에 다시 전달됩니다. 메시지가 비지속인 경우, 보고 메시지는 버리지만 기타 변경사항은 커밋됩니다. 구독자가 구독을 취소하여 지연된 발행을 전달할 수 없거나 연결이 끊어져서 지속 가능하지 않은 구독자인 경우에는 메시지가 자동으로 버려집니다. 앞에서 설명한 대로 보고 메시지는 계속 생성됩니다.

지연된 발행을 구독자에게 전달할 수 없으며 대신에 이를 데드-레터 큐에 넣지만 데드-레터 큐에 넣기가 실패하는 경우에는 메시지가 버려집니다.

전달 지연이 완료된 후에 대상 큐에 넣기가 실패하는 가능성을 줄이기 위해, 큐 관리자는 **JMS** 클라이언트가 0이 아닌 전달 지연으로 메시지를 송신할 때 일부 기본 확인을 수행합니다. 이러한 확인에는 큐를 사용 불가능으로 넣는지, 메시지가 허용된 최대 메시지 길이를 초과하는지 및 큐가 가득 차 있는지 여부가 포함됩니다.

발행/구독

사용 가능한 구독에 대한 발행의 일치는 **JMS** 애플리케이션이 0이 아닌 전달 지연으로 메시지를 송신할 때 발생합니다. 각 일치하는 구독자에 대한 메시지는 **SYSTEM.DELAY.LOCAL.QUEUE** 큐에 놓이며, 여기서 이는 전달 지연이 완료될 때까지 유지됩니다. 해당 구독자 중 하나가 다른 큐 관리자에 대한 프록시 구독인 경우, 해당 큐 관리자의 팬아웃은 전달 지연이 완료된 이후에 발생합니다. 그 결과로 인해 다른 큐 관리자의 구독자는 구독하기 전에 원래 발행된 발행물을 수신할 수 있습니다. 이는 또는 이후 **JMS 2.0** 사양과의 차이점입니다.

발행/구독의 전달 지연은 대상 토픽이 **(N)PMSGDLV = ALLAVAIL**로 구성된 경우에만 지원됩니다. 기타 값을 사용하려고 시도하면 **MQRC_PUBLICATION_FAILURE** 오류가 발생합니다. 대상 큐에 메시지를 넣는 동안에 전달 지연 프로세서가 실패하는 경우, 결과는 "대상 큐에 넣기 실패" 절에 설명한 대로입니다.

보고 메시지

무시되지만 대상 큐에 송신될 때 메시지에서 전달되는 다음 옵션을 제외하면 모든 보고서 옵션은 전달 프로세서에 의해 지원되고 처리됩니다.

- **MQRO_COA***

- MQRO_COD*
- MQRO_PAN/MQRO_NAN
- MQRO_ACTIVITY

복제 및 공유된 구독

여러 소비자가 동일한 구독에 접근할 수 있도록 하는 방법은 두 가지가 있습니다. 이러한 두 개의 메소드는 복제된 구독을 사용하거나 공유된 구독을 사용하여 실행됩니다.

복제된 구독

복제된 구독은 IBM MQ 확장입니다. 복제된 구독은 상이한 JVM(Java virtual machines)의 다중 이용자에게 구독에 대한 동시 액세스를 허용합니다. 이 작동은 `connectionFactory` 오브젝트에서 **CLONESUPP** 특성을 사용하여 설정하여 사용될 수 있습니다. 기본적으로 **CLONESUPP**는 사용 안함입니다. 복제된 구독은 지속 가능 구독에서만 사용됩니다. **CLONESUPP**가 사용 가능한 경우에는 이 `connectionFactory`를 사용하여 작성된 각각의 후속 연결이 복제됩니다.

하나 이상의 이용자가 해당 구독에서 메시지를 수신하기 위해 작성된 경우(즉, 동일한 구독 이름을 지정하여 작성됨) 지속 가능한 구독은 복제되었다고 간주될 수 있습니다. 이는 이용자가 작성된 연결의 **CLONESUPP**이 `MQConnectionFactory`에서 사용으로 설정된 경우에만 수행될 수 있습니다. 메시지가 구독의 토픽에서 발행되면 해당 메시지의 사본이 구독에 송신됩니다. 임의의 이용자가 메시지를 사용할 수 있지만 오직 하나만 이를 수신합니다.

참고: 복제된 구독을 사용하면 JMS 스펙이 확장됩니다.

공유된 구독

사양에서 JMS 2.0 도입된 공유 구독은 하나의 주제 구독에서 발생하는 메시지를 여러 소비자 간에 공유할 수 있도록 합니다. 구독의 각 메시지는 해당 구독의 이용자 중 하나에만 전달됩니다. 공유 구독은 또는 이후 버전의 JMS 2.0 API에 대한 관련 호출을 통해 활성화됩니다.

API는 다음 방법 중 하나로 호출될 수 있습니다.

- Java SE 애플리케이션(또는 Java EE Client Container)에서.
- 서블릿 또는 MDB의 구현에서.

또는 이후 JMS 2.0 사양은 공유 구독에서 MDB를 구동하는 표준 방식을 정의하지 않으므로, 이를 위해 활성화 IBM MQ 사양 **sharedSubscription** 속성을 제공합니다. 이 특성에 대한 자세한 정보는 409 페이지의 『인바운드 통신용 자원 어댑터 구성』 및 423 페이지의 『**sharedSubscription** 특성을 정의하는 방법의 예』의 내용을 참조하십시오.

공유된 구독이 사용되면 이의 공유를 취소할 수 없습니다.

공유된 구독은 지속 가능 또는 지속 불가능 구독으로서 작성될 수 있습니다. 큐 관리자 측에서는 일반적인 JMS 구성 외에 별도로 객체를 생성할 필요가 없습니다. 필요한 모든 객체는 동적으로 생성됩니다.

공유 또는 복제된 구독 간에 결정

공유 구독과 복제 구독 중 어떤 것을 사용할지 결정할 때는 양쪽의 장점을 고려하십시오. 가능한 경우, 특정 IBM MQ 확장이 아닌 사양 정의된 동작인 공유 구독을 사용하십시오.

다음 표에는 공유 및 복제된 구독 간에 결정할 때 고려할 일부 관점이 포함되어 있습니다.

표 48. 공유 구독과 복제 구독에 대한 고려 사항 비교	
공유된 구독	복제된 구독
공유 구독은 또는 이후 JMS 2.0 사양의 표준 구성 요소입니다.	복제된 구독은 IBM MQ 특정 확장입니다.
공유된 구독은 명시적 API 메소드 호출을 사용하여 작성됩니다.	복제된 구독은 <code>ConnectionFactory</code> 레벨에서 관리적으로 제어됩니다.

표 48. 공유 구독과 복제 구독에 대한 고려 사항 비교 (계속)	
공유된 구독	복제된 구독
공유된 구독은 지속 가능 또는 지속 불가능일 수 있습니다.	복제된 구독은 지속 가능만 가능합니다.
공유된 구독은 개별 구독 기반으로 명시적으로 작성됩니다.	복제된 구독은 기능이 사용되는 연결 하의 지속 가능한 구독에 사용됩니다.
구독이 공유됨으로 작성된 경우 이는 나중에 공유되지 않음으로 변경될 수 있으며, 그 반대의 경우도 가능합니다.	소유 중인 연결의 CLONESUPP 특성이 변경된 경우, 구독은 다시 열릴 때마다 복제됨에서 복제되지 않음으로 변경될 수 있습니다.

관련 개념

[구독자 및 구독](#)

[구독 지속성](#)

관련 태스크

[JMS 2.0 공유 구독 사용](#)

관련 참조

423 페이지의 『[sharedSubscription 특성을 정의하는 방법의 예](#)』

WebSphere Liberty server.xml 파일 내에서 활성화 스펙의 sharedSubscription 특성을 정의할 수 있습니다. 또는 어노테이션을 사용하여 메시지 구독 Bean(MDB) 내에 특성을 정의할 수 있습니다.

[CLONESUPP](#)

SupportMQExtensions 특성

해당 JMS 2.0 사양은 특정 동작 방식에 변경 사항을 도입했습니다. IBM MQ 8.0 그리고 이후에는 속성을 포함하는데 **com.ibm.mq.jms.SupportMQExtensions**, 이 속성을 TRUE로 설정하면 변경된 동작들을 이전 구현 방식으로 되돌릴 수 있습니다.

▶ **JM 3.0** 해당 **com.ibm.mq.jakarta.jms.SupportMQExtensions** 속성(Jakarta Messaging 3.0)은 에서 사용할 수 있는 IBM MQ classes for Jakarta Messaging에 com.ibm.mq.jakarta.client.jar 의해 지원됩니다.

▶ **JMS 2.0** (JMS 2.0) **com.ibm.mq.jms.SupportMQExtensions** 속성은 또는 com.ibm.mq.allclient.jar 에서 사용할 수 있는 com.ibm.mqjms.jar 에 의해 IBM MQ classes for JMS지원됩니다.

세 개의 기능 영역을 **SupportMQExtensions**를 True로 설정하여 되돌립니다.

메시지 우선순위

메시지에는 0에서 9까지의 우선순위가 지정될 수 있습니다. 이전에는 메시지가 `JMS 2.0-1` 값을 사용할 수도 있었는데, 이는 큐의 기본 우선순위가 사용됨을 나타냅니다. JMS 2.0 그리고 이후에는 메시지 우선순위가 '-1'로 설정되는 것을 허용하지 않습니다. 전원 켜기 시 **SupportMQExtensions-1**의 값을 사용할 수 있습니다.

클라이언트 ID

버전 JMS 2.0 1.0 이상 사양에서는 연결 시 비어 있지 않은 클라이언트 ID의 고유성을 확인해야 합니다. 이 **SupportMQExtensions** 옵션을 켜면 해당 요구사항이 무시되며, 클라이언트 ID를 재사용할 수 있습니다.

NoLocal

이 상수가 활성화된 경우, 소비자 JMS 2.0 (consumer)는 동일한 클라이언트 ID(client ID)가 게시한 메시지를 수신할 수 없어야 합니다. JMS 2.0 이전에는 자체 연결에 의해 발행되는 메시지의 수신을 방지할 수 있도록 이 속성이 구독자에서 설정되었습니다. **SupportMQExtensions**를 켜면 이 작동이 이전 구현으로 되돌아갑니다.

이 특성은 다음과 같이 설정될 수 있습니다.

```
java -Dcom.ibm.mq.jms.SupportMQExtensions=true
```

이 특성은 **java** 명령에서 표준 JVM 시스템 특성으로 설정되거나 IBM MQ classes for JMS 구성 파일 내에 포함될 수 있습니다.

관련 개념

98 페이지의 『[Jakarta Messaging 구성 IBM MQ classes for JMS 파일](#)』

구성 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 파일은 구성에 사용되는 속성을 IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 지정합니다.

관련 참조

105 페이지의 『[JMS 클라이언트 동작을 구성하는 데 사용된 특성](#)』

JMS 클라이언트의 동작을 구성하려면 이러한 특성을 사용하십시오.

애플리케이션에서 JMS 공유 구독 사용하기

공유 구독에서는 단일 구독이 여러 소비자들 사이에서 공유되며, 특정 시점에는 오직 하나의 소비자만이 간행물을 수신합니다.

공유 구독은부터 JMS 2.0 이용 가능합니다. 애플리케이션을 JMS 개발할 때, 특히 버전 10 이상을 대상으로 IBM MQ 8.0 개발할 경우, 이 기능이 큐 매니저에 미치는 영향을 고려해야 할 수 있습니다.

공유 구독의 기본 개념은 여러 소비자가 부담을 분담하는 것입니다. 지속 가능한 구독도 여러 소비자 사이에 공유될 수 있습니다.

예를 들어, 다음을 가정합니다.

- 풋볼 경기 업데이트를 수신하기 위해 FIFA2014/UPDATES 토픽을 구독하는 SUB 구독은 3명의 소비자, C1, C2, C3에서 공유됨
- FIFA2014/UPDATES 토픽에서 생성자 P1이 발행됨

FIFA2014/UPDATES 에 게시물이 게시되면, 해당 게시물은 세 소비자(C1, 또는 C2, 또는 C3) 중 오직 하나만이 수신하게 되며 모두에게 수신되지는 않습니다.

다음 예제는 공유 구독의 사용법을 보여주고, 또한 메시지 본문만 가져오기 위한 JMS 2.0 추가 API의 `Message.receiveBody()` 사용법을 보여줍니다.

샘플에서는 3개의 구독자 스레드를 작성합니다. 그러면 FIFA2014/UPDATES 토픽에 대한 공유된 구독과 하나의 발행자 스레드를 작성합니다.

JM 3.0

```
package mqv91Samples;

import jakarta.jms.JMSEException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import jakarta.jms.JMSContext;
import jakarta.jms.Topic;
import jakarta.jms.Queue;
import jakarta.jms.JMSConsumer;
import jakarta.jms.Message;
import jakarta.jms.JMSProducer;

/*
 * Implements both Subscriber and Publisher
 */
class SharedNonDurableSubscriberAndPublisher implements Runnable {
    private Thread t;
    private String threadName;

    SharedNonDurableSubscriberAndPublisher( String name){
        threadName = name;
        System.out.println("Creating Thread:" + threadName );
    }

    /*
     * Demonstrates shared non-durable subscription in JMS 2.0 and later
     */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
```



```

JMSContext msgContext = null;

try {
    // Create Factory for WMQ JMS provider
    JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
    // Create connection factory
    cf = ff.createConnectionFactory();
    // Set MQ properties
    cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
    cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
    // Create message context
    msgContext = cf.createContext();

    // Create a topic destination
    Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

    // Create a consumer. Subscription name specified, required for sharing of subscription.
    JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

    // Loop around to receive publications
    while(true){

        String msgBody=null;

        // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
        msgBody = msgCons.receiveBody(String.class);

        if(msgBody != null){
            System.out.println(threadName + " : " + msgBody);
        }
    }
} catch (JMSException jmsEx) {
    System.out.println(jmsEx);
}
}

```

JMS 2.0

```

package mqv91Samples;

import javax.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import javax.jms.JMSContext;
import javax.jms.Topic;
import javax.jms.Queue;
import javax.jms.JMSConsumer;
import javax.jms.Message;
import javax.jms.JMSProducer;

/*
 * Implements both Subscriber and Publisher
 */
class SharedNonDurableSubscriberAndPublisher implements Runnable {
    private Thread t;
    private String threadName;

    SharedNonDurableSubscriberAndPublisher( String name){
        threadName = name;
        System.out.println("Creating Thread:" + threadName );
    }

    /*
     * Demonstrates shared non-durable subscription in JMS 2.0 and later
     */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context

```



```

msgContext = cf.createContext();

// Create a topic destination
Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

// Create a consumer. Subscription name specified, required for sharing of subscription.
JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

// Loop around to receive publications
while(true){

    String msgBody=null;

    // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
    msgBody = msgCons.receiveBody(String.class);

    if(msgBody != null){
        System.out.println(threadName + " : " + msgBody);
    }
}
} catch(JMSEException jmsEx){
    System.out.println(jmsEx);
}
}
}

```

```

/*
 * Publisher publishes match updates like current attendance in the stadium, goal score and ball
 possession by teams.
 */
private void matchUpdatePublisher(){
    JmsConnectionFactory cf = null;
    JMSContext msgContext = null;
    int nederlandsGoals = 0;
    int chileGoals = 0;
    int stadiumAttendance = 23231;
    int switchIndex = 0;
    String msgBody = "";
    int nederlandsHolding = 60;
    int chileHolding = 40;

    try {
        // Create Factory for WMQ JMS provider
        JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);

        // Create connection factory
        cf = ff.createConnectionFactory();
        // Set MQ properties
        cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
        cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);

        // Create message context
        msgContext = cf.createContext();

        // Create a topic destination
        Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

        // Create publisher to publish updates from stadium
        JMSProducer msgProducer = msgContext.createProducer();

        while(true){
            // Send match updates
            switch(switchIndex){
                // Attendance
                case 0:
                    msgBody = "Stadium Attendance " + stadiumAttendance;
                    stadiumAttendance += 314;
                    break;

                    // Goals
                case 1:
                    msgBody = "SCORE: The Netherlands: " + nederlandsGoals + " - Chile:" + chileGoals;
                    break;

                    // Ball possession percentage
                case 2:
                    msgBody = "Ball possession: The Netherlands: " + nederlandsHolding + "% - Chile:
" + chileHolding + "%";
                    if((nederlandsHolding > 60) && (nederlandsHolding < 70)){
                        nederlandsHolding -= 2;
                        chileHolding += 2;
                    }
                    else{

```

```

        nederlandsHolding += 2;
        chileHolding -= 2;
    }
    break;
}

// Publish and wait for two seconds to publish next update
msgProducer.send (fifaScores, msgBody);
try{
    Thread.sleep(2000);
}catch(InterruptedException iex){

}

// Increment and reset the index if greater than 2
switchIndex++;
if(switchIndex > 2)
    switchIndex = 0;
}
}catch(JMSEException jmsEx){
    System.out.println(jmsEx);
}
}

/*
 * (non-Javadoc)
 * @see java.lang.Runnable#run()
 */
public void run() {
    // If this is a publisher thread
    if(threadName == "PUBLISHER"){
        matchUpdatePublisher();
    }else{
        // Create subscription and start receiving publications
        sharedNonDurableSubscriptionDemo();
    }
}

// Start thread
public void start (){
    System.out.println("Starting " + threadName );
    if (t == null)
    {
        t = new Thread (this, threadName);
        t.start ();
    }
}
}
}

```

```

/*
 * Demonstrate JMS 2.0 and later simplified API using IBM MQ 91 JMS Implementation
 */
public class Mqv91jms2Sample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Create first subscriber and start
        SharedNonDurableSubscriberAndPublisher subOne = new
        SharedNonDurableSubscriberAndPublisher( "SUB1");
        subOne.start();

        // Create second subscriber and start
        SharedNonDurableSubscriberAndPublisher subTwo = new
        SharedNonDurableSubscriberAndPublisher( "SUB2");
        subTwo.start();

        // Create third subscriber and start
        SharedNonDurableSubscriberAndPublisher subThree = new
        SharedNonDurableSubscriberAndPublisher( "SUB3");
        subThree.start();

        // Create publisher and start
        SharedNonDurableSubscriberAndPublisher publisher = new
        SharedNonDurableSubscriberAndPublisher( "PUBLISHER");
        publisher.start();
    }
}

```

관련 개념

[IBM MQ Java 언어 인터페이스](#)

모듈식 애플리케이션이 또는 를 IBM MQ classes for JMS 사용하도록 구성하기 IBM MQ classes for Jakarta Messaging

적절한 모듈을 애플리케이션 내에서 요구하고, 모듈 경로에 해당 디렉터리를 포함시킴으로써 와 IBM MQ classes for Jakarta Messaging 를 모듈 방식으로 IBM MQ classes for JMS 사용할 수 있습니다.

모듈식 포장

통합된 JAR 파일은 및 에 IBM MQ classes for JMS 대해 자동 IBM MQ classes for Jakarta Messaging 모듈 이름을 제공하며, 이는 JAR 파일 이름에서 파생된 기본 이름을 대체합니다.

- IBM MQ classes for JMS (com.ibm.mq.allclient.jar)에는 모듈 com.ibm.mq.javax이름이 제공됩니다.
- IBM MQ classes for Jakarta Messaging (com.ibm.mq.jakarta.client.jar)에는 모듈 com.ibm.mq.jakarta이름이 제공됩니다.

기본 MQ_HOME/java/lib 디렉터리는 모듈식 사용에 적합하지 않습니다. 모듈은 동일한 패키지를 포함할 수 없는데, 기본 디렉터리에는 여러 JAR 파일에 동일한 패키지가 포함되어 있기 때문입니다. 따라서, 필요한 JAR 파일만 포함하고 JAR 간 패키지 중복이 없는 새로운 디렉터리를 사용할 수 있습니다. 이러한 디렉터리는 module-path에 포함하기에 적합합니다.

참고: 사용 가능한 JAR 파일을 모듈식 환경에서 기본 모듈 이름을 기반으로 사용하는 애플리케이션이 있는 경우, 새 모듈 이름을 요구하도록 애플리케이션을 업데이트해야 합니다. 기본 모듈 이름은 JAR 파일 이름에서 파생됩니다.

모듈식 애플리케이션이 사용하도록 구성하기 IBM MQ classes for JMS

다음 단계를 완료하여 모듈식 애플리케이션이 (com.ibm.mq.allclient.jar) IBM MQ classes for JMS 를 사용하도록 구성할 수 있습니다:

- 응용 프로그램을 구성하여 해당 com.ibm.mq.javax 모듈을 요구하도록 설정하십시오.
- 모듈 경로에 MQ_HOME/java/lib/modules/javax 디렉터리를 포함하도록 애플리케이션을 구성하십시오.

모듈식 애플리케이션이 사용하도록 구성하기 IBM MQ classes for Jakarta Messaging

다음 단계를 완료하여 모듈식 애플리케이션이 (com.ibm.mq.jakarta.client.jar) IBM MQ classes for Jakarta Messaging 를 사용하도록 구성할 수 있습니다:

- 응용 프로그램을 구성하여 해당 com.ibm.mq.jakarta 모듈을 요구하도록 설정하십시오.
- 모듈 경로에 MQ_HOME/java/lib/modules/jakarta 디렉터리를 포함하도록 애플리케이션을 구성하십시오.

모듈식 애플리케이션이 사용하도록 구성하기 IBM MQ classes for Java

모듈식 애플리케이션에서 를 IBM MQ classes for Java 사용하려면, 두 클라이언트 JAR 파일 모두 IBM MQ classes for Jakarta Messaging를 지원하므로 용 구성 IBM MQ classes for JMS 또는 용 구성 IBM MQ classes for Java 중 하나를 사용할 수 있습니다. 그러나 귀하의 애플리케이션은 이 두 구성 중 하나만 사용해야 하며, 둘 다 사용할 수 없습니다.

IBM MQ classes for JMS 애플리케이션 서버 기능

이 주제에서는 IBM MQ classes for JMS가 Session 클래스의 고급 기능 및 ConnectionConsumer 클래스를 구현하는 방법을 설명합니다. 또한 서버 세션 풀의 기능도 요약하여 설명합니다.

중요사항: 이 정보는 참조 전용입니다. 이 인터페이스를 사용하도록 애플리케이션을 작성해서는 안 됩니다: 이 인터페이스는 리소스 IBM MQ 어댑터 내에서 EE Java 서버에 연결하기 위해 사용됩니다. 실제 연결 정보는 [386 페이지의 『리소스 IBM MQ 어댑터 사용』](#)의 내용을 참조하십시오.

IBM MQ classes for JMS 애플리케이션 서버 기능(ASF)을 지원하며, 이는 사양 *Java Message Service (Oracle Technology Network for Java Developers 참조)*에 명시되어 있습니다. 이 스펙은 이 프로그래밍 모델 내에서 세 개의 식별을 식별합니다.

- **JMS 제공자**는 ConnectionConsumer 및 고급 세션 기능을 제공합니다.
- **애플리케이션 서버**는 ServerSessionPool 및 ServerSession 기능을 제공합니다.
- **클라이언트 애플리케이션**은 JMS 제공자 및 애플리케이션 서버가 제공하는 기능을 사용합니다.

이 주제의 정보는 애플리케이션이 브로커에 대한 실시간 연결을 사용하는 경우에는 적용되지 않습니다.

JMS ConnectionConsumer

ConnectionConsumer 인터페이스는 스레드의 풀에 메시지를 동시 전달하기 위한 고성능 메소드를 제공합니다.

JMS 스펙을 사용하면 애플리케이션 서버가 ConnectionConsumer 인터페이스를 사용하여 JMS 구현과 밀접하게 통합할 수 있습니다. 이 기능은 메시지의 동시 처리를 제공합니다. 일반적으로 애플리케이션 서버는 스레드의 풀을 작성하며, JMS 구현은 이러한 스레드가 메시지를 사용할 수 있도록 합니다. JMS-인식 애플리케이션 서버(예: WebSphere Application Server)는 이 기능을 사용하여 상위 레벨 메시징 기능(예: 메시지 구동 Bean)을 제공할 수 있습니다.

일반 애플리케이션은 ConnectionConsumer를 사용하지 않지만, 전문 JMS 클라이언트는 이를 사용할 수 있습니다. 해당 클라이언트의 경우, ConnectionConsumer는 스레드의 풀에 메시지를 동시 전달하기 위한 고성능 메소드를 제공합니다. 메시지가 큐 또는 토픽에 도착할 때 JMS는 풀에서 스레드를 선택하고 이에 메시지의 배치를 전달합니다. 이를 수행하기 위해 JMS는 연관된 MessageListener의 onMessage() 메소드를 실행합니다.

각각 등록된 MessageListener가 있는 다중 Session 및 MessageConsumer 오브젝트를 구성하여 동일한 효과를 달성할 수 있습니다. 그러나 ConnectionConsumer는 보다 우수한 성능, 자원 절감 및 유연성 증대를 제공합니다. 특히 보다 적은 Session 오브젝트가 필요합니다.

ASF를 사용하여 애플리케이션 계획

이 절에서는 다음을 포함하여 애플리케이션을 계획하는 방법을 알려줍니다.

- [308 페이지의 『ASF를 사용한 포인트-투-포인트 메시징의 일반 원칙』](#)
- [309 페이지의 『ASF를 사용한 발행/구독 메시징의 일반 원칙』](#)
- [310 페이지의 『ASF의 큐에서 메시지 제거』](#)
- ASF에서 변조 메시지를 핸들링합니다. [224 페이지의 『IBM MQ classes for JMS에서 변조 메시지 핸들링』](#)의 내용을 참조하십시오.

ASF를 사용한 포인트-투-포인트 메시징의 일반 원칙

ASF를 사용한 포인트-투-포인트 메시징에 대한 일반 정보는 이 주제를 참조하십시오.

애플리케이션이 QueueConnection 오브젝트에서 ConnectionConsumer를 작성하는 경우, 이는 JMS 큐 오브젝트 및 선택자 문자열을 지정합니다. 그리고 ConnectionConsumer는 연관된 ServerSessionPool의 세션에 메시지를 제공하기 시작합니다. 메시지는 큐에 도착하며, 선택자와 일치하는 경우 이는 연관된 ServerSessionPool의 세션에 전달됩니다.

IBM MQ 용어에서, 큐 오브젝트는 로컬 큐 관리자의 QLOCAL 또는 QALIAS를 참조합니다. QALIAS인 경우, 해당 QALIAS는 QLOCAL을 참조해야 합니다. 완전히 해석된 IBM MQ QLOCAL을 기본 QLOCAL이라고 합니다. 닫혀 있지 않으며 해당 상위 QueueConnection가 시작된 경우, ConnectionConsumer를 활성이라고 합니다.

각각 서로 다른 선택자가 있는 다중 ConnectionConsumers가 동일한 기본 QLOCAL에 대해 실행될 수 있습니다. 성능을 유지하려면 원하지 않는 메시지가 큐에 누적되지 않아야 합니다. 원하지 않는 메시지는 활성 ConnectionConsumer에 일치하는 선택자가 없는 메시지입니다. 이러한 원하지 않는 메시지가 큐에서 제거될 수 있도록 QueueConnectionFactory를 설정할 수 있습니다(세부사항은 [310 페이지의 『ASF의 큐에서 메시지 제거』](#) 참조). 두 가지 방법 중 하나로 이 작동을 설정할 수 있습니다.

- JMS 관리 도구를 사용하여 QueueConnectionFactory를 MRET(NO)로 설정하십시오.

- 프로그램에서 다음을 사용하십시오.

```
MQQueueConnectionFactory.setMessageRetention(WMQConstants.WMQ_MRET_NO)
```

이 설정을 변경하지 않은 경우, 기본값은 큐에서 원하지 않는 해당 메시지를 보유하는 것입니다.

IBM MQ 큐 관리자를 설정할 때는 다음과 같은 점을 고려하십시오.

- 기본 QLOCAL이 공유 입력에 대해 사용되어야 합니다. 이를 수행하려면 다음 MQSC를 사용하십시오.

```
ALTER QLOCAL( your.qlocal.name ) SHARE GET(ENABLED)
```

- 큐 관리자에는 사용 중인 데드-레터 큐가 있어야 합니다. 데드-레터 큐에 메시지를 넣을 때 ConnectionConsumer에서 문제점이 발생하면 기본 QLOCAL의 메시지 전달이 중지됩니다. 데드-레터 큐를 정의하려면 다음을 사용하십시오.

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- ConnectionConsumer를 실행하는 사용자는 MQOO_SAVE_ALL_CONTEXT 및 MQOO_PASS_ALL_CONTEXT와 함께 MQOPEN을 수행할 권한이 있어야 합니다. 세부사항은 특정 플랫폼에 대한 IBM MQ 문서를 참조하십시오.
- 원하지 않는 메시지가 큐에 남아 있으면 시스템 성능이 저하됩니다. 따라서 메시지 선택자 간에 ConnectionConsumers가 큐에서 모든 메시지를 제거할 수 있도록 메시지 선택자를 계획하십시오.

MQSC 명령에 대한 세부사항은 [MQSC 명령](#)을 참조하십시오.

ASF를 사용한 발행/구독 메시징의 일반 원칙

ConnectionConsumer는 지정된 토픽에 대한 메시지를 수신합니다. ConnectionConsumer는 지속 가능하거나 지속 불가능합니다. ConnectionConsumer가 사용하는 큐를 지정해야 합니다.

애플리케이션이 TopicConnection 오브젝트에서 ConnectionConsumer를 작성하는 경우, 이는 토픽 오브젝트 및 선택자 문자열을 지정합니다. 그리고 ConnectionConsumer는 구독되는 토픽에 대한 보유된 발행을 포함하여 해당 토픽의 선택자와 일치하는 메시지를 수신하기 시작합니다.

또는 애플리케이션은 특정 이름과 연관된 지속 가능한 ConnectionConsumer를 작성할 수 있습니다. 지속 가능한 ConnectionConsumer가 마지막 활성인 이후 이 ConnectionConsumer는 토픽에서 발행된 메시지를 수신합니다. 이는 토픽의 선택자와 일치하는 모든 해당 메시지를 수신합니다. 그러나 ConnectionConsumer가 미리 읽기를 사용 중인 경우, 이는 닫힐 때 클라이언트 버퍼에 있는 비지속 메시지를 유실할 수 있습니다.

IBM MQ classes for JMS가 IBM MQ 메시징 제공자 마이그레이션 모드인 경우, 별도의 큐가 지속 가능하지 않은 ConnectionConsumer 구독에 사용됩니다. TopicConnectionFactory의 CCSUB 구성 가능 옵션은 사용할 큐를 지정합니다. 일반적으로, CCSUB는 동일한 TopicConnectionFactory를 사용하는 모든 ConnectionConsumer가 사용할 단일 큐를 지정합니다. 그러나 큐 이름 접두부와 별표(*)를 차례로 지정하여 각 ConnectionConsumer가 임시 큐를 생성하도록 할 수 있습니다.

IBM MQ classes for JMS가 IBM MQ 메시징 제공자 마이그레이션 모드인 경우, 토픽의 CCDSUB 특성은 지속 가능한 구독에 사용할 큐를 지정합니다. 이는 이미 존재하는 큐이거나 큐 이름 접두부 뒤에 별표(*)가 있는 큐일 수 있습니다. 이미 존재하는 큐를 지정하면 토픽을 구독하는 모든 지속 가능한 ConnectionConsumers가 이 큐를 사용합니다. 큐 이름 접두부와 별표(*)를 연속해서 지정하는 경우, 큐는 지속 가능 ConnectionConsumer가 특정 이름으로 처음 작성될 때 생성됩니다. 이 큐는 나중에 지속 가능한 ConnectionConsumer가 동일한 이름으로 작성될 때 재사용됩니다.

IBM MQ 큐 관리자를 설정할 때는 다음과 같은 점을 고려하십시오.

- 큐 관리자에는 사용 중인 데드-레터 큐가 있어야 합니다. 데드-레터 큐에 메시지를 넣을 때 ConnectionConsumer에서 문제점이 발생하면 기본 QLOCAL의 메시지 전달이 중지됩니다. 데드-레터 큐를 정의하려면 다음을 사용하십시오.

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- ConnectionConsumer를 실행하는 사용자는 MQOO_SAVE_ALL_CONTEXT 및 MQOO_PASS_ALL_CONTEXT와 함께 MQOPEN을 수행할 권한이 있어야 합니다. 세부사항은 사용자 플랫폼의 IBM MQ 문서를 참조하십시오.

- 해당되는 별도의 전용 큐를 작성함으로써 개별 ConnectionConsumer에 대한 성능을 최적화할 수 있습니다. 이는 추가 자원 사용의 비용으로 이루어집니다.

ASF의 큐에서 메시지 제거

애플리케이션이 ConnectionConsumers를 사용할 때 JMS는 다수의 상황에서 큐의 메시지를 제거해야 할 수 있습니다.

이러한 상황은 다음과 같습니다.

잘못 형식화된 메시지

JMS가 구문 분석할 수 없는 메시지가 도착할 수 있습니다.

포이즌 메시지

메시지가 백아웃 임계값에 도달할 수 있지만, ConnectionConsumer가 백아웃 큐에서 이를 리큐잉하는 데 실패합니다.

관심 없는 ConnectionConsumer

포인트-투-포인트 메시징의 경우, 원하지 않는 메시지를 보유하지 않도록 QueueConnectionFactory가 설정되면 ConnectionConsumers가 원하지 않는 메시지가 도착합니다.

이 상황에서 ConnectionConsumer는 큐에서 메시지 제거를 시도합니다. 메시지의 MQMD의 보고서 필드에서 배치 옵션은 정확한 작동을 설정합니다. 이러한 옵션은 다음과 같습니다.

MQRO_DEAD_LETTER_Q

메시지가 큐 관리자의 데드-레터 큐에 리큐잉됩니다. 기본값입니다.

MQRO_DISCARD_MSG

메시지가 제거됩니다.

ConnectionConsumer는 보고 메시지도 생성하며, 이 또한 메시지의 MQMD의 보고서 필드에 의존합니다. 이 메시지는 ReplyToQmgr에서 메시지의 ReplyToQ에 송신됩니다. 보고 메시지가 송신될 때 오류가 있으면 메시지가 데드-레터 큐에 대신 송신됩니다. 메시지의 MQMD의 보고서 필드에서 예외 보고서 옵션은 보고 메시지의 세부사항을 설정합니다. 이러한 옵션은 다음과 같습니다.

MQRO_EXCEPTION

원래 메시지의 MQMD가 포함된 보고 메시지가 생성됩니다. 이는 메시지 본문 데이터를 포함하지 않습니다.

MQRO_EXCEPTION_WITH_DATA

MQMD, 임의의 MQ 헤더 및 100바이트의 본문 데이터가 포함된 보고 메시지가 생성됩니다.

MQRO_EXCEPTION_WITH_FULL_DATA

원래 메시지의 모든 데이터가 포함된 보고 메시지가 생성됩니다.

기본값

보고 메시지가 생성되지 않습니다.

보고 메시지가 생성될 때는 다음 옵션이 고려됩니다.

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

변조 메시지를 리큐잉할 수 없는 경우(가능한 원인: 데드-레터 큐가 가득 차 있거나 권한 부여가 올바르게 설정되지 않음)에 발생하는 상황은 메시지의 지속성에 달려 있습니다. 메시지가 비지속인 경우, 메시지는 버려지며 보고 메시지가 생성되지 않습니다. 메시지가 지속인 경우, 해당 목적지에서 대기 중인 모든 연결 이용자에 대한 메시지의 전달이 중지됩니다. 해당 연결 이용자를 닫아야 하며, 재작성되고 메시지 전달이 다시 시작될 수 있기 전에 문제점이 해결되어야 합니다.

데드-레터 큐를 정의하고 이를 주기적으로 확인하여 문제점이 발생하지 않도록 보장하는 일이 중요합니다. 특히 데드-레터 큐가 최대 용량에 도달하지 않았는지와 최대 메시지 크기가 모든 메시지에 대해 충분히 큰지 확인하십시오.

메시지가 데드-레터 큐에 리큐잉될 때는 IBM MQ 데드 레터 헤더(MQDLH)가 이에 선행합니다. MQDLH의 형식에 대한 세부사항은 MQDLH - 데드 레터 헤더를 참조하십시오. 다음 필드에 의해 ConnectionConsumer가 데드-레터 큐에 둔 메시지 또는 ConnectionConsumer가 생성한 보고 메시지를 식별할 수 있습니다.

- PutApplType은 MQAT_JAVA (0x1C)입니다.
- PutApplName: " MQ JMS ConnectionConsumer "

이러한 필드는 보고 메시지의 MQMD 및 데드-레터 큐의 메시지의 MQDLH에 있습니다. MQMD의 피드백 필드와 MQDLH의 이유 필드에는 오류를 설명하는 코드가 포함되어 있습니다. 이러한 코드에 대한 세부사항은 312 페이지의 『ASF의 이유 및 피드백 코드』의 내용을 참조하십시오. 기타 필드는 MQDLH - 데드 레터 헤더에 설명되어 있습니다.

ASF에서 변조 메시지 핸들링

ASF(Application Server Facilities)에서, 변조 메시지 핸들링은 IBM MQ classes for JMS의 다른 위치에서와는 조금 다르게 핸들링됩니다.

IBM MQ classes for JMS에서 유해 제품 정보 핸들링에 대한 정보는 224 페이지의 『IBM MQ classes for JMS에서 변조 메시지 핸들링』의 내용을 참조하십시오.

ASF(Application Server Facilities)를 사용할 때는 MessageConsumer가 아닌 ConnectionConsumer가 변조 메시지를 처리합니다. ConnectionConsumer는 큐의 BackoutThreshold 및 BackoutRequeueQName 특성에 따라 메시지를 리큐잉합니다.

애플리케이션이 ConnectionConsumers를 사용하는 경우, 메시지가 백아웃되는 환경은 애플리케이션 서버가 제공하는 세션에 달려 있습니다.

- 세션이 AUTO_ACKNOWLEDGE 또는 DUPS_OK_ACKNOWLEDGE로 트랜잭션되지 않은 경우, 메시지는 시스템 오류 이후에만 또는 애플리케이션이 예상치 못하게 종료된 경우에 백아웃됩니다.
- 세션이 CLIENT_ACKNOWLEDGE로 트랜잭션되지 않은 경우, 수신확인되지 않은 메시지는 Session.recover()를 호출 중인 애플리케이션 서버에 의해 백아웃될 수 있습니다.

일반적으로 MessageListener의 클라이언트 구현 또는 애플리케이션 서버는 Message.acknowledge()를 호출합니다. Message.acknowledge()는 이제까지 세션에 전달된 모든 메시지를 수신확인합니다.

- 세션이 트랜잭션된 경우, 수신확인되지 않은 메시지는 Session.rollback()을 호출 중인 애플리케이션 서버에 의해 백아웃될 수 있습니다.
- 애플리케이션 서버가 XASession을 제공하는 경우, 메시지는 분배 트랜잭션에 따라 커밋되거나 백아웃됩니다. 애플리케이션 서버에는 트랜잭션 완료에 대한 책임이 있습니다.

관련 개념

224 페이지의 『IBM MQ classes for JMS에서 변조 메시지 핸들링』

독성 메시지는 수신 애플리케이션이 처리할 수 없는 메시지를 의미합니다. 예를 들어 메시지의 유형이 예상치 못했거나, 애플리케이션의 로직이 처리할 수 없는 정보를 포함하는 경우 등이 해당됩니다. 애플리케이션이 메시지를 처리할 수 없는 경우, 해당 메시지를 원래 있던 큐로 롤백합니다. 기본적으로 중독된 메시지를 애플리케이션에 반복적으로 IBM MQ classes for JMS 재전송하며, 이로 인해 애플리케이션이 중독된 메시지를 처리하고 롤백하려는 루프에 계속 갇힐 수 있습니다. 이를 방지하기 위해, 애플리케이션에 전달된 후 지정된 횟수만큼 롤백된 독성 메시지를 백아웃 큐로 이동시킬 수 IBM MQ classes for JMS 있습니다.

ASF 오류 처리

이 절에서는 311 페이지의 『ASF의 오류 상태에서 복구』 및 312 페이지의 『ASF의 이유 및 피드백 코드』를 포함하여 다양한 측면의 오류 핸들링을 다루고 있습니다.

ASF의 오류 상태에서 복구

ConnectionConsumer에서 심각한 오류가 발생하는 경우, 동일한 QLOCAL에 관심이 있는 모든 ConnectionConsumer에 대한 메시지 전달이 모두됩니다. 이러한 상황이 발생하면 영향을 받은 연결에서 등록된 ExceptionListener는 알림을 받습니다. 애플리케이션이 이러한 오류 상태에서 복구할 수 있는 두 가지 방법이 있습니다.

일반적으로, 이러한 네이처의 심각한 오류는 ConnectionConsumer가 메시지를 데드-레터 큐에 리큐잉할 수 없거나 QLOCAL에서 메시지를 읽을 때 오류가 발생하는 경우에 발생합니다.

영향을 받은 Connection에서 등록된 ExceptionListener가 알림을 받으므로, 이를 사용하여 문제점의 원인을 식별할 수 있습니다. 일부 경우에는 시스템 관리자가 문제점을 해결하기 위해 개입해야 합니다.

이러한 오류 상태에서 복구하려면 다음 기술 중 하나를 사용하십시오.

- 영향을 받은 모든 ConnectionConsumer에서 close()를 호출하십시오. 애플리케이션은 영향을 받은 모든 ConnectionConsumers가 닫히고 시스템 문제점이 해결된 후에만 새 ConnectionConsumers를 작성할 수 있습니다.
- 영향을 받은 모든 연결에서 stop()을 호출하십시오. 모든 연결이 중지되고 시스템 문제점이 해결된 후에 애플리케이션은 해당 연결의 start()를 성공적으로 수행할 수 있습니다.

ASF의 이유 및 피드백 코드

이유 및 피드백 코드를 사용하여 오류의 원인을 판별할 수 있습니다. ConnectionConsumer에 의해 생성된 공통 이유 코드가 여기서 제공됩니다.

오류의 원인을 판별하려면 다음 정보를 사용하십시오.

- 보고 메시지의 피드백 코드
- 데드-레터 큐에서 임의의 메시지의 MQDLH의 이유 코드

ConnectionConsumer는 다음의 이유 코드를 생성합니다.

MQRC_BACKOUT_THRESHOLD_REACHED(0x93A, 2362)

원인

메시지가 QLOCAL에 정의된 백아웃 임계값에 도달했지만 백아웃 큐가 정의되지 않았습니다.

백아웃 큐를 정의할 수 없는 플랫폼에서 메시지가 JMS-정의 백아웃 임계값인 20에 도달했습니다.

Action

이를 원하지 않으면 관련 QLOCAL에 대해 백아웃 큐를 정의하십시오. 또한 다중 백아웃의 원인을 찾으십시오.

MQRC_MSG_NOT_MATCHED(0x93B, 2363)

원인

포인트-투-포인트 메시징에서, 큐를 모니터링 중인 ConnectionConsumers에 대한 선택자와 일치하지 않는 메시지가 있습니다. 성능을 유지하기 위해 메시지는 데드-레터 큐에 리큐잉됩니다.

Action

이러한 상황을 피하려면 큐를 사용 중인 ConnectionConsumers가 모든 메시지를 처리하는 선택자 세트를 제공하는지 확인하거나 메시지를 보유하도록 QueueConnectionFactory를 설정하십시오.

또는 메시지의 소스를 검사하십시오.

MQRC_JMS_FORMAT_ERROR(0x93C, 2364)

원인

JMS가 큐에서 메시지를 해석할 수 없습니다.

Action

메시지의 원본을 검사하십시오. JMS는 일반적으로 예상치 못한 형식의 메시지를 BytesMessage 또는 TextMessage로서 전달합니다. 가끔, 메시지가 매우 잘못 형식화된 경우에 이는 실패합니다.

이 필드에 나타나는 기타 코드는 메시지를 백아웃 큐에 리큐잉하는 데 실패하여 나타납니다. 이러한 상황에서 코드는 리큐잉이 실패한 이유를 설명합니다. 이러한 오류의 원인을 진단하려면 [API>API 완료 및 원인 코드를 참조](#) 하십시오.

보고 메시지를 ReplyToQ에 넣을 수 없는 경우에는 이를 데드-레터 큐에 넣습니다. 이러한 상황에서 MQMD의 피드백 필드는 이 주제에서 설명한 대로 완료됩니다. MQDLH의 이유 필드는 보고 메시지를 ReplyToQ에 둘 수 없는 이유를 설명합니다.

AFS에서 서버 세션 풀의 기능

이 주제에서는 서버 세션 풀의 기능을 요약하여 설명합니다.

[313 페이지의 그림 45](#)에서는 ServerSessionPool 및 ServerSession 기능의 원칙을 요약하여 설명합니다.

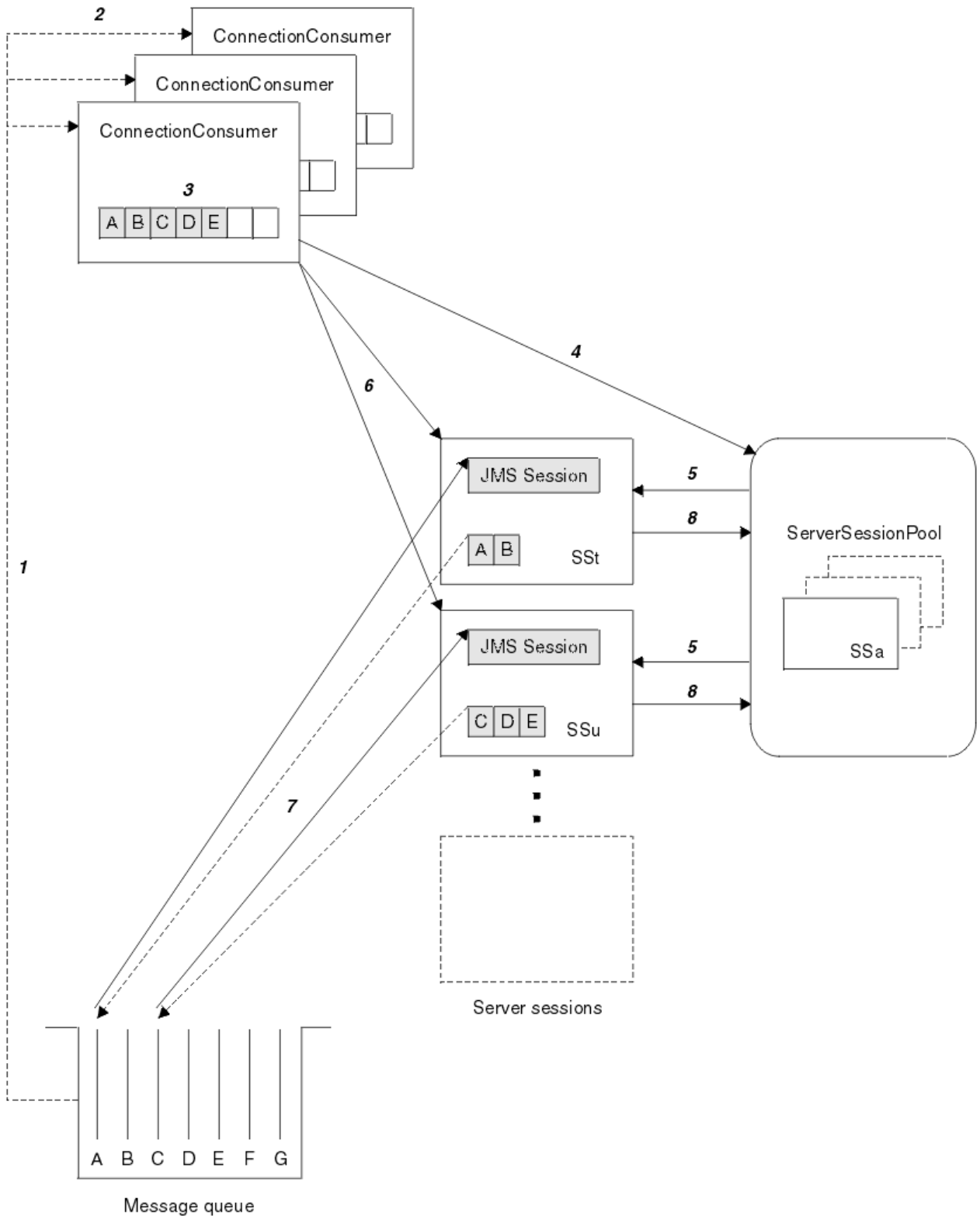


그림 45. ServerSessionPool 및 ServerSession 기능

1. ConnectionConsumer는 큐에서 메시지 참조를 가져옵니다.
2. 각 ConnectionConsumer는 특정 메시지 참조를 선택합니다.
3. ConnectionConsumer 버퍼는 선택된 메시지 참조를 보유합니다.
4. ConnectionConsumer는 ServerSessionPool에서 하나 이상의 ServerSession을 요청합니다.
5. ServerSession이 ServerSessionPool에서 할당됩니다.

6. ConnectionConsumer는 ServerSessions에 대해 메시지 참조를 지정하며 ServerSession 스레드 실행을 시작합니다.
7. 각 ServerSession은 해당 참조 메시지를 큐에서 검색합니다. JMS Session과 연관된 MessageListener 에서 onMessage 메소드로 전달합니다.
8. 해당 처리가 완료된 후에 ServerSession은 풀에 리턴됩니다.

애플리케이션 서버는 일반적으로 ServerSessionPool 및 ServerSession 기능을 제공합니다.

▶ z/OS JVM CICS Liberty 서버에서 IBM MQ classes for JMS 사용

Java JVM CICS Liberty 서버에서 실행 중인 프로그램은 IBM MQ classes for JMS 사용하여 접근할 수 있습니다 IBM MQ.

사용 중인 리소스 IBM MQ 어댑터 버전이 IBM MQ 9.1.0 이상이어야 합니다. Fix Central에서 자원 어댑터를 확보할 수 있습니다(398 페이지의 『리소스 어댑터 및 IBM MQ IBM MQ 리소스 어댑터 획득 Jakarta Messaging』 참조).

Profile JVM에는 두 Liberty 가지 유형이 있으며, CICS 5.3 이후 버전에서는 연결 가능한 유형이 다음과 IBM MQ 같이 제한됩니다:

CICS Liberty 표준

- IBM MQ 자원 어댑터는 서비스 중인 모든 IBM MQ 버전에 CLIENT 모드로 연결할 수 있습니다.
- 동일한 CICS 지역에서 동일한 큐 매니저에 대한 활성 연결(CICS MQCONN 리소스 정의)이 CICS 존재하지 않을 경우, 리소스 IBM MQ 어댑터는 BINDINGS 모드에서 서비스 중인 IBM MQ for z/OS 모든 버전의 리소스에 연결할 수 있습니다.

CICS Liberty 통합됨

- IBM MQ 자원 어댑터는 서비스 중인 모든 IBM MQ 버전에 CLIENT 모드로 연결할 수 있습니다.
- BINDINGS 모드 연결은 지원되지 않습니다.

시스템 설정 및 구성에 대한 자세한 내용은 [Liberty JVM 서버에서 IBM MQ classes for JMS 사용하기](#) 문서를 CICS 참조하십시오.

사용 IBM MQ classes for JMS/ Jakarta Messaging in IMS

표준 기반 메시징 지원은 환경 IMS 내에서 또는 IBM MQ classes for JMS 사용을 통해 IBM MQ classes for Jakarta Messaging 제공됩니다.

엔터프라이즈에서 사용하는 IMS 시스템에 대한 시스템 요구사항을 확인하십시오. 자세한 정보는 [IMS 15.2](#)를 참조하십시오.

이 주제 모음은 환경에서 IMS IBM MQ classes for JMS 를 설정하는 방법과 클래식(JMS 1.1) 및 단순화된(JMS 2.0) 인터페이스 사용 시 적용되는 API 제한 사항을 설명합니다. API별 정보의 목록은 [319 페이지의 『JMS API 제한사항』](#)의 내용을 참조하십시오.

참고: 유사한 제한사항인 레거시(JMS 1.0.2) 도메인별 인터페이스에 적용되지만 특별히 여기서 설명하지는 않습니다.

▶ **JM 3.0** IBM MQ 9.3.0부터, Jakarta Messaging 3.0 는 새로운 애플리케이션 개발에 지원됩니다. IBM MQ 9.3.0 그리고 향후에도 기존 애플리케이션에 대한 지원을 JMS 2.0 계속할 예정입니다. 동일한 애플리케이션에서 두 JMS 2.0 Jakarta Messaging 3.0 API를 모두 사용하는 것은 지원되지 않습니다. 자세한 내용은 [『JMS/ Jakarta Messaging용 IBM MQ 클래스 사용』](#)을 참조하십시오.

지원되는 IMS 종속 영역

다음 종속 영역이 지원됩니다.

- MPR
- BMP
- IFP

- JMP 31비트 및 64비트 Java 가상 머신(JVM)
- JBP 31비트 및 64비트 JVM

다음 항목에서 특별히 언급되지 않는 한, IBM MQ classes for JMS 와 는 모든 지역 IBM MQ classes for Jakarta Messaging 유형에서 동일하게 동작합니다.

지원되는 Java 가상 머신

IBM MQ classes for JMS 그리고 런타임 환경, Java 기술 IBM MQ classes for Jakarta Messaging IBM 에디션 8 이 필요합니다. IBM Semeru Runtime Certified Edition for z/OS, 버전 11은 지원되지 않습니다.

기타 제한사항

다음 제한 사항은 환경에서 IMS IBM MQ classes for JMS 를 사용할 때 적용됩니다:

- 클라이언트 모드 연결은 지원되지 않습니다.
- 연결은 IBM MQ 메시징 제공자 Normal, 모드를 사용하는 IBM MQ 8.0 큐 관리자에만 지원됩니다.
연결 팩토리의 **PROVIDERVERSION** 속성은 지정되지 않거나 값이 7 이상이어야 합니다.
- XA 연결 팩토리(예: `com.ibm.mq.jms.MQXAConnectionFactory`) 사용은 지원되지 않습니다.

관련 태스크

[정의하다 IBM MQ IMS](#)

/와 IBM MQ classes for JMS 함께 사용하기 위한 IMS 어댑터 설정 Jakarta Messaging

IBM MQ classes for JMS 다른 프로그래밍 언어에서 사용하는 IBM MQ classes for Jakarta Messaging 것과 동일한 IBM MQIMS 어댑터를 활용합니다. 이 어댑터는 IMS ESAF(External Subsystem Attach Facility)를 사용합니다.

시작하기 전에

다음 절차를 완료하기 전에, '[IMS 어댑터 설정](#)'에 설명된 대로 관련 큐 관리자 및 IMS 제어 영역과 종속 영역에 대한 IMS 어댑터를 구성해야 합니다.



주의: 다른 용도로 동적 스텝을 필요로 하지 않는 한 동적 스텝 빌드를 설명하는 단계를 수행할 필요가 없습니다.

어댑터를 IMS 구성한 후 다음 절차를 수행하십시오.

프로시저

1. IBM MQ classes for JMS 고유 라이브러리를 포함하도록 종속 영역 JCL(예: DFSJVM EV)에서 ENVIRON 매개변수별로 참조된 IMS PROCLIB의 멤버에서 LIBPATH 변수를 업데이트하십시오.
즉, `libmqjims.so`을(를) 포함하는 zFS 디렉토리입니다. 예를 들어, DFSJVM EV는 다음과 같이 보일 수 있으며, 마지막 줄은 또는 IBM MQ classes for Jakarta Messaging 네이티브 IBM MQ classes for JMS 라이브러리가 포함된 디렉토리입니다:

```
LIBPATH=>
/java/latest/bin/j9vm:>
/java/latest/bin:>
/ims/latest/dbdc/imsjava/lib:>
/mqm/latest/java/lib
```

2. 의존 IMS 영역에서 사용하는 `JVM`의 클래스 경로에 `` 또는 `IBM MQ classes for JMS` IBM MQ classes for Jakarta Messaging `를 추가하려면 `java.class.path` 옵션을 업데이트하십시오.
[IMS PROCLIB 데이터 세트의 DFSJVMMS 멤버](#)의 지시사항에 따라 이를 수행하십시오.

예를 들어 다음을 사용할 수 있으며, 여기서 굵게 표시된 행은 업데이트를 나타냅니다.

JM 3.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutn.jar:/ims/latest/dbdc/imsjava/imsudb.jar:  
/mqm/latest/java/lib/com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutn.jar:/ims/latest/dbdc/imsjava/imsudb.jar:  
/mqm/latest/java/lib/com.ibm.mq.allclient.jar
```

참고: 해당 디렉토리에 다양한 jar 파일이 존재하지만, 또는 IBM MQ classes for JMS 를 포함하는 디렉토리에서 IBM MQ classes for Jakarta Messaging(JMS 2.0) 또는 com.ibm.mq.allclient.jar 만 필요함니다 com.ibm.mq.jakarta.client.jar (Jakarta Messaging 3.0).

3. 또는 IBM MQ classes for JMS 을 사용할 종속 IMS 영역을 IBM MQ classes for Jakarta Messaging중지하고 다시 시작하십시오.

다음에 수행할 작업

연결 팩토리 및 목적지를 작성 및 구성하십시오.

연결 팩토리 및 목적지의 IBM MQ 구현을 인스턴스화하는 데는 세 가지 가능한 접근법이 있습니다. 자세한 내용은 [199 페이지의 『연결 팩토리 및 목적지 작성 및 구성』](#)의 내용을 참조하십시오.

이러한 세 가지 접근법은 IMS 환경에서 모두 유효합니다.

관련 개념

[IMS 어댑터 설정](#)

[정의하다 IBM MQ IMS](#)

트랜잭션 작동

IMS 환경에서 IBM MQ classes for JMS 가 보내고 받은 메시지는 항상 현재 태스크에서 활성 상태인 IMS 작업 단위 (UOW) 와 연관됩니다.

UOW는 com.ibm.ims.dli.tm.Transaction 오브젝트의 인스턴스에서 커밋 또는 롤백 메소드를 호출하거나 UOW가 암시적으로 커밋된 경우 정상적으로 종료되는 IMS 태스크를 통해서만 완료될 수 있습니다. IMS 태스크가 비정상적으로 종료되면 UOW가 롤백됩니다.

이에 대한 결과로 Connection.createSession 또는 ConnectionFactory.createContext 메소드를 호출할 경우 **transacted** 및 **acknowledgeMode** 인수의 값이 무시됩니다. 또한 다음 메소드는 지원되지 않습니다. 다음 메소드 중에서 호출할 경우, 세션 케이스에서 IllegalStateException이 발생합니다.

- javax.jms.Session.commit()
- javax.jms.Session.recover()
- javax.jms.Session.rollback()

및 JMS 컨텍스트 케이스의 IllegalStateExceptionRuntimeSession:

- javax.jms.JMSContext.commit()
- javax.jms.JMSContext.recover()
- javax.jms.JMSContext.rollback()

이 동작에 대한 하나의 예외가 있습니다. 세션 또는 JMS 컨텍스트가 다음 메커니즘 중 하나를 사용하여 작성된 경우:

- Connection.createSession(false, Session.AUTO_ACKNOWLEDGE)
- Connection.createSession(Session.AUTO_ACKNOWLEDGE)
- ConnectionFactory.createContext(JMSContext.AUTO_ACKNOWLEDGE)

해당 세션의 동작 또는 JMS 컨텍스트는 다음과 같습니다.

- 보내진 메시지는 IMS UOW 외부로 보내집니다. 즉, 이는 대상 목적지에서 즉시 또는 제공된 전달 지연 간격이 끝날 경우 사용 가능해집니다.
- 세션 또는 JMS 컨텍스트를 작성한 연결 팩토리에 [SYNCPPOINTALLGETS](#) 특성이 지정되지 않은 경우 IMS UOW 외부에서 비지속 메시지가 수신됩니다.
- 지속 메시지는 항상 IMS UOW 내부에서 수신됩니다.

예를 들어 UOW가 롤백되더라도 감사 메시지를 큐에 쓰려는 경우 이 방법이 유용할 수 있습니다.

IMS 동기점의 영향

IBM MQ classes for JMS는 ESAF를 이용하는 기존 IBM MQ 어댑터 지원을 기반으로 합니다. 이는 동기점이 발생할 때 IMS 어댑터에서 닫히는 모든 열린 핸들을 포함하여 문서화된 동작이 적용됨을 의미합니다.

 자세한 정보는 72 페이지의 『IMS 애플리케이션의 동기점』의 내용을 참조하십시오.

이 점에 대해 설명하려면 JMP 환경에서 실행 중인 다음 코드를 고려하십시오. 두 번째 호출은 모든 열린 IBM MQ 연결 및 객체 핸들이 닫히도록 하는 `messageQueue.getUnique(inputMessage)` 코드 결과로 인해 `JMSEException` 를 `mp.send()` 발생시킵니다.

`getUnique()` 호출이 `Transaction.commit()`로 바뀐 경우 유사한 동작이 관찰되지만 `Transaction.rollback()`이 사용된 경우에는 그렇지 않습니다.

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Get a message from an IMS message queue. This results in a GU call
//which results in all MQ handles being closed.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//This attempt to send another message will result in a JMSEException containing a
//MQRC_HCONN_ERROR as the connection/handle has been closed.
mp.send(m);
```

이 시나리오에서 사용할 올바른 코드는 다음과 같습니다. 이 경우에는 IBM MQ에 대한 연결이 `getUnique()` 호출 전에 닫힙니다. 그런 다음 연결 및 세션은 다른 메시지를 송신하기 위해 재작성됩니다.

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Close the connection to MQ, which closes all MQ object handles.
//The send of the message will be committed by the subsequent GU call.
c.close();
c = null;
s = null;
mp = null;

//Get a message from an IMS message queue. This results in a GU call.
Application a = ApplicationFactory.createApplication();
```

```

MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//Re-create the connection to MQ and send another message;
c = cf.createConnection();
s = c.createSession();
mp = s.createProducer(q);
m = s.createTextMessage("Hello world 2!");
mp.send(m);

```

IMS 어댑터 사용을 위한 고려사항

다음 제한사항에 유념해야 합니다. 각 큐 관리자에 대해 연결 핸들을 하나씩만 가질 수 있습니다. 네이티브 코드와 함께 JMS 사용할 때 IBM MQ 상호작용에는 함의가 있습니다. 연결 인증 및 권한 부여에 대한 제한사항이 있습니다.

각 큐 관리자에 대한 하나의 연결 핸들

특정 큐 관리자에 대해 한 번에 하나의 연결 핸들만 IMS 종속 영역에서 허용됩니다. 동일한 큐 관리자에 연결하기 위한 모든 후속 시도는 기존 핸들을 재사용합니다.

IBM MQ IBM MQ classes for JMS이 동작은 만 사용하는 애플리케이션에서는 아무런 문제를 일으키지 않지만, COBOL 나 C와 같은 언어로 작성된 네이티브 코드에서와 MQI 를 IBM MQ classes for JMS 모두 사용하는 경우,와 상호작용하는 애플리케이션에서는 문제를 일으킬 수 있습니다.

네이티브 코드와 JMS 함께 사용할 때 IBM MQ 상호작용의 함의

코드와 네이티브 코드가 모두 동일한 IBM MQ 기능을 사용하며, 네이티브 코드나 자바 코드 Java 중 하나를 종료하기 전에 연결이 IBM MQ 닫히지 않을 경우 문제가 발생할 수 있습니다.

예를 들어, 다음 의사 코드에서 큐 관리자에 대한 연결 핸들은 원래 코드에서 Java 를 사용하여 IBM MQ classes for JMS 설정됩니다. 연결 핸들은 COBOL 코드에서 재사용되며 MQDISC에 대한 호출에 의해 무효화됩니다.

다음 번에 IBM MQ classes for JMS가 연결을 이용할 때 MQRC_HCONN_ERROR 결과의 이유 코드와 함께 JMSEException을 처리합니다.

```

COBOL code running in message processing region
Use the Java Native Interface (JNI) to call Java code
Create MQ connection and session - this creates an MQ connection handle
Send message to MQ queue
Store connection and session in static variable
Return to COBOL code

MQCONN - picks up MQ connection handle established in Java code
MQDISC - invalidates connection handle

Use the Java Native Interface (JNI) to call Java code
Get session from static variable
Create a message consumer - fails as connection handle invalidated

```

MQRC_HCONN_ERROR를 유발할 수 있는 다른 유사한 사용 패턴이 있습니다.

네이티브 코드와 Java C++ 코드 간에 연결 핸들을 IBM MQ 공유하는 것이 가능하지만(예를 들어, MQDISC 호출이 없었다면 이전 예제가 작동했을 것임), 일반적으로는 C++ Java 코드에서 네이티브 코드로 전환하거나 그 반대의 경우에 연결 핸들을 닫는 것이 최선의 방법입니다.

연결 인증 및 권한 부여

JMS 스펙은 연결 또는 JMS 컨텍스트 오브젝트를 작성할 때 인증 및 권한 부여를 위해 사용자 이름 및 비밀번호가 지정되도록 허용합니다.

이는 IMS 환경에서 지원되지 않습니다. 사용자 이름과 암호를 지정하여 연결을 시도하면 예외가 JMS Exception 발생합니다. 사용자 이름 및 비밀번호를 지정하는 동안 JMS 컨텍스트를 작성하려고 시도할 경우, JMSRuntimeException이 처리됩니다.

대신, 환경에서 IMS IBM MQ 에 연결할 때 기존 인증 및 권한 부여 메커니즘을 사용해야 합니다.

자세한 정보는 z/OS에서 [보안 설정](#)을 참조하십시오. 특히 사용될 수 있는 사용자 ID를 설명하는 [보안 검사용 사용자 ID](#)를 참조하십시오.

관련 태스크

[z/OS에서 보안 설정](#)

JMS API 제한사항

사양 JMS 관점에서, 항상 JTA 트랜잭션이 진행 중인 EE(Enterprise Java Edition) 또는 Jakarta EE 호환 애플리케이션 서버로 IMS 취급합니다 IBM MQ classes for JMS .

예를 들어 JMS 스펙이 JTS 트랜잭션이 진행 중인 동안 JEE EJB 또는 웹 컨테이너에서 이를 호출할 수 없다고 언급하므로 사용자는 IMS에서 `javax.jms.Session.commit()`를 호출할 수 없습니다.

이로 인해 예 설명된 제한 사항 외에도 API에 [JMS 316 페이지의 『트랜잭션 작동』](#) 다음과 같은 제한 사항이 적용됩니다.

클래식 API 제한사항

- `javax.jms.Connection.createConnectionConsumer(javax.jms.Destination, String, javax.jms.ServerSessionPool, int)` 항상 `JMSException`을 던진다.
- `javax.jms.Connection.createDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` 항상 `JMSException`을 던진다.
- 연결의 기존 세션이 이미 활성 상태인 경우 `javax.jms.Connection.createSession`의 세 가지 변형은 모두 항상 `JMSException`을 처리합니다.
- `javax.jms.Connection.createSharedConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` 항상 `JMSException`을 던진다.
- `javax.jms.Connection.createSharedDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` 항상 `JMSException`을 던진다.
- `javax.jms.Connection.setClientID()` 항상 `JMSException`을 던진다.
- `javax.jms.Connection.setExceptionListener(javax.jms.ExceptionListener)` 항상 `JMSException`을 던진다.
- `javax.jms.Connection.stop()` 항상 `JMSException`을 던진다.
- `javax.jms.MessageConsumer.setMessageListener(javax.jms.MessageListener)` 항상 `JMSException`을 던진다.
- `javax.jms.MessageConsumer.getMessageListener()` 항상 `JMSException`을 던진다.
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message, javax.jms.CompletionListener)` 항상 `JMSException`을 던진다.
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message, int, int, long, javax.jms.CompletionListener)` 항상 `JMSException`을 던진다.
- `javax.jms.MessageProducer.send(javax.jms.Message, int, int, long, javax.jms.CompletionListener)` 항상 `JMSException`을 던진다.
- `javax.jms.MessageProducer.send(javax.jms.Message, javax.jms.CompletionListener)` 항상 `JMSException`을 던진다.
- `javax.jms.Session.run()` 항상 `JMSRuntimeException`을 던진다.
- `javax.jms.Session.setMessageListener(javax.jms.MessageListener)` 항상 `JMSException`을 던진다.
- `javax.jms.Session.getMessageListener()` 항상 `JMSException`을 던진다.

간소화된 API 제한사항

- `javax.jms.JMSContext.createContext(int)` 항상 `JMSRuntimeException`을 던진다.

- `javax.jms.JMSContext.setClientID(String)` 항상. `JMSRuntimeException`을 던진다.
- `javax.jms.JMSContext.setExceptionListener(javax.jms.ExceptionListener)` 항상. `JMSRuntimeException`을 던진다.
- `javax.jms.JMSContext.stop()` 항상. `JMSRuntimeException`을 던진다.
- `javax.jms.JMSProducer.setAsync(javax.jms.CompletionListener)` 항상. `JMSRuntimeException`을 던진다.

IBM MQ classes for Java 사용

사용 IBM MQJava 환경에서 사용하십시오. IBM MQ classes for Java를 사용하면 Java 애플리케이션을 IBM MQ 클라이언트로 IBM MQ에 연결하거나 IBM MQ 큐 관리자에 직접 연결할 수 있습니다.

참고:

Stabilized IBM은 IBM MQ classes for Java에 대해 추가적인 개선 계획이 없으며 IBM MQ 8.0에 제공된 레벨에서 기능적으로 안정되어 있습니다. IBM MQ classes for Java를 사용하는 기존 애플리케이션은 계속 완벽하게 지원되지만, 새 기능은 추가되지 않으며, 개선 요청은 거부됩니다. 완전히 지원이란 IBM MQ 시스템 요구사항의 변경에 따라 필요한 모든 변경사항과 함께 결함이 수정되는 것을 의미합니다.

IBM MQ classes for Java는 IMS에서 지원되지 않습니다.

IBM MQ classes for Java는 WebSphere Liberty에서 지원되지 않습니다. 이는 IBM MQ Liberty 메시징 기능 또는 일반 JCA 지원에서는 사용될 수 없습니다. 자세한 정보는 [J2EE/JEE 환경에서 WebSphere MQ Java 인터페이스 사용](#)을 참조하십시오.

IBM MQ classes for Java 애플리케이션이 리소스에 JavaIBM MQ 접근하기 위해 사용할 수 있는 세 가지 대 API 중 하나입니다. 다른 API들은 다음과 같습니다:

- **JM 3.0** IBM MQ classes for Jakarta Messaging
- **JMS 2.0** IBM MQ classes for JMS

자세한 정보는 87 페이지의 [『IBM MQ - Java API 선택』](#)의 내용을 참조하십시오.

IBM MQ 9.3에서 IBM MQ classes for Java는 Java 8로 빌드됩니다. Java 8 런타임 환경에서는 이전 버전의 클래스 파일 실행을 지원합니다.

IBM MQ classes for Java Message Queue Interface (MQI)와 네이티브 IBM MQ API를 포함하며, C++ 및 .NET 인터페이스와 유사한 객체 모델을 사용합니다.

프로그래밍 가능한 옵션을 사용하면 IBM MQ classes for Java가 다음 방법 중 하나로 IBM MQ에 연결할 수 있습니다.

- [클라이언트 모드](#)에서 TCP/IP(Transmission Control Protocol/Internet Protocol)를 사용하여 IBM MQ MQI client로 연결
- [바인딩 모드](#)에서 네이티브 인터페이스 Java (JNI)를 IBM MQ 사용하여 직접 연결

참고: 자동 클라이언트 재연결은 IBM MQ classes for Java에서 지원되지 않습니다.

클라이언트 모드 연결

IBM MQ classes for Java 애플리케이션은 클라이언트 모드를 사용하여 지원되는 큐 관리자에 연결할 수 있습니다.

클라이언트 모드로 큐 관리자에 연결하기 위해 IBM MQ classes for Java 애플리케이션이 큐 관리자가 실행 중인 시스템과 동일한 시스템 또는 다른 시스템에서 실행될 수 있습니다. 어느 경우든 IBM MQ classes for Java가 TCP/IP를 통해 큐 관리자에 연결합니다.

클라이언트 모드 연결을 사용하도록 애플리케이션을 작성하는 방법에 대한 자세한 정보는 343 페이지의 [『IBM MQ classes for Java 연결 모드』](#)의 내용을 참조하십시오.

바인딩 모드 연결

바인딩 모드에서 사용할 때 IBM MQ classes for Java는 네트워크를 통해 통신하지 않고 JNI(Java Native Interface)를 사용하여 기존 큐 관리자 API를 직접 호출합니다. 대부분의 환경에서 바인딩 모드로 연결하면 TCP/IP 통신의 비용을 들이지 않으므로 클라이언트 모드로 연결하는 것보다 IBM MQ classes for Java 애플리케이션의 성능이 향상됩니다.

바인딩 모드에서 IBM MQ classes for Java를 사용하여 연결하는 애플리케이션은 연결할 큐 관리자와 동일한 시스템에서 실행해야 합니다.

애플리케이션을 IBM MQ classes for Java 실행하는 데 사용되는 Java runtime environment 서버는 IBM MQ classes for Java 라이브러리를 로드하도록 구성되어야 합니다. 자세한 정보는 [329 페이지의 『IBM MQ classes for Java 라이브러리』](#)의 내용을 참조하십시오.

바인딩 모드 연결을 사용하도록 애플리케이션을 작성하는 방법에 대한 자세한 정보는 [343 페이지의 『IBM MQ classes for Java 연결 모드』](#)의 내용을 참조하십시오.

관련 개념

[IBM MQ Java 언어 인터페이스](#)

[83 페이지의 『사용 IBM MQ classes for JMS/Jakarta Messaging』](#)

IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 는 와 함께 제공되는 IBM MQ메시징 Java 공급자입니다. 이 메시징 공급자들은 및 Jakarta Messaging JMS 사양에서 정의된 인터페이스를 구현하는 것 외에도, 메시징 Java API에 두 가지 확장 기능을 추가합니다.

관련 태스크

[IBM MQ classes for Java 애플리케이션 추적](#)

[문제 해결 Java 및 JMS 문제점](#)

IBM MQ classes for Java를 사용해야 하는 이유가 무엇입니까?

Java 애플리케이션은 IBM MQ classes for Java 또는 IBM MQ classes for JMS를 사용하여 IBM MQ 자원에 액세스할 수 있습니다.

참고: 기존에 를 사용하는 IBM MQ classes for Java 애플리케이션은 계속해서 완전히 지원되지만, 새로운 애플리케이션은 를 사용해야 IBM MQ classes for Jakarta Messaging 합니다. 최근에 추가된 IBM MQ기능들, 예를 들어 비동기적 소비 및 자동 재연결은 에서 IBM MQ classes for Java사용할 수 없지만, 와 IBM MQ classes for JMS 에서 사용할 IBM MQ classes for Jakarta Messaging수 있습니다. 자세한 정보는 [86 페이지의 『IBM MQ classes for JMS를 사용해야 하는 이유가 무엇입니까?』](#) 및 [84 페이지의 『IBM MQ classes for Jakarta Messaging를 사용해야 하는 이유가 무엇입니까?』](#)의 내용을 참조하십시오.

참고:

Stabilized IBM은 IBM MQ classes for Java에 대해 추가적인 개선 계획이 없으며 IBM MQ 8.0에 제공된 레벨에서 기능적으로 안정되어 있습니다. IBM MQ classes for Java를 사용하는 기존 애플리케이션은 계속 완벽하게 지원되지만, 새 기능은 추가되지 않으며, 개선 요청은 거부됩니다. 완전히 지원이란 IBM MQ 시스템 요구사항의 변경에 따라 필요한 모든 변경사항과 함께 결함이 수정되는 것을 의미합니다.

IBM MQ classes for Java는 IMS에서 지원되지 않습니다.

IBM MQ classes for Java는 WebSphere Liberty에서 지원되지 않습니다. 이는 IBM MQ Liberty 메시징 기능 또는 일반 JCA 지원에서는 사용될 수 없습니다. 자세한 정보는 [J2EE/JEE 환경에서 WebSphere MQ Java 인터페이스 사용](#)을 참조하십시오.

관련 개념

[87 페이지의 『IBM MQ - Java API 선택』](#)

IBM MQ 세 가지 Java 언어 인터페이스를 제공합니다.

IBM MQ classes for Java의 전제조건

IBM MQ classes for Java를 사용하려면 기타 특정 소프트웨어 제품이 필요합니다.

IBM MQ classes for Java의 필수조건에 대한 정보는 [시스템 요구 사항 IBM MQ 웹 페이지](#)를 참조하십시오.

IBM MQ classes for Java 애플리케이션을 개발하려면 JDK(Java Development Kit)가 필요합니다. 운영 체제에서 지원되는 JDK의 자세한 내용은 **시스템 요구 사항 IBM MQ** 정보에 있습니다.

IBM MQ classes for Java 애플리케이션을 실행하려면 다음 소프트웨어 컴포넌트가 필요합니다.

- 큐 관리자에 연결하는 애플리케이션의 경우, IBM MQ 큐 관리자
- A Java runtime environment (JRE), 애플리케이션을 실행하는 각 시스템마다. 적합한 가 와 JRE 함께 IBM MQ제공됩니다.
-  IBM i의 경우 운영 체제의 옵션 30인 QShell.
-  z/OS에 대하여, z/OS UNIX System Services (z/OS UNIX)

인증된 FIPS 140-2 암호화 모듈을 사용하기 위해 TLS 연결이 필요한 경우, JSSE FIPS 공급자(IBMIBMJava JSSEFIPS)가 필요합니다. 모든 IBM JDK 및 버전 1.4.2JRE 이상의 JDK에는 IBMJSSEFIPS가 포함됩니다.

 인증된 FIPS 140-3 암호화 모듈을 사용하기 위해 TLS 연결이 필요한 경우, 보안 공급자 (OpenJCEPlusFIPSIBM Semeru RuntimeJava)가 필요합니다.

가상 머신 Java (JVM)과 운영 체제의 구현 TCP/IP 환경에서 지원하는 애플리케이션에서 만약 IPv6 라면IBM MQ classes for Java 버전 6(IPv6) 주소를 사용할 Internet Protocol 수 있습니다.

애플리케이션 IBM MQ classes for Java 실행 Java EE

사용하기 IBM MQ classes for Java 전에 반드시 고려해야 할 특정 제한 Java EE 사항과 설계 고려 사항이 있습니다.

IBM MQ classes for Java (Java EE) Java Platform, Enterprise Edition 환경 내에서 사용 시 제한 사항이 있습니다. 애플리케이션이 IBM MQ classes for Java 환경 Java EE 내에서 실행되도록 설계, 구현 및 관리할 때 고려해야 할 추가 사항들도 있습니다. 이러한 제한사항과 고려사항은 다음 섹션에 간략하게 설명되어 있습니다.

JTA 트랜잭션 제한

IBM MQ classes for Java를 사용하는 애플리케이션에 지원되는 유일한 트랜잭션 관리자는 IBM MQ 자체입니다. JTA가 제어하는 애플리케이션이 IBM MQ classes for Java를 사용할 수 있지만 이러한 클래스를 통해 수행한 작업은 JTA 작업 단위를 통해 제어하지 않습니다. 대신 JTA 인터페이스를 통해 애플리케이션 서버에서 관리하는 작업 단위와 별개의 논리 작업 단위를 형성합니다. 특히 JTA 트랜잭션을 롤백해도 송신하거나 수신한 메시지를 롤백하지 않습니다. 이 제한사항은 애플리케이션이나 Bean 관리 트랜잭션 및 컨테이너 관리 트랜잭션과 모든 Java EE 컨테이너에 적용됩니다. 애플리케이션 서버 통합 트랜잭션 내부에서 IBM MQ와 직접 메시징 작업을 수행하려면 IBM MQ classes for JMS를 대신 사용해야 합니다.

스레드 작성

IBM MQ classes for Java에서는 다양한 조작을 위해 내부적으로 스레드를 작성합니다. 예를 들어 로컬 큐 관리자에서 직접 호출하기 위해 BINDINGS 모드에서 실행할 때 IBM MQ classes for Java에서 내부적으로 작성한 '작업 프로그램' 스레드에서 호출합니다. 예를 들어, 연결 풀에서 사용하지 않은 연결을 지우거나 종료된 발행/구독 애플리케이션의 구독을 제거하기 위해 내부적으로 다른 스레드를 작성할 수 있습니다.

일부 Java EE 애플리케이션(예: EJB 및 웹 컨테이너에서 실행 중인 애플리케이션)은 새 스레드를 작성하지 않아야 합니다. 대신, 애플리케이션 서버에서 관리하는 기본 애플리케이션 스레드에서 모든 작업을 수행해야 합니다. 애플리케이션에서 IBM MQ classes for Java를 사용할 때 애플리케이션 서버에서 애플리케이션 코드와 IBM MQ classes for Java 코드를 구분할 수 없으므로 이전에 설명한 스레드로 인해 애플리케이션이 컨테이너 스펙을 준수하지 않게 됩니다. IBM MQ classes for JMS는 이러한 Java EE 스펙을 위반하지 않으므로 대신 사용할 수 있습니다.

보안 제한사항

애플리케이션 서버에서 구현한 보안 정책으로 인해 IBM MQ classes for Java API에서 수행하는 특정 조작(예: 이전 섹션에 설명되어 있는 새로운 제어 스레드 작성 및 운영)이 방지됩니다.

예를 들어, 애플리케이션 서버는 대개 Java 보안을 기본적으로 사용하지 않게 설정하여 실행하며 이 보안은 애플리케이션 서버 특정 구성을 통해 사용으로 설정할 수 있습니다(일부 애플리케이션 서버에서는 Java 보안에서 사

용한 정책을 더욱 자세하게 구성할 수 있음). Java 보안을 사용할 때 IBM MQ classes for Java에서 애플리케이션 서버에 정의된 Java 보안 정책 스레드 작성 규칙을 위반할 수 있으며 API가 작동하는 데 필요한 모든 스레드를 작성하지 못할 수 있습니다. 스레드 관리 문제를 방지하기 위해, 보안이 활성화된 Java 환경에서는 의 IBM MQ classes for Java 사용이 지원되지 않습니다.

애플리케이션 격리 고려사항

Java EE 환경에서 애플리케이션을 실행하면 애플리케이션을 격리하는 이점이 있습니다. 설계와 구현은 환경보다 IBM MQ classes for JavaJava EE 먼저 이루어졌다. IBM MQ classes for Java는 애플리케이션 격리 개념을 지원하지 않는 방식으로 사용할 수 있습니다. 이 영역의 특정 고려사항은 예를 들어 다음과 같습니다.

- MQEnvironment 클래스에서 정적(JVM 프로세스 전체) 설정 사용. 예를 들어 다음과 같습니다.

- 연결 식별 및 인증에 사용할 사용자 ID 및 비밀번호
- 클라이언트 연결에 사용하는 호스트 이름, 포트 및 채널
- 보안 클라이언트 연결을 위한 TLS 구성

한 애플리케이션의 이점을 위해 MQEnvironment 특성을 수정하면 동일한 특성을 사용하는 다른 애플리케이션에도 영향을 미칩니다. Java EE 등의 다중 애플리케이션 환경에서 실행할 때 각 애플리케이션에서 프로세스 전체 MQEnvironment 클래스에 구성된 특성으로 기본값을 지정하지 않고 특정 특성 세트가 있는 MQQueueManager 오브젝트를 작성하여 고유한 구성을 사용해야 합니다.

- MQEnvironment 클래스에서는 동일한 JVM 프로세스에서 IBM MQ classes for Java를 사용하여 모든 애플리케이션에서 글로벌하게 작동하는 여러 정적 메소드를 소개하고 특정 애플리케이션에서 이 동작을 대체하는 방법이 없습니다. 예를 들면 다음과 같습니다.

- TLS 특성 구성(예: 키 저장소 위치)
- 클라이언트 채널 엑시트 구성
- 진단 추적 사용 또는 사용 안함
- 큐 관리자에 대한 연결 사용을 최적화하는 데 사용하는 기본 연결 풀 관리

이러한 메소드를 호출하면 동일한 Java EE 환경에서 실행 중인 모든 애플리케이션에 영향을 미칩니다.

- 동일한 큐 관리자에 다수 연결하는 프로세스를 최적화하기 위해 연결 풀을 사용합니다. 기본 연결 풀 관리자는 프로세스 전체에 해당하며 여러 애플리케이션에서 공유합니다.

MQEnvironment.setDefaultConnectionFactory() 메소드를 사용하여 한 애플리케이션의 기본 연결 관리자를 바꾸는 등의 연결 풀 구성을 변경하므로 동일한 Java EE 애플리케이션 서버에서 실행 중인 다른 애플리케이션에 영향을 미칩니다.

- TLS는 MQEnvironment 클래스와 MQQueueManager 오브젝트 특성을 사용하여 IBM MQ classes for Java를 사용하는 애플리케이션에 구성됩니다. 애플리케이션 서버 자체의 관리 보안 구성과 통합되지 않습니다. 애플리케이션 서버 구성을 사용하지 않고 필요한 보안 레벨을 제공하도록 IBM MQ classes for Java를 적절하게 구성해야 합니다.

바인딩 모드 제한

IBM MQ 큐 매니저와 함께 제공되는 리소스 IBM MQ 어댑터(WebSphere Application Server RA)의 주요 버전이 서로 다른 경우에도 동일한 머신에 설치할 수 있습니다 WebSphere Application Server .

큐 관리자와 자원 어댑터 주요 버전이 다르면 바인딩 연결을 사용할 수 없습니다. 자원 어댑터를 사용하여 WebSphere Application Server에서 큐 관리자에 연결할 때 클라이언트 유형 연결을 사용해야 합니다. 버전이 같으면 바인딩 연결을 사용할 수 있습니다.

IBM MQ classes for Java에서 문자열 변환

IBM MQ classes for Java에서는 문자 문자열 변환을 위해 CharsetEncoders 및 CharsetDecoders를 직접 사용합니다. 문자 문자열 변환을 위한 기본 동작은 두 개의 시스템 특성으로 구성할 수 있습니다. 맵핑할 수 없는 문자가 포함되는 메시지 처리는 com.ibm.mq.MQMD를 통해 구성할 수 있습니다.

IBM MQ 8.0 이전에는 IBM MQ classes for Java의 문자열 변환이 java.nio.charset.Charset.decode(ByteBuffer) 및 Charset.encode(CharBuffer) 메소드를 호출하여 수행되었습니다.

이러한 방법 중 하나를 사용하면 형식이 잘못되거나 변환 불가능한 데이터의 기본 대체(REPLACE)가 발생합니다. 이 작동으로 인해 애플리케이션의 오류가 모호해질 수 있으며, 변환된 데이터에서 예상치 못한 문자(예: ?)가 발생할 수 있습니다.

IBM MQ 8.0부터는, 이러한 문제를 더 빠르고 효율적으로 감지하기 위해 IBM MQ classes for Java이(가) CharsetEncoders 및 CharsetDecoders를 직접 사용하여 잘못된 형식의 데이터 및 변환 불가능한 데이터의 처리를 명시적으로 구성합니다. 기본 동작은 적합한 MQException을 발생시켜 이러한 문제를 REPORT하는 것입니다.

구성

UTF-16(Java에서 사용되는 문자 표현)에서 고유 문자 세트(예: UTF-8)로의 변환을 *encoding*이라고 하고 반대 방향으로의 변환을 *decoding*이라고 합니다.

코드 번역은 CharsetDecoders의 기본 동작을 수행하며 예외를 발생시켜 오류를 보고합니다.

하나의 설정을 사용하여 인코딩 및 디코딩 모두에서 오류 핸들링을 제어하기 위한 `java.nio.charset.CodingErrorAction`을 지정합니다. 다른 설정은 인코딩 시에 대체 바이트를 제어하는 데 사용됩니다. 기본 Java 대체 문자열은 디코딩 조작에서 사용됩니다.

IBM MQ classes for Java에서 번역할 수 없는 데이터 처리 구성

IBM MQ 8.0부터 `com.ibm.mq.MQMD`에는 다음 두 필드가 포함됩니다.

`byte[] unMappableReplacement`

입력 문자를 변환할 수 없으며 REPLACE를 지정한 경우에 인코딩된 문자열에 쓰여질 바이트 시퀀스입니다.

기본값: `"?".getBytes()`

기본 Java 대체 문자열이 디코딩 조작에서 사용됩니다.

`java.nio.charset.CodingErrorAction unMappableAction`

인코딩 및 디코딩에서 변환 불가능한 데이터에 대해 취하는 조치를 지정합니다.

기본값: `CodingErrorAction.REPORT;`

시스템 기본값 설정을 위한 시스템 특성

IBM MQ 8.0부터 문자열 변환과 관련된 기본 동작을 구성하는 데 다음 두 가지 Java 시스템 특성을 사용할 수 있습니다.

`com.ibm.mq.cfg.jmqi.UnmappableCharacterAction`

인코딩 및 디코딩에서 변환 불가능한 데이터에 대해 취하는 조치를 지정합니다. 값은 REPORT, REPLACE 또는 IGNORE일 수 있습니다.

`com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement`

문자가 인코딩 조작에서 맵핑될 수 없을 때 적용할 대체 바이트를 설정하거나 가져옵니다. 기본 Java 대체 문자열이 디코딩 조작에서 사용됩니다.

Java 문자 및 고유 바이트 표현 간의 혼란을 피하려면, 고유 문자 세트에서 대체 바이트를 표시하는 10진수로 `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement`를 지정해야 합니다.

예를 들어, 고유 문자 세트가 ASCII 기반(예: ISO-8859-1)인 경우에 고유 바이트로서 ?의 10진수는 63입니다. 한편 고유 문자 세트가 EBCDIC인 경우에 이는 111입니다.

참고: MQMD 또는 MQMessage 오브젝트에서 `unMappableAction` 또는 `unMappableReplacement` 필드가 설정된 경우 이 필드 값이 Java 시스템 특성보다 우선됩니다. 이를 통해 필요한 경우 각 메시지에 대해 Java 시스템 특성에서 지정한 값을 대체할 수 있습니다.

관련 개념

[135 페이지의 『IBM MQ classes for JMS에서 문자열 변환』](#)

IBM MQ classes for JMS에서는 문자 문자열 변환을 위해 CharsetEncoders 및 CharsetDecoders를 직접 사용합니다. 문자 문자열 변환을 위한 기본 동작은 두 개의 시스템 특성으로 구성할 수 있습니다. 맵핑할 수 없는 문자가 포함되는 메시지 처리는 UnmappableCharacterAction 및 대체 바이트를 설정하기 위한 메시지 특성을 통해 구성할 수 있습니다.

IBM MQ classes for Java 설치 및 구성

이 섹션에서는 IBM MQ classes for Java를 설치할 때 작성한 디렉토리와 파일을 설명하고 설치 후에 IBM MQ classes for Java를 구성하는 방법을 알려줍니다.

IBM MQ classes for Java에 설치된 항목

최신 버전의 IBM MQ classes for Java는 IBM MQ와 함께 설치됩니다. 이 설치를 완료하려면 기본 설치 옵션을 대체해야 할 수도 있습니다.

IBM MQ 설치에 대한 자세한 정보는 다음을 참조하십시오.

- ▶ **Multi** [IBM MQ 설치](#)
- ▶ **z/OS** [IBM MQ for z/OS 제품 설치](#)

IBM MQ classes for Java은(는) JAR(Java Archive) 파일, com.ibm.mq.jar 및 com.ibm.mq.jmqi.jar에 포함되어 있습니다.

프로그래밍 가능 명령 형식(PCF)과 같은 표준 메시지 헤더 지원은 JAR 파일 com.ibm.mq.headers.jar에 포함되어 있습니다.

프로그래밍 가능 명령 형식(PCF)에 대한 지원은 JAR 파일 com.ibm.mq.pcf.jar에 포함되어 있습니다.

참고: 애플리케이션 서버에서 IBM MQ classes for Java를 사용하지 않는 것이 좋습니다. 이 환경에서 실행할 때 적용되는 제한사항에 대한 정보는 322 페이지의 『애플리케이션 IBM MQ classes for Java 실행 Java EE』의 내용을 참조하십시오. 자세한 내용은 J2EE/JEE 환경에서 WebSphere MQJava 인터페이스 사용을 참조하십시오.

중요사항: 325 페이지의 『IBM MQ classes for Java 재배포 가능 JAR 파일』에서 설명하는 재배포 가능 JAR 파일과 별개로 IBM MQ classes for Java JAR 파일이나 고유 라이브러리를 다른 시스템 또는 IBM MQ classes for Java가 설치된 시스템의 다른 위치에 복사하는 기능은 지원되지 않습니다.

IBM MQ classes for Java 재배포 가능 JAR 파일

재배포 가능 JAR 파일은 IBM MQ classes for Java 실행에 필요한 시스템으로 이동될 수 있습니다.

중요사항:

- 재배포 가능 JAR 파일에서 설명하는 재배포 가능 JAR 파일과 별개로 IBM MQ classes for Java JAR 파일이나 고유 라이브러리를 다른 시스템 또는 IBM MQ classes for Java가 설치된 시스템의 다른 위치에 복사하는 기능은 지원되지 않습니다.
- 클래스 로더 충돌을 방지하기 위해 동일한 Java 런타임 내 여러 애플리케이션에서 재배포 가능 JAR 파일을 번들로 제공하는 것은 권장되지 않습니다. 이 시나리오에서 IBM MQ 재배포 가능 JAR 파일을 Java 런타임의 클래스 경로에서 사용할 수 있도록 하는 방법을 고려하십시오.
- Java EE 애플리케이션 서버(예: WebSphere Application Server)에 배치된 애플리케이션 내 재배포 가능 JAR 파일을 포함하지 마십시오. 이러한 환경에서 IBM MQ 자원 어댑터가 대신 배치 및 사용됩니다. 여기에서는 IBM MQ classes for Java를 포함하기 때문입니다. WebSphere Application Server는 IBM MQ 자원 어댑터를 임베드하므로, 이 환경에 수동으로 배치하지 않아도 됩니다. 이외에도 IBM MQ classes for Java는 WebSphere Liberty에서 지원되지 않습니다. 자세한 내용은 을 참조하십시오 391 페이지의 『WebSphere Liberty 그리고 IBM MQ 리소스 어댑터』.
- 애플리케이션에서 재배포 가능 JAR 파일을 번들로 제공하는 경우 재배포 가능 JAR 파일에서 설명한 대로, 모든 필수 JAR 파일을 포함해야 합니다. 또한 IBM MQ classes for Java에서 최신 상태를 유지하고 알려진 문제를 수정할 수 있도록 애플리케이션 유지보수의 일부로, 번들로 제공된 JAR 파일을 업데이트하는 적절한 프로시저를 보유하고 있어야 합니다.

재배포 가능 JAR 파일

엔터프라이즈 내에서 다음 파일은 IBM MQ classes for Java 애플리케이션을 실행하는 데 필요한 시스템으로 이동할 수 있습니다.

- ▶ **JMS 2.0** [com.ibm.mq.allclient.jar](#) 326 페이지의 『1』

- **JM 3.0** `com.ibm.mq.jakarta.client.jar` [326 페이지의 『2』](#)
- `bcpkix-jdk18on.jar` [326 페이지의 『3』](#)
- `bcpkix-jdk15to18.jar` [326 페이지의 『4』](#)
- `bcprov-jdk18on.jar` [326 페이지의 『3』](#)
- `bcprov-jdk15to18.jar` [326 페이지의 『4』](#)
- `bcutil-jdk18on.jar` [326 페이지의 『3』](#)
- `bcutil-jdk15to18.jar` [326 페이지의 『4』](#)
- `org.json.jar`

참고:

1. JMS 2.0 및 JMS 1.1
2. [Jakarta Messaging 3.0](#)
3. 시작 IBM MQ 9.4.0
4. 이전 IBM MQ 9.4.0

바운시 캐슬 보안 제공업체 및 CMS 지원 JAR 파일

Bouncy Castle 보안 제공자 및 CMS 지원 JAR 파일이 필요합니다. 자세한 정보는 [AMS로 비IBM JRE에 대한 지원을 참조하십시오](#).

다음 JAR 파일이 필요합니다.

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

org.json.jar

이것은 JSON 형식으로 된 클라이언트 채널 정의 테이블(CCDT)을 IBM MQ classes for Java 파싱할 때 사용됩니다. 또한 객체의 JSON 표현을 생성하기 위해 `com.ibm.mq.MQQueueManager.toString()` 메서드가 호출될 때도 사용됩니다.

com.ibm.mq.allclient.jar 그리고 com.ibm.mq.jakarta.client.jar

파일 `com.ibm.mq.allclient.jar` 과 `com.ibm.mq.jakarta.client.jar` 에는 IBM MQ classes for Java IBM MQ classes for JMS, PCF 및 `com.ibm.mq.jakarta.client.jar` 헤더 클래스가 포함되어 있습니다. 이 파일들을 새 위치로 이동할 경우, 해당 새 위치에 새로운 IBM MQ 수정 패키지를 지속적으로 적용하여 유지 관리되도록 조치를 취해야 합니다. 또한 임시 해결책을 적용하는 경우 해당 파일 사용 사실을 IBM 지원팀에 반드시 알려주세요.

파일 `com.ibm.mq.allclient.jar` `com.ibm.mq.jakarta.client.jar` 또는 파일의 버전을 확인하려면 다음 명령을 사용하십시오:

```
JM 3.0
java -jar com.ibm.mq.jakarta.client.jar
```

```
JMS 2.0
java -jar com.ibm.mq.allclient.jar
```

다음 예제는 이 명령의 샘플 출력을 표시합니다.

```
C:\Program Files\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:      Java Message Service Client
Version:   9.3.0.0
Level:     p000-L140428.1
Build Type: Production
```

```

Location:  file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:      IBM MQ classes for Java Message Service
Version:   9.3.0.0
Level:     p000-L140428.1
Build Type: Production
Location:  file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:      IBM MQ JMS Provider
Version:   9.3.0.0
Level:     p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:  file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:      Common Services for Java Platform, Standard Edition
Version:   9.3.0.0
Level:     p000-L140428.1
Build Type: Production
Location:  file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

```

IBM MQ classes for Java의 설치 디렉토리

IBM MQ classes for Java 파일과 샘플은 플랫폼에 따라 다른 위치에 설치됩니다. (JRE) Java runtime environment의 위치는 설치된 플랫폼에 IBM MQ 따라 달라집니다.

IBM MQ classes for Java 파일의 설치 디렉토리

327 페이지의 표 49에서는 IBM MQ classes for Java 파일이 설치된 위치를 보여줍니다.

표 49. IBM MQ classes for Java 설치 디렉토리	
플랫폼	디렉토리
 AIX	<code>MQ_INSTALLATION_PATH/java/lib</code>
 IBM i	<code>/QIBM/ProdData/mqm/java/lib</code>
 Linux	<code>MQ_INSTALLATION_PATH/java/lib</code>
 Windows	<code>MQ_INSTALLATION_PATH\java\lib</code>
 z/OS	<code>MQ_INSTALLATION_PATH/mqm/V8R0M0/java /lib</code>

`MQ_INSTALLATION_PATH`은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

샘플의 설치 디렉토리

IVP(Installation Verification Program)와 같은 샘플 애플리케이션은 IBM MQ와 함께 제공됩니다. 327 페이지의 표 50에서는 샘플 애플리케이션이 설치된 위치를 보여줍니다. IBM MQ classes for Java 샘플은 `wmqjava` 서브디렉토리에 있습니다. PCF 샘플은 `pcf`라는 서브디렉토리에 있습니다.

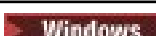
표 50. 샘플 디렉토리	
플랫폼	디렉토리
 AIX	<code>MQ_INSTALLATION_PATH/samp/wmqjava/</code>
 IBM i	<code>/QIBM/ProdData/mqm/java/samples</code>
 Linux	<code>MQ_INSTALLATION_PATH/samp/wmqjava/</code>
 Windows	<code>MQ_INSTALLATION_PATH\tools\wmqjava\</code>
 z/OS	<code>MQ_INSTALLATION_PATH/mqm/V8R0M0/java/samples</code>

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

JRE의 설치 디렉토리

Multi

이 기능은 (이상 IBM MQ classes for JMS) Java runtime environmentJava 7 버전이 필요합니다JRE. 적합한 버전이 JRE 설치되어 IBM MQ있습니다. 328 페이지의 표 51 이것이 JRE 설치된 위치를 보여줍니다. 제공된 샘플과 같은 프로그램을 Java 실행하려면, 이 JRE를 사용하여 명시적으로 를 JRE_LOCATION/bin/java 호출하거나 플랫폼의 PATH 환경 변수(또는 이에 상응하는 것)에 를 JRE_LOCATION/bin 추가하십시오. 328 페이지의 표 51여기서 JRE_LOCATION는 에 지정된 디렉터리입니다.

표 51. JRE 디렉토리	
플랫폼	디렉토리
 AIX	MQ_INSTALLATION_PATH/java/jre
 IBM i	/QIBM/ProdData/mqm/java/jre
 Linux	MQ_INSTALLATION_PATH/java/jre
 Windows	MQ_INSTALLATION_PATH\java\jre

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

IBM MQ classes for Java와 관련된 환경 변수

IBM MQ classes for Java 애플리케이션을 실행하려는 경우 클래스 경로에서 IBM MQ classes for Java 및 샘플 디렉토리를 포함해야 합니다.

실행할 IBM MQ classes for Java 애플리케이션의 경우 클래스 경로에는 적절한 IBM MQ classes for Java 디렉토리가 포함되어야 합니다. 샘플 애플리케이션을 실행하려면 클래스 경로에 적절한 샘플 디렉터리도 포함해야 합니다. 이 정보는 호출 명령어나 환경 Java **CLASSPATH** 변수를 통해 제공될 수 있습니다.

중요사항: IBM MQ classes for Java을(를) 포함하도록 Java 옵션 -Xbootclasspath을(를) 설정하는 것은 지원되지 않습니다.

328 페이지의 표 52 각 플랫폼에서 애플리케이션(샘플 애플리케이션 포함)을 IBM MQ classes for Java 실행하는 데 사용해야 할 적절한 **CLASSPATH** 설정을 보여줍니다.

표 52. CLASSPATH 샘플 IBM MQ classes for Java 애플리케이션을 포함한 애플리케이션을 IBM MQ classes for Java 실행하도록 설정	
플랫폼	CLASSPATH 설정
 AIX	CLASSPATH= MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jar: MQ_INSTALLATION_PATH/samp/wmqjava/samples:
 IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/samples/wmqjava/samples:
 Linux	CLASSPATH= MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jar: MQ_INSTALLATION_PATH/samp/wmqjava/samples:
 Windows	CLASSPATH= MQ_INSTALLATION_PATH\Java\lib\com.ibm.mq.jar; MQ_INSTALLATION_PATH\tools\wmqjava\samples;

표 52. **CLASSPATH** 샘플 IBM MQ classes for Java 애플리케이션을 포함한 애플리케이션을 IBM MQ classes for Java 실행하도록 설정 (계속)

플랫폼	CLASSPATH 설정
 z/OS	CLASSPATH= MQ_INSTALLATION_PATH/mqm/V10R0M0 /java/lib/com.ibm.mq.jar: MQ_INSTALLATION_PATH/mqm//java/samples/wmqjavaV10R0M0: MQ_INSTALLATION_PATH/mqm//java/samples/pcfV10R0M0

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

-Xlint 옵션을 사용하여 컴파일하는 경우 com.ibm.mq.ese.jar이(가) 없음을 경고하는 메시지가 표시될 수 있습니다. 이 경고는 무시해도 됩니다. 이 파일은 Advanced Message Security를 설치한 경우에만 제공됩니다.

IBM MQ classes for JMS와 함께 제공되는 스크립트는 다음 환경 변수를 사용하십시오.

MQ JAVA DATA_PATH

이 환경 변수는 로그 및 추적 출력을 위한 디렉토리를 지정합니다.

MQ JAVA INSTALL_PATH

이 환경 변수는 설치 IBM MQ classes for Java 디렉터리에서 표시된 대로 설치된 IBM MQ classes for Java 파일의 디렉토리를 지정합니다.

MQ JAVA LIB_PATH

이 환경 변수는 각 플랫폼별 라이브러리 IBM MQ classes for Java 위치에 표시된 대로 IBM MQ classes for Java 라이브러리가 저장된 디렉토리를 지정합니다. IBM MQ classes for Java에서 제공하는 일부 스크립트 (예: IVTRun)는 이 환경 변수를 사용합니다.

 Windows에서 모든 환경 변수는 설치 중 자동으로 설정됩니다.

 Linux  AIX and Linux에서는 setjmsenv(32비트 JVM을 사용하는 경우) 또는 setjmsenv64(64비트 JVM을 사용하는 경우) 스크립트를 사용하여 환경 변수를 설정할 수 있습니다. 이러한 스크립트는 MQ_INSTALLATION_PATH/java/bin 디렉토리에 있습니다.

 IBM i 날짜에 환경 변수는 Y 로 IBM i **QIBM_MULTI_THREADED** 설정되어야 합니다. 이렇게 하면 단일 스레드된 애플리케이션을 실행하는 것과 동일한 방식으로 멀티스레드된 애플리케이션을 실행할 수 있습니다. 자세한 내용은 [JMS를 사용한 Java IBM MQ 설정에서](#) 참조하십시오.

IBM MQ classes for Java 7이 Java 필요합니다 Java runtime environment (JRE). 설치된 적합한 위치에 대한 JRE 정보는 을 참조하십시오 [327 페이지의 『IBM MQ classes for Java의 설치 디렉토리』 IBM MQ.](#)

IBM MQ classes for Java 라이브러리

IBM MQ classes for Java 라이브러리의 위치는 플랫폼에 따라 다릅니다. 애플리케이션을 시작할 때 이 위치를 지정하십시오.

JNI(Java Native Interface) 라이브러리의 위치를 지정하려면 다음 형식의 **java** 명령을 사용하여 애플리케이션을 시작하십시오.

```
java -Djava.library.path=library_path application_name
```

여기서 *library_path*는 JNI 라이브러리의 경로입니다. [330 페이지의 표 53](#)에서는 각 플랫폼의 IBM MQ classes for Java 라이브러리 위치를 표시합니다. 이 표에서 MQ_INSTALLATION_PATH는 IBM MQ가 설치된 상위 레벨 디렉토리를 나타냅니다.

표 53. 각 플랫폼별 및 JNI 라이브러리의 IBM MQ classes for Java 위치		
플랫폼	IBM MQ classes for Java 라이브러리 디렉터리	64비트 JNI 라이브러리 디렉터리
 AIX  Linux (POWER, x86-64 및 s390x 플랫폼)	MQ_INSTALLATION_PATH/java/lib	MQ_INSTALLATION_PATH/java/lib64
 Windows	MQ_INSTALLATION_PATH\java\lib	MQ_INSTALLATION_PATH\java\lib64
 z/OS	MQ_INSTALLATION_PATH/mqm/버전_번호/java/lib, 여기서 버전_번호는 VRM 형식의 버전 번호입니다. 예를 들어, V10R0M0입니다.	31비트 JNI 라이브러리와 동일합니다

참고:

1. 에서 Windows, **java** 명령어에서 라이브러리 위치를 지정하는 대신 PATH IBM MQ classes for Java 환경 변수를 사용하여 라이브러리 위치를 지정할 수 있습니다.
2.  바인딩 모드에서 IBM MQ classes for Java 사용하려면 라이브러리 목록에 QMQMJAVA IBM i 라이브러리가 포함되어 있는지 확인하십시오.
3.  z/OS에서는 31비트 또는 64비트 Java 가상 머신 (JVM)을 사용할 수 있습니다. 사용할 라이브러리를 지정하지 않아도 됩니다. IBM MQ classes for Java 자체에서 로드할 JNI 라이브러리를 판별합니다.

관련 개념

IBM MQ classes for Java 사용

IBM MQ classes for Java를 설치한 후, 자체 애플리케이션을 실행하도록 설치를 구성할 수 있습니다.

IBM MQ classes for Java를 사용하여 OSGi 지원

OSGi에서는 애플리케이션 배치를 번들로 지원하는 프레임워크를 제공합니다. IBM MQ classes for Java의 일부로 3개의 OSGi 번들이 제공됩니다.

OSGi는 번들 양식으로 제공되는 애플리케이션 배치를 지원하는 안전한 범용 관리 Java 프레임워크를 제공합니다. OSGi 준수 디바이스는 번들을 다운로드하고 설치할 수 있으며 번들이 더 이상 필요하지 않으면 삭제할 수 있습니다. 프레임워크는 동적이고 확장 가능한 방식으로 번들의 설치 및 업데이트를 관리합니다.

IBM MQ classes for Java에는 다음 OSGi 번들이 포함됩니다.

com.ibm.mq.osgi.java.version_number.jar

애플리케이션에서 IBM MQ classes for Java를 사용하도록 허용하는 JAR 파일입니다.

com.ibm.mq.jakarta.osgi.allclient_version_number.jar

 이 JAR Jakarta Messaging 3.0 파일은 애플리케이션이 와 IBM MQ classes for JMS 를 모두 사용할 IBM MQ classes for Java수 있도록 하며, PCF 메시지를 처리하는 코드도 포함합니다.

com.ibm.mq.osgi.allclient_version_number.jar

 이 JAR JMS 2.0 파일은 애플리케이션이 와 IBM MQ classes for JMS 를 모두 사용할 IBM MQ classes for Java수 있도록 하며, PCF 메시지를 처리하는 코드도 포함합니다.

com.ibm.mq.jakarta.osgi.allclientprereqs_version_number.jar

 이 JAR 파일은 에 Jakarta Messaging 3.0 필요한 전제 com.ibm.mq.jakarta.osgi.allclient_version_number.jar조건을 제공합니다.

com.ibm.mq.osgi.allclientprereqs_version_number.jar

 이 JAR 파일은 에 JMS 2.0 필요한 전제 com.ibm.mq.osgi.allclient_version_number.jar조건을 제공합니다.

설치된 의 IBM MQ 버전 version_number 번호입니다.

번들은 IBM MQ 설치의 java/lib/OSGi 서브디렉토리 또는 Windows의 java\lib\OSGi 폴더에 설치됩니다.

IBM MQ 8.0부터는 모든 새 애플리케이션에 com.ibm.mq.osgi.allclient_8.0.0.0.jar 및 com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar 번들을 사용하십시오. 이러한 번들을 사용하면 동일한 OSGi 프레임워크에서 IBM MQ classes for JMS와 IBM MQ classes for Java를 둘 다 실행할 수 없는 제한사항이 제거됩니다. 그러나 다른 제한사항은 여전히 모두 적용됩니다. 버전 이전의 IBM MQ 경우 IBM MQ 8.0, 또는 중 IBM MQ classes for JMS 하나만 사용해야 IBM MQ classes for Java 한다는 제한 사항이 적용됩니다.

다른 9개의 번들도 IBM MQ 설치의 java/lib/OSGi 서브디렉토리 또는 Windows의 java\lib\OSGi 폴더에 설치됩니다. 이러한 번들은 IBM MQ classes for JMS의 일부이며 IBM MQ classes for Java 번들을 로드한 OSGi 런타임 환경에 로드하지 않아야 합니다. OSGi IBM MQ classes for Java 번들이 다른 IBM MQ classes for JMS 번들(bundles)도 로드된 OSGi 런타임 환경에 로드된 경우, 해당 IBM MQ classes for Java 번들(bundle) 또는 번들들 IBM MQ classes for JMS (bundles)을 사용하는 애플리케이션을 실행하면 다음과 같은 예시에서 보여지는 오류가 발생합니다:

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

IBM MQ classes for Java의 OSGi 번들은 OSGi 릴리스 4 스펙에 기록되어 있습니다. OSGi 릴리스 3 환경에서 작동하지 않습니다.

OSGi 런타임 환경이 필수 DLL 파일 또는 공유 라이브러리를 찾을 수 있도록 시스템 경로 또는 라이브러리 경로를 올바르게 설정해야 합니다.

IBM MQ classes for Java의 OSGi 번들을 사용하는 경우 OSGi와 같은 다중 클래스 로더 환경에서 클래스를 로드하는 데 관한 내재적인 문제점으로 인해 Java에 작성된 채널 엑시트 클래스도 지원되지 않습니다. 사용자 번들에서는 IBM MQ classes for Java 번들을 인지할 수 있지만 IBM MQ classes for Java 번들은 사용자 번들을 인지하지 못합니다. 결과적으로 IBM MQ classes for Java 번들에서 사용된 클래스 로더는 사용자 번들에 있는 채널 엑시트 클래스를 로드할 수 없습니다.

OSGi에 대한 자세한 정보는 [OSGi alliance](#) 웹 사이트를 참조하십시오.

'IBM MQ classes for Java'을 'z/OS'에 설치하기

z/OS에서 런타임 시 사용된 STEPLIB에는 IBM MQ SCSQAUTH 및 SCSQANLE 라이브러리가 포함되어야 합니다.

다음 예시와 같이 'z/OS UNIX System Services'에서 '.profile'의 한 줄을 사용하여 'thlqual'을 'IBM MQ'를 설치할 때 선택한 상위 데이터 세트 한정자로 대체하여 이러한 라이브러리를 추가할 수 있습니다:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

다른 환경에서는 일반적으로 STEPLIB 연결에 SCSQAUTH를 포함하도록 시동 JCL을 편집해야 합니다.

```
STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
         DD DSN=thlqual.SCSQANLE,DISP=SHR
```

IBM MQ classes for Java 구성 파일

IBM MQ classes for Java 구성 파일은 IBM MQ classes for Java 구성에 사용되는 특성을 지정합니다.

IBM MQ classes for Java 구성 파일의 형식은 표준 Java 특성 파일의 형식입니다.

샘플 구성 파일 mqjava.config은(는) IBM MQ classes for Java 설치 디렉토리의 bin 서브디렉토리에 제공됩니다. 이 파일은 지원되는 모든 특성 및 기본값을 문서화합니다.

참고: 샘플 구성 파일은 IBM MQ 설치가 차후 수정팩으로 업그레이드될 때 겹쳐 씁니다. 따라서 애플리케이션에 사용할 샘플 구성 파일의 복사본을 만드는 것이 좋습니다.

IBM MQ classes for Java 구성 파일의 이름 및 위치를 선택할 수 있습니다. 애플리케이션을 시작할 때 다음 형식의 **java** 명령을 사용하십시오.

```
java -Dcom.ibm.msg.client.config.location=config_file_url application_name
```

명령에서 `config_file_url`은 IBM MQ classes for Java 구성 파일의 이름 및 위치를 지정하는 URL(Uniform Resource Locator)입니다. `http`, `file`, `ftp` 및 `jar` 유형의 URL이 지원됩니다.

다음 예에서는 **java** 명령을 보여줍니다.

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/mqjava.config MyAppClass
```

이 명령은 IBM MQ classes for Java 구성 파일을 로컬 Windows 시스템의 `D:\mydir\mqjava.config` 파일로 식별합니다.

애플리케이션이 시작될 때 IBM MQ classes for Java는 구성 파일의 콘텐츠를 읽고 지정된 특성을 내부 특성 저장소에 저장합니다. **java** 명령이 구성 파일을 식별하지 않거나 구성 파일을 찾을 수 없으면 IBM MQ classes for Java에서 모든 특성의 기본값을 사용합니다. 필요한 경우, **java** 명령에서 시스템 특성으로 지정함으로써 구성 파일의 특성을 대체할 수 있습니다.

IBM MQ classes for Java 구성 파일은 애플리케이션 및 큐 관리자 또는 브로커 간에 지원되는 전송에서 사용될 수 있습니다.

IBM MQ MQI client 구성 파일에 지정된 특성 대체

IBM MQ MQI client 구성 파일은 IBM MQ classes for Java의 구성에 사용되는 특성을 지정할 수도 있습니다. 그러나 IBM MQ MQI client 구성 파일에 지정된 특성은 애플리케이션이 클라이언트 모드로 큐 관리자에 연결될 때만 적용됩니다.

필요한 경우, 구성 IBM MQ MQI client 파일에서 속성을 지정하여 구성 IBM MQ classes for Java 파일의 속성으로 재정의할 수 있습니다. IBM MQ MQI client 구성 파일의 속성을 대체하려면, IBM MQ classes for Java 구성 파일에서 다음 형식의 항목을 사용하십시오.

```
com.ibm.mq.cfg.stanza.propName=propValue
```

항목의 변수는 다음을 의미합니다.

stanza

속성을 포함하는 IBM MQ MQI client 구성 파일의 스탠자 이름

propName

IBM MQ MQI client 구성 파일에 지정된 속성의 이름

propValue

IBM MQ MQI client 구성 파일에 지정된 속성 값을 대체하는 특성 값

또는 구성 IBM MQ MQI client 파일의 속성을 재정의하려면 해당 속성을 **java** 명령줄에서 시스템 속성으로 지정할 수 있습니다. 이전 형식을 사용하여 특성을 시스템 특성으로 지정하십시오.

IBM MQ MQI client 구성 파일의 다음 속성만 IBM MQ classes for Java와 관련됩니다. 기타 속성을 지정하거나 대체하는 경우 효과가 없습니다. 특히, 클라이언트 구성 파일의 CHANNELS 구역에 있는 `ChannelDefinitionFileChannelDefinitionDirectory` 와 는 사용되지 않음을 유의하십시오. CCDT 를 사용하는 방법에 대한 자세한 내용은 IBM MQ classes for Java 346 페이지의 『IBM MQ classes for Java 에서 클라이언트 채널 정의 테이블 사용』 참조하십시오.

표 54. 클라이언트 구성 파일의 스탠자에 포함되어 있는 속성	
스탠자	속성
클라이언트 구성 파일의 CHANNELS 스탠자	Put1DefaultAlwaysSync
클라이언트 구성 파일의 CHANNELS 스탠자	PasswordProtection
클라이언트 구성 파일의 ClientExitPath 스탠자	ExitsDefaultPath
클라이언트 구성 파일의 ClientExitPath 스탠자	ExitsDefaultPath64
클라이언트 구성 파일의 ClientExitPath 스탠자	JavaExitsClasspath
클라이언트 구성 파일의 JMQUI 스탠자	useMQCSPauthentication

표 54. 클라이언트 구성 파일의 스탠자에 포함되어 있는 속성 (계속)	
스탠자	속성
클라이언트 구성 파일의 MessageBuffer 스탠자	MaximumSize
클라이언트 구성 파일의 MessageBuffer 스탠자	PurgeTime
클라이언트 구성 파일의 MessageBuffer 스탠자	UpdatePercentage
클라이언트 구성 파일의 TCP 스탠자	CIntRcvBufSize
클라이언트 구성 파일의 TCP 스탠자	CIntSndBufSize
클라이언트 구성 파일의 TCP 스탠자	Connect_Timeout
클라이언트 구성 파일의 TCP 스탠자	KeepAlive

구성 IBM MQ MQI client 관련 자세한 내용은 을 참조하십시오 [IBM MQ MQI client 구성 파일, mqclient.ini](#).

관련 태스크

[Java 애플리케이션용 IBM MQ 클래스 추적](#)

Java [Java trace](#)에 대한 표준 환경 추적 스탠자

Java 표준 환경 추적 설정 스탠자를 사용하여 IBM MQ classes for Java 추적 기능을 구성할 수 있습니다.

com.ibm.msg.client.commonservices.trace.outputName = *traceOutputName*

V10.0.0 - V10.0.0의 *traceOutputName* 값은 추적 출력이 전송될 위치를 지정합니다:

stdout (또는 System.out)

추적 정보를 표준 출력 스트림으로 직접 출력합니다(대소문자 구분 없음).

stderr (또는 System.err)

출력을 표준 오류 스트림으로 직접 전송합니다(대소문자 구분 없음).

<file>

그 외의 값은 파일 경로로 간주되며, 추적 정보는 해당 파일로 전송됩니다.

기본적으로 추적 파일은 애플리케이션의 현재 작업 디렉토리에 있는 추적 파일에 기록됩니다. 추적 파일의 이름은 애플리케이션이 실행 중인 환경에 따라 다릅니다.

- 애플리케이션이 재배포 가능 JAR 파일에서 IBM MQ classes for Javacom.ibm.mq.allclient.jar 로드된 경우, 추적 정보는 라는 mqjavaclient_%PID%.cl%u.trc 파일에 기록됩니다.
- 애플리케이션이 JAR 파일에서 IBM MQ classes for Java 로드한 경우, 추적 com.ibm.mq.jar정보는 라는 mqjava_%PID%.cl%u.trc 파일에 기록됩니다.

여기서, %PID%는 추적하는 애플리케이션의 프로세스 ID이고, %u는 서로 다른 Java 클래스 로더에서 추적을 실행하는 스레드 간 파일을 구별하기 위한 고유한 숫자입니다.

프로세스 ID가 사용 불가능한 경우, 난수가 생성되며 문자 f로 접두부가 지정됩니다. 지정한 파일 이름에 프로세스 ID를 포함시키려면 문자열 %PID%를 사용하십시오.

대체 디렉토리를 지정하는 경우, 이는 존재해야 하며 사용자는 이 디렉토리에 대한 쓰기 권한이 있어야 합니다. 쓰기 권한이 없으면 추적 출력이 System.err에 쓰여집니다.

com.ibm.msg.client.commonservices.trace.include = *includeList*

*includeList*는 추적되는 패키지 및 클래스의 목록이거나 특수 값 ALL 또는 NONE입니다.

패키지 또는 클래스 이름을 세미콜론 ;으로 분리하십시오. *includeList*의 기본값은 ALL이며, IBM MQ classes for Java의 모든 패키지 및 클래스를 추적합니다.

참고: 패키지를 포함하지만 해당 패키지의 서브패키지를 제외시킬 수 있습니다. 예를 들어, 패키지 a.b는 포함하지만 패키지 a.b.x는 포함하지 않은 경우에 추적에는 a.b.y 및 a.b.z의 모든 것이 포함되지만 a.b.x 또는 a.b.x.1은 포함되지 않습니다.

com.ibm.msg.client.commonservices.trace.exclude = excludeList

*excludeList*는 추적되는 패키지 및 클래스의 목록이거나, 특수 값 ALL 또는 NONE입니다.

패키지 또는 클래스 이름을 세미콜론 ;으로 분리하십시오. *excludeList*의 기본값은 NONE이며, 따라서 IBM MQ classes for JMS의 어떤 패키지 및 클래스도 추적에서 제외되지 않습니다.

참고: 패키지를 제외하지만 해당 패키지의 서브패키지를 포함시킬 수 있습니다. 예를 들어, 패키지 a.b는 제외하지만 패키지 a.b.x는 포함하는 경우에 추적에는 a.b.x 및 a.b.x.1의 모든 것이 포함되지만 a.b.y 또는 a.b.z는 포함되지 않습니다.

동일 레벨에서 포함됨 및 제외됨 모두로서 지정된 임의의 패키지 또는 클래스가 포함됩니다.

com.ibm.msg.client.commonservices.trace.maxBytes = maxArrayBytes

*maxArrayBytes*는 바이트 배열에서 추적된 최대 바이트 수입니다.

*maxArrayBytes*가 양의 정수로 설정된 경우, 이는 추적 파일에 쓰여지는 바이트 배열의 바이트 수를 제한합니다. 이는 *maxArrayBytes* 쓰기 이후 바이트 배열을 자릅니다. *maxArrayBytes*를 설정하면 결과 추적 파일의 크기가 줄어들며, 애플리케이션의 성능에서 추적의 효과가 줄어듭니다.

이 특성에 대한 0의 값은 바이트 배열의 콘텐츠가 추적 파일에 송신되지 않음을 의미합니다.

기본값은 -1이며, 이는 추적 파일에 송신된 바이트 배열에서 바이트 수의 한계를 제거합니다.

com.ibm.msg.client.commonservices.trace.limit = maxTraceBytes

*maxTraceBytes*는 추적 출력 파일에 쓰여진 최대 바이트 수입니다.

*maxTraceBytes*는 *traceCycles*와 함께 작동됩니다. 작성된 추적 바이트 수가 한계에 근접하면 파일이 닫히며 새 추적 출력 파일이 시작됩니다.

0 값은 추적 출력 파일의 길이가 0임을 의미합니다. 기본값은 -1이며, 이는 추적 출력 파일에 쓰여지는 데이터의 양이 무제한임을 의미합니다.

com.ibm.msg.client.commonservices.trace.count = traceCycles

*traceCycles*는 순환되는 추적 출력 파일의 수입니다.

현재 추적 출력 파일이 *maxTraceBytes*에서 지정된 한계에 도달하면 파일이 닫힙니다. 추가 추적 출력은 시퀀스의 다음 추적 출력 파일에 쓰여집니다. 각각의 추적 출력 파일은 파일 이름에 추가된 숫자 접미부로 구분됩니다. 현재 또는 최신 추적 출력 파일은 mqjms.trc.0이며, 다음의 최신 추적 출력 파일은 mqjms.trc.1입니다. 보다 이전의 추적 파일은 한계에 도달할 때까지 동일한 번호 지정 패턴을 따릅니다.

*traceCycles*의 디폴트 값은 1입니다. *traceCycles*가 1인 경우, 현재 추적 출력 파일이 최대 크기에 도달할 때 파일이 닫히고 삭제됩니다. 동일한 이름의 새 추적 출력 파일이 시작됩니다. 따라서 한 번에 하나의 추적 출력 파일만이 존재합니다.

com.ibm.msg.client.commonservices.trace.parameter = traceParameters

*traceParameters*는 메소드 매개변수 및 리턴 값이 추적에 포함되는지 여부를 제어합니다.

*traceParameters*의 기본값은 TRUE입니다. *traceParameters*가 FALSE로 설정되면 메소드 서명만 추적됩니다.

com.ibm.msg.client.commonservices.trace.startup = startup

IBM MQ classes for Java의 초기화 단계가 있으며, 그 동안에 자원이 할당됩니다. 기본 추적 기능은 자원 할당 단계 중에 초기화됩니다.

*startup*이 TRUE로 설정되면 시동 추적이 사용됩니다. 추적 정보가 즉시 생성되며, 추적 기능 자체를 포함한 모든 컴포넌트의 설정이 포함됩니다. 시동 추적 정보를 사용하여 구성 문제점을 진단할 수 있습니다. 시동 추적 정보는 항상 System.err에 쓰여집니다.

*startup*의 기본값은 FALSE입니다.

*startup*은 초기화가 완료되기 전에 확인됩니다. 이러한 이유 때문에, Java 시스템 특성으로서 명령행에서 특성만 지정합니다. IBM MQ classes for Java 구성 파일에는 이를 지정하지 마십시오.

com.ibm.msg.client.commonservices.trace.compress = compressedTrace

*compressedTrace*를 TRUE로 설정하여 추적 출력을 압축할 수 있습니다.

*compressedTrace*의 기본값은 FALSE입니다.

*compressedTrace*가 TRUE로 설정되면 추적 출력이 압축됩니다. 기본 추적 출력 파일 이름의 확장자는 .trz입니다. 압축이 기본값인 FALSE로 설정되면, 파일의 확장자는 .trc이며 이는 압축되지 않음을 표시합니다. 그러나 추적 출력의 파일 이름이 *traceOutputName*에 지정되면 해당 이름이 대신 사용됩니다. 파일에는 접미부가 적용되지 않습니다.

압축 추적 출력은 압축되지 않은 추적 출력보다 크기가 작습니다. 입/출력이 적으므로 압축되지 않은 추적보다 빨리 기록될 수 있습니다. 압축된 추적은 압축되지 않은 추적보다 IBM MQ classes for Java의 성능에 영향을 덜 줍니다.

maxTraceBytes 및 *traceCycles*가 설정된 경우, 다중 압축 추적 파일이 다중 플랫폼 파일 대신에 작성됩니다.

IBM MQ classes for Java가 비제어 방식으로 종료되는 경우, 압축 추적 파일이 유효하지 않을 수 있습니다. 이러한 이유 때문에, 추적 압축은 IBM MQ classes for Java가 제어된 방식으로 닫힐 때만 사용되어야 합니다. 검사 중인 문제점 때문에 JVM 자체가 예상치 못하게 중지되지 않는 경우에만 추적 압축을 사용하십시오. System.Halt() 종료 또는 비정상적 제어되지 않은 JVM 종료를 발생시킬 수 있는 문제점을 진단 중인 경우에는 추적 압축을 사용하지 마십시오.

com.ibm.msg.client.commonservices.trace.level = *traceLevel*

*traceLevel*은 추적의 필터링 레벨을 지정합니다. 정의된 추적 레벨은 다음과 같습니다.

- TRACE_NONE: 0
- TRACE_EXCEPTION: 1
- TRACE_WARNING: 3
- TRACE_INFO: 6
- TRACE_ENTRYEXIT: 8
- TRACE_DATA: 9
- TRACE_ALL: Integer.MAX_VALUE

각 추적 레벨에는 모든 하위 레벨이 포함됩니다. 예를 들어, 추적 레벨이 TRACE_INFO에서 설정된 경우에 TRACE_EXCEPTION, TRACE_WARNING 또는 TRACE_INFO의 정의된 레벨이 있는 추적 지점이 추적에 쓰여집니다. 다른 모든 추적 지점은 제외됩니다.

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

*standaloneTrace*는 IBM MQ classes for Java 클라이언트 추적 서비스가 WebSphere Application Server 환경에서 사용되는지 여부를 제어합니다.

*standaloneTrace*가 TRUE로 설정된 경우에는 IBM MQ classes for Java 클라이언트 추적 특성을 사용하여 추적 구성을 판별합니다.

*standaloneTrace*가 FALSE로 설정되어 있으며 IBM MQ classes for Java 클라이언트가 WebSphere Application Server 컨테이너에서 실행 중인 경우에는 WebSphere Application Server 추적 서비스가 사용됩니다. 생성되는 추적 정보는 애플리케이션 서버의 추적 설정에 따라 다릅니다.

*standaloneTrace*의 기본값은 FALSE입니다.

IBM MQ classes for Java 및 소프트웨어 관리 도구

Apache Maven과 같은 소프트웨어 관리 도구를 IBM MQ classes for Java에서 사용할 수 있습니다.

많은 대형 개발 조직은 이러한 도구를 사용하여 써드파티 라이브러리의 저장소를 중앙 집중식으로 관리합니다.

IBM MQ classes for Java는 다수의 JAR 파일로 구성되어 있습니다. 이 API를 사용하여 Java 언어 애플리케이션을 개발 중인 경우에는 IBM MQ Server, IBM MQ Client 또는 IBM MQ Client SupportPac의 설치가 애플리케이션이 개발되는 시스템에서 필요합니다.

소프트웨어 관리 도구를 사용하고 IBM MQ classes for Java를 구성하는 JAR 파일을 중앙 관리되는 저장소에 추가하려면 다음과 같은 점을 관찰해야 합니다.

- 저장소 또는 컨테이너를 조직 내의 개발자만 사용할 수 있어야 합니다. 조직 외부의 배포는 허용되지 않습니다.

- 저장소는 단일 IBM MQ 릴리스 또는 수정팩에서 JAR 파일의 전체 일관된 세트를 포함해야 합니다.
- 사용자는 IBM 지원에서 제공하는 유지보수로 저장소를 업데이트해야 합니다.

IBM MQ 8.0부터는, `com.ibm.mq.allclient.jar` JAR 파일을 저장소에 설치해야 합니다.

IBM MQ 9.0에서는 Bouncy Castle 보안 제공자와 CMS 지원 JAR 파일이 필요합니다. 자세한 정보는 [325 페이지](#)의 『IBM MQ classes for Java 재배치 가능 JAR 파일』 및 [비IBM JRE 지원을 참조하십시오](#).

IBM MQ classes for Java 애플리케이션의 설치 후 설정

IBM MQ classes for Java를 설치한 후, 자체 애플리케이션을 실행하도록 설치를 구성할 수 있습니다.

IBM MQ 제품 readme 파일에서 최신 정보 또는 환경에 대한 더욱 구체적인 정보를 확인하십시오. 제품 readme 파일의 최신 버전은 [IBM MQ, WebSphere MQ, 및 MQSeries® 제품 사용 설명서](#) 웹 페이지에서 사용할 수 있습니다.

바인딩 모드에서 IBM MQ classes for Java 애플리케이션을 실행하기 전에, [구성](#) 항목에 설명된 대로 IBM MQ 설정을 완료했는지 확인하십시오.

IBM MQ classes for Java에서 클라이언트 연결을 승인하도록 큐 관리자 구성

클라이언트에서 수신되는 연결 요청을 승인하도록 큐 관리자를 구성하려면 서버 연결 채널의 사용을 정의하고 허용하며 리스너 프로그램을 시작하십시오.

자세한 내용은 [979 페이지](#)의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』의 내용을 참조하십시오.

아래에서 애플리케이션 IBM MQ classes for Java 실행 Java security manager

IBM MQ classes for Java는 Java security manager를 사용한 상태에서 실행할 수 있습니다. Java security manager를 사용한 상태로 애플리케이션을 실행하려면 적절한 정책 정의 파일로 JVM(Java Virtual Machine)을 구성해야 합니다.

적합한 정책 정의 파일을 생성하는 가장 간단한 방법은 (JREJava runtime environment)와 함께 제공되는 정책 파일을 수정하는 것입니다. 대부분의 시스템에서 이 파일은 사용자의 JRE 디렉터리 상대 경로로 `path lib/security/java.policy`. 정책 파일은 선호하는 편집기를 사용하거나, 제공된 `policytool` JRE 프로그램을 사용하여 편집할 수 있습니다.

다음을 수행할 수 있도록 `com.ibm.mq.jmqi.jar` 파일에 권한을 부여해야 합니다.

- 소켓 작성(클라이언트 모드)
- 고유 라이브러리 로드(바인딩 모드)
- 환경에서 당야한 특성 읽기

시스템 속성에서 실행될 `os.name` 때 IBM MQ classes for Java 에 Java security manager 반드시 사용 가능해야 합니다.

Java 애플리케이션이 Java security manager(를) 사용하는 경우에는 애플리케이션이 사용하는 `java.security.policy` 파일에 다음 권한을 추가해야 합니다. 그렇지 않으면 애플리케이션에 예외가 발생 합니다.

```
permission java.lang.RuntimePermission "modifyThread";
```

이 `RuntimePermission`은 큐 관리자에 대한 TCP/IP 연결을 통한 다중 대화의 지정 및 닫기 관리의 일부로서 클라이언트에 필요합니다.

예 정책 파일 항목

다음은 IBM MQ classes for Java가 기본 보안 관리자에서 성공적으로 실행되도록 허용하는 정책 파일 입력 항목의 예입니다. 이 예의 문자열 `MQ_INSTALLATION_PATH`를 시스템에서 IBM MQ classes for Java가 설치된 위치로 바꾸십시오.

```
grant codeBase "file: MQ_INSTALLATION_PATH/java/lib/*" {
//We need access to these properties, mainly for tracing
permission java.util.PropertyPermission "user.name","read";
permission java.util.PropertyPermission "os.name","read";
```

```

permission java.util.PropertyPermission "user.dir","read";
permission java.util.PropertyPermission "line.separator","read";
permission java.util.PropertyPermission "path.separator","read";
permission java.util.PropertyPermission "file.separator","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

//Tracing - we need the ability to control java.util.logging
permission java.util.logging.LoggingPermission "control";
// And access to create the trace file and read the log file - assumed to be in the current
directory
permission java.io.FilePermission "*", "read,write";

// Required to allow a trace file to be written to the filesystem.
// Replace 'TRACE_FILE_DIRECTORY' with the directory name where trace is to be written to
permission java.io.FilePermission "TRACE_FILE_DIRECTORY", "read,write";
permission java.io.FilePermission "TRACE_FILE_DIRECTORY/*", "read,write";

// We'd like to set up an mBean to control trace
permission javax.management.MBeanServerPermission "createMBeanServer";
permission javax.management.MBeanPermission "*", "*";

// We need to be able to read manifests etc from the jar files in the installation directory
permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-", "read";

//Required if mqclient.ini/mqs.ini configuration files are used
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini", "read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini", "read";

//For the client transport type.
permission java.net.SocketPermission "*", "connect,resolve";

//For the bindings transport type.
permission java.lang.RuntimePermission "loadLibrary.*";

//For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB", "read";

//For applications that use User Exits
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*", "read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*", "read";
permission java.lang.RuntimePermission "createClassLoader";

//Required for the z/OS platform
permission java.util.PropertyPermission "com.ibm.vm.bitmode", "read";

// Used by the internal ConnectionFactory implementation
permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

// Used for controlled class loading
permission java.lang.RuntimePermission "setContextClassLoader";

// Used to default the Application name in Client mode connections
permission java.util.PropertyPermission "sun.java.command", "read";

// Used by the IBM JSSE classes
permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace", "read";

//Required to determine if an IBM Java Runtime is running in FIPS mode,
//and to modify the property values status as required.
permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider", "read,write";
permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS", "read,write";
//Required if an IBM FIPS provider is to be used for SSL communication.
permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

// Required for non-IBM Java Runtimes that establish secure client
// transport mode connections using mutual TLS authentication
permission java.util.PropertyPermission "javax.net.ssl.keyStore", "read";
permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword", "read";

// Required for Java applications that use the Java Security Manager
permission java.lang.RuntimePermission "modifyThread";
};

```

이 정책 파일의 예를 사용하면 IBM MQ classes for Java가 보안 관리자에서 올바르게 작동할 수 있지만, 애플리케이션이 작동하기 전에 고유 코드를 올바르게 실행할 수 있어야 합니다.

IBM MQ classes for Java와 함께 제공되는 샘플 코드는 보안 관리자와 사용하도록 특별히 설정되지 않았습니다. 그러나 IVT 테스트는 이 정책 파일과 기본 보안 관리자를 사용하여 실행됩니다.

중요사항:

IBM MQ classes for Java 추적 기능은 시스템 특성에 대한 추가 조회를 수행하고 추가 파일 시스템 작업도 수행하므로 추가 권한이 필요합니다.

추적이 사용으로 설정된 보안 관리자에서 실행하기에 적합한 템플릿 보안 정책 파일은 IBM MQ 설치의 `samples/wmqjava` 디렉토리에 `example.security.policy`(으)로 제공됩니다.

기본 설치의 경우 `example.security.policy` 파일은 다음 위치에 있습니다.

Windows

`C:\Program Files\IBM\MQ\Tools\wmqjava\samples\example.security.policy`에

Linux

`/opt/mqm/samp/wmqjava/samples/example.security.policy`에

AIX

`/usr/mqm/samp/wmqjava/samples/example.security.policy`에

CICS Transaction Server에서 IBM MQ classes for Java 애플리케이션 실행

IBM MQ classes for Java 애플리케이션은 CICS Transaction Server에서 트랜잭션으로 실행될 수 있습니다.

IBM MQ classes for Java 애플리케이션을 z/OS용 CICS Transaction Server에서 트랜잭션으로 실행하려면 다음 단계를 수행하십시오.

1. 제공된 CEDA 트랜잭션을 사용하여 CICS에 애플리케이션과 트랜잭션을 정의합니다.
2. IBM MQ CICS 어댑터가 CICS 시스템에 설치되었는지 확인합니다.  (세부사항은 [CICS에서 IBM MQ 사용](#)을 참조하십시오.)
3. CICS에 지정된 JVM 환경에 적절한 CLASSPATH와 LIBPATH 항목을 포함되어 있는지 확인합니다.
4. 정상 프로세스를 사용하여 트랜잭션을 시작합니다.

CICS Java 트랜잭션 실행에 대한 자세한 정보는 CICS 시스템 문서를 참조하십시오.

IBM MQ classes for Java 설치 확인

설치 확인 프로그램, MQIVP는 IBM MQ classes for Java와 함께 제공됩니다. 이 프로그램을 사용하여 IBM MQ classes for Java의 연결 모드를 모두 테스트할 수 있습니다.

이 프로그램에서 확인할 연결 모드를 판별하기 위해 여러 선택 사항과 기타 데이터를 묻는 메시지를 표시합니다. 다음 프로시저를 사용하여 설치를 확인하십시오.

1. 클라이언트 모드로 프로그램을 실행할 경우, 예 설명된 대로 큐 979 페이지의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』 관리자를 구성하십시오. 사용할 큐는 `SYSTEM.DEFAULT.LOCAL.QUEUE`입니다
2. 클라이언트 모드에서 프로그램을 실행하려는 경우 320 페이지의 『IBM MQ classes for Java 사용』도 참조하십시오.

프로그램을 실행할 시스템에서 이 프로시저의 나머지 단계를 수행하십시오.

3. 328 페이지의 『IBM MQ classes for Java와 관련된 환경 변수』의 지시사항에 따라 CLASSPATH 환경 변수를 업데이트했는지 확인하십시오.
4. 디렉토리를 `MQ_INSTALLATION_PATH/mqm/samp/wmqjava/samples`(으)로 변경하십시오. 여기서 `MQ_INSTALLATION_PATH`은(는) IBM MQ 설치의 경로입니다. 그런 다음 명령 프롬프트에서 다음을 입력하십시오.

```
java -Djava.library.path= library_path MQIVP
```

여기서 `library_path`는 IBM MQ classes for Java 라이브러리의 경로입니다(329 페이지의 『IBM MQ classes for Java 라이브러리』 참조).

(1)로 표시된 프롬프트에서 다음을 수행하십시오.

- TCP/IP 연결을 사용하려면 IBM MQ 서버 호스트 이름을 입력하십시오.
- 고유 연결(바인딩 모드)을 사용하려면 필드를 공백으로 두십시오(이름을 입력하지 않음).

프로그램에서 다음을 시도합니다.

1. 큐 관리자에 연결합니다.
2. SYSTEM.DEFAULT.LOCAL.QUEUE 큐를 열고 큐에 메시지를 넣은 다음 큐에서 메시지를 가져오고 큐를 닫습니다.
3. 큐 관리자에서 연결을 끊습니다.
4. 조작에 성공하면 메시지를 리턴합니다.

다음은 표시될 수 있는 프롬프트와 응답의 예입니다. 실제 프롬프트와 응답은 IBM MQ 네트워크에 따라 다릅니다.

```
Please enter the IP address of the MQ server      : ipaddress(1)
Please enter the port to connect to               : (1414) (2)
Please enter the server connection channel name  : channelname (2)
Please enter the queue manager name              : qmname
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager
```

```
Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...
```

참고:

1.  z/OS에서는 ⁽¹⁾로 표시된 프롬프트에서 필드를 공백으로 두십시오.
2. 서버 연결을 선택하면 ⁽²⁾로 표시된 프롬프트가 표시되지 않습니다.
3.  IBM i에서는 QShell에서 `java MQIVP` 명령만 실행할 수 있습니다. 또는 CL 명령 `RUNJAVA CLASS(MQIVP)`를 사용하여 애플리케이션을 실행할 수 있습니다.

IBM MQ classes for Java 샘플 애플리케이션 사용

IBM MQ classes for Java 샘플 애플리케이션은 IBM MQ classes for Java API의 공통 기능 개요를 제공합니다. 이를 사용하여 설치 및 메시징 서버 설정을 확인하고 자신만의 애플리케이션을 빌드하는 데 도움을 받을 수 있습니다.

이 태스크 정보

자체 애플리케이션을 작성하는 데 도움이 필요하면 시작점으로서 샘플 애플리케이션을 사용할 수 있습니다. 각 애플리케이션에 대해 소스 버전과 컴파일 버전 모두가 제공됩니다. 샘플 소스 코드를 검토하고 애플리케이션에 대해 필요한 각 오브젝트(MQQueueManager, MQConstants, MQMessage, MQPutMessageOptions, MQDestination)를 작성하고 애플리케이션의 작동 방식을 지정하는 데 필요한 특정 특성을 설정하기 위한 주요 단계를 식별합니다. 자세한 정보는 342 페이지의 [『IBM MQ classes for Java 애플리케이션 작성』](#)의 내용을 참조하십시오. 샘플은 IBM MQ Java의 향후 릴리스에서 변경될 수 있습니다.

339 페이지의 표 55에서는 각 플랫폼에서 IBM MQ classes for Java 샘플 애플리케이션이 설치된 위치를 표시합니다.

표 55. IBM MQ classes for Java 샘플 애플리케이션에 대한 설치 디렉토리	
플랫폼	디렉토리
 AIX  Linux	MQ_INSTALLATION_PATH/samp/wmqjava/samples

표 55. IBM MQ classes for Java 샘플 애플리케이션에 대한 설치 디렉토리 (계속)	
플랫폼	디렉토리
 Windows	<code>MQ_INSTALLATION_PATH\tools\wmqjava\samples</code>
 IBM i	<code>/qibm/proddata/mqm/java/samples/wmqjava/samples</code>
 z/OS	<code>MQ_INSTALLATION_PATH/java/samples/wmqjava</code>

340 페이지의 표 56에서는 IBM MQ classes for Java에서 제공하는 샘플 애플리케이션 세트를 표시합니다.

표 56. IBM MQ classes for Java 샘플 애플리케이션	
샘플 이름	설명
<code>IMSBridgeSample.java</code>	IMS Bridge를 IBM MQ classes for Java와 함께 사용하는 단순 프로그램입니다.
<code>MQIVP.java</code>	IBM MQ Java 설치 확인 프로그램
<code>MQMessagePropertiesSample.java</code>	메시지 속성 API의 사용법을 보여줍니다.
<code>MQPubSubApiSample.java</code>	게시/구독 API 사용법을 보여줍니다.
<code>MQSample.java</code>	큐에서 메시지를 넣고 가져오는 방법을 설명하는 간단한 프로그램입니다.
<code>MQSampleMessageManager.java</code>	IBM MQ 기본 Java 샘플에서 메시지 처리를 위한 유틸리티 클래스입니다.
<code>mqjcivp.properties</code>	이 자원 번들에는 IBM MQ classes for Java 설치 검증 프로그램(<code>MQIVP.java</code>)에 사용되는 메시지가 포함되어 있습니다.

IBM MQ classes for Java은(는) 샘플 애플리케이션을 실행하는 데 사용할 수 있는 `runjms` 스크립트를 제공합니다. 이 스크립트는 IBM MQ 환경을 설정하여 IBM MQ classes for Java 샘플 애플리케이션을 실행할 수 있습니다.

340 페이지의 표 57에서는 각 플랫폼에서 스크립트 위치를 표시합니다.

표 57. <code>runjms</code> 스크립트의 위치	
플랫폼	디렉토리
 AIX  Linux	<code>MQ_INSTALLATION_PATH/java/bin/runjms</code>
 Windows	<code>MQ_INSTALLATION_PATH\java\bin\runjms.bat</code>
 IBM i	<code>/qibm/proddata/mqm/java/bin/runjms</code> 또는 <code>/qibm/proddata/mqm/java/bin/runjms64</code>
 z/OS	<code>MQ_INSTALLATION_PATHjava/bin/runjms</code>

`runjms` 스크립트를 사용하여 샘플 애플리케이션을 호출하려면 다음 단계를 완료하십시오.

프로시저

1. 명령 프롬프트를 실행하고 실행하려는 샘플 애플리케이션을 포함하는 디렉토리로 이동하십시오.
2. 다음 명령을 입력하십시오.

```
Path to the runjms script/runjms sample_application_name
```

샘플 애플리케이션은 필요한 매개변수 목록을 표시합니다.

3. 다음 명령을 입력하여 다음 매개변수로 샘플을 실행하십시오.

```
Path to the runjms script/runjms sample_application_name parameters
```

예

Linux 예를 들어, Linux에서 MQIVP를 실행하려면 다음 명령을 입력하십시오.

```
cd /opt/mqm/samp/wmqjava/samples
/opt/mqm/java/bin/runjms MQIVP
```

관련 개념

90 페이지의 『IBM MQ classes for JMS에 설치된 항목』

IBM MQ classes for JMS를 설치할 때 여러 파일과 디렉토리가 작성됩니다. Windows에서 환경 변수를 자동으로 설정하여 설치 중에 일부 구성을 수행합니다. 다른 플랫폼과 특정 Windows 환경에서 환경 변수를 설정해야 IBM MQ classes for JMS 애플리케이션을 실행할 수 있습니다.

IBM MQ classes for Java 문제 해결

처음에 설치 확인 프로그램을 실행하십시오. 추적 기능도 사용해야 합니다.

애플리케이션이 성공적으로 완료되지 않으면 설치 확인 프로그램을 실행하고 진단 메시지에 지정된 조언을 따르십시오. 설치 확인 프로그램은 338 페이지의 『IBM MQ classes for Java 설치 확인』에 설명되어 있습니다.

문제점이 계속되고 IBM 서비스 팀에 문의해야 하는 경우 추적 기능을 켜도록 요청할 수 있습니다. 다음 예에 표시된 대로 수행하십시오.

MQIVP 프로그램을 추적하려면 다음을 수행하십시오.

- com.ibm.mq.commonservices 특성 파일을 작성하십시오([com.ibm.mq.commonservices 사용 참조](#)).
- 다음 명령을 입력하십시오.

```
java -Dcom.ibm.mq.commonservices=commonservices_properties_file java
-Djava.library.path= library_path MQIVP -trace
```

설명:

- commonservices_properties_file은 com.ibm.mq.commonservices 특성 파일의 경로입니다(파일 이름 포함).
- library_path는 IBM MQ classes for Java 라이브러리의 경로입니다(329 페이지의 『IBM MQ classes for Java 라이브러리』 참조).

추적을 사용하는 방법에 대한 자세한 정보는 [IBM MQ classes for Java 애플리케이션 추적](#)을 참조하십시오.

z/OS MQ Adv. VUE Java 클라이언트 연결성: 배치 애플리케이션 실행 시 z/OS

특정 조건 하에서, 의 애플리케이션은 IBM MQ classes for Java 클라이언트 연결을 사용하여 z/OS z/OS 의 큐 관리자에 연결할 수 있습니다. 클라이언트 연결을 사용하면 토폴로지를 IBM MQ 단순화할 수 있습니다.

다음 조건 중 하나에 해당하는 경우 지원됩니다:

- IBM MQ classes for Java 는 위치에 있으며 IBM MQ 9.4.0, 큐 관리자는 어떤 버전에서든, 어떤 PID 아래에서든 실행될 수 있습니다.
- IBM MQ classes for Java APAR PH56722 이 적용된 상태에서 IBM MQ 9.2.0 IBM MQ 9.3.0 , 큐 관리자는 어떤 버전에서든, 어떤 PID 아래에서든 실행될 수 있습니다.

- IBM MQ classes for Java APAR PH56722 이 적용되었거나 IBM MQ 9.2.0 적용되지 않은 IBM MQ 9.3.0 상태에서, 큐 관리자는 모든 버전에서 실행될 수 있으나 반드시 권한 부여 IBM MQ Advanced for z/OS Value Unit Edition 상태로 실행되어야 하며, 따라서 해당 **ADVCAP** 매개변수가 ENABLED로 설정되어 있어야 합니다.

자세한 내용은 을 IBM MQ Advanced for z/OS Value Unit Edition 참조하십시오 [IBM MQ 제품 식별자 및 수출 정보](#).

자세한 내용은 [DISPLAY QMGR](#)을 참조하십시오. **ADVCAP** 또한 [QMGRPRODSTART QMGR](#) 에 대한 자세한 내용은 해당 항목을 참조하십시오.

클라이언트 모드 연결을 사용하여 실행 중인 IBM MQ classes for Java 큐 관리자가 아닌 큐 z/OS 관리자에 연결할 수 없습니다. z/OS

클라이언트 모드로 z/OS 연결을 시도하는 애플리케이션이 IBM MQ classes for Java 허용되지 않을 경우 [MQRC_ENVIRONMENT_ERROR](#)가 반환됩니다.

Advanced Message Security(AMS) 지원

IBM MQ classes for Java 클라이언트 애플리케이션은 이 항목에서 앞서 설명한 조건에 따라 원격 z/OS 큐 관리자에 연결할 때 AMS 사용할 수 있습니다.

이러한 방식으로 AMS 사용하려면 클라이언트 애플리케이션은 에서 키 `jceracfks` 저장소 유형으로 `keystore.conf` 를 사용해야 하며, 여기서:

- 특성 이름 접두부는 `jceracfks`이고 이 이름 접두부에서는 대소문자를 구분하지 않습니다.
- 키 저장소는 RACF 키링입니다.
- 비밀번호는 필요하지 않고 무시됩니다. 이는 RACF 키링에서 비밀번호를 사용하지 않기 때문입니다.
- 제공자를 지정하는 경우 제공자는 `IBMJCE`여야 합니다.

`jceracfks`를 AMS와 함께 사용하는 경우 키 저장소는 `safkeyring://user/keyring` 양식이어야 합니다. 여기서

- `safkeyring`은 리터럴이고 이 이름에서는 대소문자가 구분되지 않습니다.
- `user`는 키링을 소유하는 RACF 사용자 ID입니다.
- `keyring`은 RACF 키링의 이름이고, 키링의 이름에서는 대소문자가 구분됩니다.

다음 예제는 사용자 에 대한 표준 `AMSJOHNDOE` 키링을 사용합니다:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

관련 개념

[123 페이지의 『JMS/Jakarta Messaging 클라이언트 연결성: 실행 중인 배치 애플리케이션에 대한 z/OS』](#) 특정 조건 하에서, /Jakarta Messaging IBM MQ classes for JMS 애플리케이션은 클라이언트 연결을 사용하여 z/OSz/OS 의 큐 관리자에 연결할 수 있습니다. 클라이언트 연결을 사용하면 토폴로지를 IBM MQ 단순화할 수 있습니다.

IBM MQ classes for Java 애플리케이션 작성

이 주제 모음은 시스템과 IBM MQ 상호작용하는 애플리케이션 Java 작성에 도움이 되는 정보를 제공합니다.

큐에 IBM MQ 접근하려면 IBM MQ classes for Java , 큐에 메시지를 넣고 큐에서 IBM MQ 메시지를 가져오는 호출을 포함하는 애플리케이션을 Java 작성합니다. 개별 클래스에 대한 자세한 내용은 [IBM MQ classes for Java](#) 의 내용을 참조하십시오.

참고: 자동 클라이언트 재연결은 IBM MQ classes for Java에서 지원되지 않습니다.

IBM MQ classes for Java 인터페이스

프로시저에 따른 IBM MQ API(Application Programming Interface)에서는 오브젝트에 대해 조치를 수행하는 동사를 사용합니다. Java 프로그래밍 인터페이스에서는 메소드를 호출하여 작동하는 오브젝트를 사용합니다.

프로시저에 따른 IBM MQ API(Application Programming Interface)는 다음과 같은 동사를 중심으로 빌드됩니다.

```
MQBACK, MQBEGIN, MQCLOSE, MQCONN, MQDISC,
MQGET, MQINQ, MQOPEN, MQPUT, MQSET, MQSUB
```

이러한 동사는 모두 작동할 IBM MQ 오브젝트에 대한 핸들을 매개변수로 사용합니다. 사용자의 프로그램은 IBM MQ 오브젝트 세트로 구성되며, 사용자가 이러한 오브젝트의 메소드를 호출하여 작업을 수행합니다.

프로시저 인터페이스를 사용하는 경우 호출 MQDISC(Hconn, CompCode, Reason)를 사용하여 큐 관리자와 연결을 끊습니다. 여기서 Hconn은 큐 관리자에 대한 핸들입니다.

Java 인터페이스에서 큐 관리자는 MQQueueManager 클래스의 오브젝트로 표시됩니다. 해당 클래스에서 disconnect() 메소드를 호출하여 큐 관리자와 연결을 끊습니다.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.disconnect();
```

IBM MQ classes for Java 연결 모드

IBM MQ classes for Java용으로 프로그래밍하는 방식은 사용할 연결 모드에 따라 일부 달라집니다.

클라이언트 연결을 사용하는 경우 IBM MQ MQI client와 차이점이 여러 가지 있지만 개념상으로는 비슷합니다. 바인딩 모드를 사용하는 경우 빠른 경로 바인딩을 사용하고 MQBEGIN 명령을 실행할 수 있습니다. MQEnvironment 클래스의 변수를 설정하여 사용할 모드를 지정합니다.

IBM MQ classes for Java 클라이언트 연결

IBM MQ classes for Java를 클라이언트로 사용하는 경우 IBM MQ MQI client와 비슷하지만 여러 차이점이 있습니다.

클라이언트로 사용하기 위해 IBM MQ classes for Java 용으로 프로그래밍하는 경우 다음 차이점에 유의하십시오.

- TCP/IP만 지원합니다.
- 시동 시 IBM MQ 환경 변수를 읽지 않습니다.
- 채널 정의 및 환경 변수에 저장할 정보는 Environment라는 클래스에 저장할 수 있습니다. 또는 연결할 때 이 정보를 매개변수로 전달할 수 있습니다.
- 오류와 예외 조건은 MQException 클래스에 지정된 로그에 기록됩니다. 기본 오류 목적지는 Java 콘솔입니다.
- IBM MQ 클라이언트 구성 파일의 다음 속성만 IBM MQ classes for Java와 관련됩니다. 다른 속성을 지정해도 적용되지 않습니다.

스탠자	속성
클라이언트 구성 파일의 ClientExitPath 스탠자	ExitsDefaultPath
클라이언트 구성 파일의 ClientExitPath 스탠자	ExitsDefaultPath64
클라이언트 구성 파일의 ClientExitPath 스탠자	JavaExitsClasspath
클라이언트 구성 파일의 MessageBuffer 스탠자	MaximumSize
클라이언트 구성 파일의 MessageBuffer 스탠자	PurgeTime
클라이언트 구성 파일의 MessageBuffer 스탠자	UpdatePercentage
클라이언트 구성 파일의 TCP 스탠자	ClntRcvBuffSize
클라이언트 구성 파일의 TCP 스탠자	ClntSndBuffSize
클라이언트 구성 파일의 TCP 스탠자	Connect_Timeout

스탠자	속성
클라이언트 구성 파일의 TCP 스탠자	KeepAlive

- 문자 데이터를 변환해야 하는 큐 관리자에 연결하는 경우, 큐 관리자에서 변환을 수행할 수 없으면 이제 V7 Java 클라이언트에서 변환을 수행할 수 있습니다. 클라이언트 JVM은 클라이언트의 CCSID와 큐 관리자의 CCSID 사이의 변환을 지원해야 합니다.
- 자동 클라이언트 다시 연결은 IBM MQ classes for Java에서 지원되지 않습니다.

클라이언트 모드에서 사용하는 경우 *IBM MQ classes for Java* 에서 MQBEGIN 호출을 지원하지 않습니다.

IBM MQ classes for Java 바인딩 모드

IBM MQ classes for Java의 바인딩 모드는 세 가지 기본 방식에서 클라이언트 모드와 다릅니다.

바인딩 모드에서 사용할 때 IBM MQ classes for Java는 네트워크를 통해 통신하지 않고 JNI(Java Native Interface)를 사용하여 기존 큐 관리자 API를 직접 호출합니다.

기본적으로 바인딩 모드로 IBM MQ classes for Java를 사용하는 애플리케이션은 *ConnectOption*, *MQCNO_STANDARD_BINDINGS*를 사용하여 큐 관리자에 연결합니다.

IBM MQ classes for Java에서 다음 *ConnectOptions*를 지원합니다.

- *MQCNO_FASTPATH_BINDING*
- *MQCNO_STANDARD_BINDING*
- *MQCNO_SHARED_BINDING*
- *MQCNO_ISOLATED_BINDING*

*ConnectOptions*에 대한 자세한 정보는 759 페이지의 『MQCONN 호출을 사용하여 큐 관리자에 연결』의 내용을 참조하십시오.

바인딩 모드에서는 IBM MQ for IBM i 및 IBM MQ for z/OS 이외의 모든 플랫폼에서 큐 관리자가 통합하는 글로벌 작업 단위를 시작하기 위해 MQBEGIN 호출을 지원합니다.

MQEnvironment 클래스에서 제공한 대부분의 매개변수는 바인딩 모드와 관련되지 않으므로 무시됩니다.

사용할 *IBM MQ classes for Java* 연결 정의

사용할 연결 유형은 MQEnvironment 클래스에서 변수를 설정하여 판별합니다.

다음 두 변수가 사용됩니다.

MQEnvironment.properties

연결 유형은 키 이름 CMQC.TRANSPORT_PROPERTY와 연관된 값을 통해 판별합니다. 가능한 값은 다음과 같습니다.

CMQC.TRANSPORT_MQSERIES_BINDINGS

바인딩 모드로 연결

CMQC.TRANSPORT_MQSERIES_CLIENT

클라이언트 모드로 연결

CMQC.TRANSPORT_MQSERIES

연결 모드는 *hostname* 특성의 값을 통해 판별합니다.

MQEnvironment.hostname

다음과 같이 이 변수의 값을 설정하십시오.

- 클라이언트 연결에서는 이 변수의 값을 사용자가 연결할 IBM MQ 서버의 호스트 이름으로 설정하십시오.
- 바인딩 모드에서는 이 변수를 설정하지 않거나 널로 설정하십시오.

큐 관리자의 조작

이 주제 컬렉션에서는 IBM MQ classes for Java를 사용하여 큐 관리자에 연결하고 연결을 끊는 방법을 설명합니다.

IBM MQ classes for Java 에 대한 IBM MQ 환경 설정

클라이언트 모드로 큐 관리자에 애플리케이션을 연결하려는 경우 애플리케이션은 채널 이름, 호스트 이름 및 포트 번호를 지정해야 합니다.

참고: 이 주제의 정보는 애플리케이션이 클라이언트 모드로 큐 관리자에 연결된 경우에만 연관성이 있습니다. 바인딩 모드에서 연결하는 경우 연관성이 없습니다. 참조: 108 페이지의 『IBM MQ classes for JMS의 연결 모드』

두 가지 방법(MQEnvironment 클래스의 필드 또는 MQQueueManager 오브젝트의 특성) 중 하나로 채널 이름, 호스트 이름 및 포트 번호를 지정할 수 있습니다.

MQEnvironment 클래스에서 필드를 설정하는 경우 특성 해시 테이블에서 대체하는 경우를 제외하고 전체 애플리케이션에 적용합니다. MQEnvironment에서 채널 이름 및 호스트 이름을 지정하려면 다음 코드를 사용하십시오.

```
MQEnvironment.hostname = "host.domain.com";
MQEnvironment.channel = "java.client.channel";
```

이는 **MQSERVER** 환경 변수 설정과 동일합니다.

```
"java.client.channel/TCP/host.domain.com".
```

기본적으로 Java 클라이언트는 1414포트에서 IBM MQ 리스너에 연결하려고 시도합니다. 다른 포트를 지정하려면 다음 코드를 사용하십시오.

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

여기서 nnnn은 필수 포트 번호입니다.

작성 시 큐 관리자 오브젝트로 특성을 전달하는 경우 해당 큐 관리자에만 적용됩니다. 키가 **hostname**, **channel** 및 **port**(선택사항)이고 적절한 값을 보유한 해시 테이블 오브젝트에서 입력 항목을 작성합니다. 기본 포트, 1414를 사용하려는 경우 **port** 입력 항목은 생략할 수 있습니다. 특성 해시 테이블을 승인하는 구성자를 사용하여 MQQueueManager 오브젝트를 작성합니다.

애플리케이션 이름을 설정하여 큐 관리자에 대한 연결 식별

애플리케이션은 큐 관리자에 대한 연결을 식별하는 이름을 설정할 수 있습니다. 이 애플리케이션 이름은 **DISPLAY CONN MQSC/PCF** 명령(여기서 필드는 **APPLTAG**라고 함)을 사용하여 표시하거나 IBM MQ 탐색기 애플리케이션 연결 표시(여기서 필드는 **App name**이라고 함)에 표시됩니다.

애플리케이션 이름은 28자로 제한되므로 더 긴 이름은 잘립니다. 애플리케이션 이름이 지정되지 않은 경우 기본 값이 제공됩니다. 기본 이름은 호출(기본) 클래스에 기반하지만 이 정보를 사용할 수 없으면 IBM MQ Client for Java 텍스트가 사용됩니다.

호출 클래스 이름이 사용되면 필요한 경우 패키지 이름의 선두 문자를 제거하여 길이에 맞게 조정됩니다. 예를 들어 호출 클래스가 com.example.MainApp이면 전체 이름이 사용되지만 호출 클래스가 com.example.dictionaryAndThesaurus.multilingual.mainApp이면 multilingual.mainApp 이름이 사용됩니다. 사용 가능한 길이에 맞도록 조정된 맨 오른쪽 패키지 이름과 클래스 이름의 가장 긴 조합이기 때문입니다.

클래스 이름 자체가 28자를 초과하면 길이에 맞게 잘립니다. 예를 들어 com.example.mainApplicationForSecondTestCase는 mainApplicationForSecondTest가 됩니다.

MQEnvironment 클래스에서 애플리케이션 이름을 설정하려면 다음 코드를 사용하여 키가 **MQConstants.APPNAME_PROPERTY**인 MQEnvironment.properties 해시 테이블에 이름을 추가하십시오.

```
MQEnvironment.properties.put(MQConstants.APPNAME_PROPERTY, "my_application_name");
```

MQQueueManager 구성자에 전달된 특성 해시 테이블에서 애플리케이션 이름을 설정하려면 키가 **MQConstants.APPNAME_PROPERTY**인 특성 해시 테이블에 이름을 추가하십시오.

IBM MQ 클라이언트 구성 파일에 지정된 특성 대체

IBM MQ 클라이언트 구성 파일은 IBM MQ classes for Java를 구성하는 데 사용하는 특성도 지정할 수 있습니다. 그러나 IBM MQ MQI client 구성 파일에 지정된 특성은 애플리케이션이 클라이언트 모드로 큐 관리자에 연결할 때만 적용됩니다.

필요한 경우 다음 방법을 사용하여 IBM MQ 구성 파일의 속성을 대체할 수 있습니다. 옵션은 우선순위 순서로 표시됩니다.

- 구성 특성의 Java 시스템 특성을 설정합니다.
- MQEnvironment.properties 맵의 특성을 설정합니다.
- Java 5 이상 릴리스에서 시스템 환경 변수를 설정합니다.

IBM MQ 클라이언트 구성 파일의 다음 속성만 IBM MQ classes for Java와 관련됩니다. 기타 속성을 지정하거나 대체하는 경우 효과가 없습니다.

스탠자	속성
클라이언트 구성 파일의 ClientExitPath 스탠자	ExitsDefaultPath
클라이언트 구성 파일의 ClientExitPath 스탠자	ExitsDefaultPath64
클라이언트 구성 파일의 ClientExitPath 스탠자	JavaExitsClasspath
클라이언트 구성 파일의 MessageBuffer 스탠자	MaximumSize
클라이언트 구성 파일의 MessageBuffer 스탠자	PurgeTime
클라이언트 구성 파일의 MessageBuffer 스탠자	UpdatePercentage
클라이언트 구성 파일의 TCP 스탠자	ClntRcvBufSize
클라이언트 구성 파일의 TCP 스탠자	ClntSndBufSize
클라이언트 구성 파일의 TCP 스탠자	Connect_Timeout
클라이언트 구성 파일의 TCP 스탠자	KeepAlive

IBM MQ classes for Java의 큐 관리자에 연결

MQQueueManager 클래스의 새 인스턴스를 작성하여 큐 관리자에 연결합니다. disconnect() 메소드를 호출하여 큐 관리자에서 연결을 끊습니다.

이제 MQQueueManager 클래스의 새 인스턴스를 작성하여 큐 관리자에 연결할 준비가 되었습니다.

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

큐 관리자에서 연결을 끊으려면 다음과 같이 큐 관리자에서 disconnect() 메소드를 호출하십시오.

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

disconnect 메소드를 호출하는 경우 큐 관리자를 통해 액세스한 열린 모든 큐와 프로세스가 닫힙니다. 그러나 프로그래밍 시 이러한 자원의 사용을 완료한 후 명시적으로 닫는 것이 좋습니다. 이 경우 관련 오브젝트에서 close() 메소드를 사용하십시오.

큐 관리자의 commit() 및 backout() 메소드는 프로시저에 따른 인터페이스와 사용하는 MQCMIT 및 MQBACK 호출에 해당합니다.

IBM MQ classes for Java에서 클라이언트 채널 정의 테이블 사용

IBM MQ classes for Java 클라이언트 애플리케이션은 클라이언트 채널 정의 테이블(CCDT)에 저장된 클라이언트 연결 채널 정의를 사용할 수 있습니다.

MQEnvironment 클래스에서 특정 필드와 환경 특성을 설정하거나 특성 해시 테이블의 MQQueueManager에 전달하여 클라이언트 연결 채널 정의를 작성하는 대신 IBM MQ classes for Java 클라이언트 애플리케이션에서 클라이언트 채널 정의 테이블에 저장된 클라이언트 연결 채널 정의를 사용할 수 있습니다. 이러한 정의는

MQSC(IBM MQ Script) 명령 또는 IBM MQ PCF(Programmable Command Format) 명령을 통해 작성하거나 IBM MQ Explorer를 사용하여 작성합니다.

애플리케이션에서 MQQueueManager 오브젝트를 작성하면 IBM MQ classes for Java 클라이언트가 클라이언트 채널 정의 테이블에서 적절한 클라이언트 연결 채널 정의를 검색하고 채널 정의를 사용하여 MQI 채널을 시작합니다. 클라이언트 채널 정의 테이블 및 이를 구성하는 방법에 대한 자세한 정보는 [클라이언트 채널 정의 테이블](#)을 참조하십시오.

클라이언트 채널 정의 테이블을 사용하려면 애플리케이션에서 먼저 URL 오브젝트를 작성해야 합니다. URL 오브젝트는 클라이언트 채널 정의 테이블이 포함된 파일의 이름과 위치를 식별하고 파일에 액세스하는 방법을 지정하는 URL(Uniform Resource Locator)을 캡슐화합니다.

예를 들어, ccdt1.tab 파일에 클라이언트 채널 정의 테이블이 포함되어 있고 애플리케이션이 실행 중인 동일한 시스템에 저장되는 경우, 애플리케이션은 다음과 같은 방법으로 URL 오브젝트를 작성할 수 있습니다.

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
```

다른 예로, ccdt2.tab 파일에 클라이언트 채널 정의 테이블이 포함되어 있고 애플리케이션이 실행 중인 시스템과 다른 시스템에 저장된다고 가정하십시오. FTP 프로토콜을 사용하여 파일에 액세스할 수 있는 경우, 애플리케이션이 다음과 같은 방법으로 URL 오브젝트를 작성할 수 있습니다.

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
```

애플리케이션에서 URL 오브젝트를 작성하고 나면 애플리케이션이 URL 오브젝트를 매개변수로 사용하는 구성자 중 하나를 사용하여 MQQueueManager 오브젝트를 작성할 수 있습니다. 다음은 예제입니다.

```
MQQueueManager mars = new MQQueueManager("MARS", chanTab2);
```

이 명령문을 사용하면 IBM MQ classes for Java 클라이언트가 URL 오브젝트 chanTab2에 식별된 클라이언트 채널 정의 테이블에 액세스하고 테이블에서 적절한 클라이언트 연결 채널 정의를 검색한 다음 채널 정의를 사용하여 MARS라는 큐 관리자에 대한 MQI 채널을 시작합니다.

애플리케이션에서 클라이언트 채널 정의 테이블을 사용하는 경우 적용되는 다음 사항을 참고하십시오.

- 애플리케이션이 URL 오브젝트를 매개변수로 사용하는 구성자를 사용하여 MQQueueManager 오브젝트를 작성하면 MQEnvironment 클래스에 필드 또는 환경 특성으로 채널 이름을 설정하지 않아야 합니다. 채널 이름이 설정되면 IBM MQ classes for Java 클라이언트에서 MQException이 발생합니다. 값이 널, 비어 있는 문자열 또는 전체가 공백 문자인 문자열이 아닌 경우 채널 이름을 지정하는 필드나 환경 특성이 설정된 것으로 간주됩니다.
- MQQueueManager 구성자의 **queueManagerName** 매개변수 값은 다음 중 하나일 수 있습니다.
 - 큐 관리자의 이름
 - 별표(*) 및 큐 관리자 그룹의 이름
 - 별표(*)
 - 널, 비어 있는 문자열 또는 전체가 공백 문자인 문자열

이러한 값은 MQI(Message Queue Interface)를 사용하는 클라이언트 애플리케이션에 의해 발행되는 MQCONN 호출의 **QMgrName** 매개변수에 대해 사용될 수 있는 동일한 값입니다. 이러한 값의 의미에 대한 자세한 정보는 [744 페이지의 『MQI\(Message Queue Interface\) 개요』](#)의 내용을 참조하십시오.

애플리케이션에서 연결 풀을 사용하는 경우 [365 페이지의 『IBM MQ classes for Java에서 기본 연결 풀 제어』](#)의 내용을 참조하십시오.

- IBM MQ classes for Java 클라이언트가 클라이언트 채널 정의 테이블에서 적당한 클라이언트 연결 채널 정의를 찾으면 이 채널 정의에서 추출한 정보만 사용하여 MQI 채널을 시작합니다. 애플리케이션이 MQEnvironment 클래스에 설정한 채널 관련 필드나 환경 특성은 무시됩니다.

특히 TLS(Transport Layer Security)를 사용하는 경우 다음 사항을 참고하십시오.

- MQI 채널은 클라이언트 채널 정의 테이블에서 추출된 채널 정의가 IBM MQ classes for Java 클라이언트에서 지원하는 CipherSpec의 이름을 지정하는 경우에만 TLS를 사용합니다.

- 또한 클라이언트 채널 정의 테이블에는 인증서 폐기 목록(CRL)을 보유하는 LDAP(Lightweight Directory Access Protocol) 서버의 위치에 대한 정보도 포함되어 있습니다. IBM MQ classes for Java 클라이언트는 CRL을 보유하는 LDAP 서버에 액세스하기 위해 이 정보만 사용합니다.
- 클라이언트 채널 정의 테이블에는 OCSP 응답자의 위치도 포함할 수 있습니다. IBM MQ classes for Java는 클라이언트 채널 정의 테이블 파일의 OCSP 정보를 사용할 수 없습니다. 그러나 [온라인 인증서 프로토콜 사용](#) 절에 설명된 대로 OCSP를 구성할 수 있습니다.

클라이언트 채널 정의 테이블과 TLS를 사용하는 데 관한 자세한 정보는 [MQI 채널에서 TLS를 사용하도록 지정](#)을 참조하십시오.

채널 엑시트를 사용하는 경우, 다음 사항도 참고하십시오.

- MQI 채널은 다른 메소드를 사용하여 지정된 채널 엑시트와 데이터보다 우선하여 클라이언트 채널 정의 테이블에서 추출된 채널 정의를 통해 지정된 채널 엑시트와 관련 사용자 데이터를 사용합니다.
- 클라이언트 채널 정의 테이블에서 추출된 채널 정의는 Java, C 또는 C++로 작성된 채널 엑시트를 지정할 수 있습니다. Java에서 채널 엑시트를 작성하는 방법에 대한 자세한 정보는 359 페이지의 『IBM MQ classes for Java에서 채널 액세스 작성』의 내용을 참조하십시오. 다른 언어로 채널 엑시트를 작성하는 방법에 대한 자세한 정보는 362 페이지의 『채널 출구를 사용하지 않고 Java 작성된 IBM MQ classes for Java』의 내용을 참조하십시오.

IBM MQ classes for Java 클라이언트 연결에 사용할 포트 범위 지정

애플리케이션이 다음 두 방법 중 하나로 바인딩할 수 있는 포트 또는 포트 범위를 지정할 수 있습니다.

IBM MQ classes for Java 애플리케이션이 클라이언트 모드로 IBM MQ 큐 관리자에 연결할 때 방화벽으로 인해 지정된 포트 또는 포트 범위에서 시작된 연결만 허용될 수 있습니다. 이 경우 애플리케이션에서 바인딩할 수 있는 포트 또는 포트 범위를 지정할 수 있습니다. 다음 방식으로 포트를 지정할 수 있습니다.

- MQEnvironment 클래스에서 localAddressSetting 필드를 설정할 수 있습니다. 다음은 예제입니다.

```
MQEnvironment.localAddressSetting = "192.0.2.0(2000,3000)";
```

- CMQC.LOCAL_ADDRESS_PROPERTY 환경 특성을 설정할 수 있습니다. 다음은 예제입니다.

```
(MQEnvironment.properties).put(CMQC.LOCAL_ADDRESS_PROPERTY,
                                "192.0.2.0(2000,3000)");
```

- MQQueueManager 오브젝트를 구성할 때 값이 "192.0.2.0(2000,3000)"인 LOCAL_ADDRESS_PROPERTY를 포함하는 특성 해시 테이블을 전달할 수 있습니다.

각 예에서 애플리케이션이 나중에 큐 관리자에 연결할 때 애플리케이션이 192.0.2.0(2000) - 192.0.2.0(3000) 범위의 로컬 IP 주소와 포트 범위에 바인딩합니다.

네트워크 인터페이스가 두 개 이상인 시스템에서 localAddressSetting 필드 또는 환경 특성

CMQC.LOCAL_ADDRESS_PROPERTY도 사용하여 연결에 사용해야 하는 네트워크 인터페이스를 지정할 수 있습니다.

포트의 범위를 제한하는 경우에는 연결 오류가 발생할 수 있습니다. 오류가 발생하면 IBM MQ 이유 코드 MQRC_Q_MGR_NOT_AVAILABLE 및 다음 메시지를 포함하는 MQException이 발생합니다.

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

지정된 범위의 모든 포트가 사용 중이거나 지정된 IP 주소, 호스트 이름 또는 포트 번호가 올바르지 않은 경우(예: 음수 포트 번호)에는 오류가 발생할 수 있습니다.

IBM MQ classes for Java로 큐, 토픽 및 프로세스에 액세스

큐, 토픽 및 프로세스에 액세스하려면 MQQueueManager 클래스의 메소드를 사용하십시오. MQOD(오브젝트 디스크립터 구조)는 이러한 메소드의 매개변수로 구분됩니다.

큐

큐를 열려면 MQQueueManager 클래스의 accessQueue 메소드를 사용할 수 있습니다. 예를 들어 queueManager라는 큐 관리자에서 다음 코드를 사용하십시오.

```
MQQueue queue = queueManager.accessQueue("qName", CMQC.MQOO_OUTPUT);
```

accessQueue 메소드는 MQQueue 클래스의 새 오브젝트를 리턴합니다.

큐 사용이 완료되면 다음 예에서와 같이 close() 메소드를 사용하여 닫으십시오.

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

또한 MQQueue 구성자를 사용하여 큐를 작성할 수 있습니다. 매개변수는 accessQueue 메소드와 똑같으며 큐 관리자 매개변수가 추가됩니다. 예를 들면, 다음과 같습니다.

```
MQQueue queue = new MQQueue(queueManager,
                             "qName",
                             CMQC.MQOO_OUTPUT,
                             "qMgrName",
                             "dynamicQName",
                             "altUserID");
```

큐를 작성할 때 여러 옵션을 지정할 수 있습니다. 이러한 옵션에 대한 자세한 내용은 Class.com.ibm.mq.MQQueue를 참조하십시오. 이 방식으로 큐 오브젝트를 구성하면 MQQueue의 고유 서브클래스를 작성할 수 있습니다.

토픽

마찬가지로 MQQueueManager 클래스의 accessTopic 메소드를 사용하여 토픽을 열 수 있습니다. 예를 들어 queueManager라는 큐 관리자에서 다음 코드를 사용하여 구독자와 발행자를 작성하십시오.

```
MQTopic subscriber =
    queueManager.accessTopic("TOPICSTRING", "TOPICNAME",
                             CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

```
MQTopic publisher =
    queueManager.accessTopic("TOPICSTRING", "TOPICNAME",
                             CMQC.MQTOPIC_OPEN_AS_PUBLICATION, CMQC.MQOO_OUTPUT);
```

토픽 사용이 완료되면 close() 메소드를 사용하여 닫으십시오.

또한 MQTopic 구성자를 사용하여 토픽을 작성할 수 있습니다. 매개변수는 accessTopic 메소드와 똑같으며 큐 관리자 매개변수가 추가됩니다. 예를 들면, 다음과 같습니다.

```
MQTopic subscriber = new
    MQTopic(queueManager, "TOPICSTRING", "TOPICNAME",
            CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

토픽을 작성할 때 여러 옵션을 지정할 수 있습니다. 이러한 옵션에 대한 자세한 내용은 Class.com.ibm.mq.MQTopic을 참조하십시오. 이 방식으로 토픽 오브젝트를 구성하면 MQTopic의 고유 서브클래스를 작성할 수 있습니다.

토픽은 발행 또는 구독용으로 열어야 합니다. MQQueueManager 클래스에는 8개의 accessTopic 메소드가 있으며 Topic 클래스에는 8개의 구성자가 있습니다. 어느 경우든, 4개에는 **destination** 매개변수가 있고 4개에는 **subscriptionName** 매개변수가 있습니다(두 매개변수를 모두 포함하는 2개 포함). 이러한 사항은 구독용으로 토픽을 여는 데만 사용할 수 있습니다. 나머지 두 메소드에는 **openAs** 매개변수가 있으며 토픽은 **openAs** 매개변수의 값에 따라 발행 또는 구독용으로만 열 수 있습니다.

토픽을 지속 가능한 구독자로 작성하려면 MQQueueManager 클래스의 accessTopic 메소드 또는 구독 이름을 승인하는 MQTopic 구성자를 사용하십시오. 두 경우 모두 CMQC.MQSO_DURABLE 옵션을 설정하십시오.

프로세스

프로세스에 액세스하려면 MQQueueManager의 accessProcess 메소드를 사용하십시오. 예를 들어 queueManager라는 큐 관리자에서 다음 코드를 사용하여 MQProcess 오브젝트를 작성하십시오.

```
MQProcess process =
    queueManager.accessProcess("PROCESSNAME",
        CMQC.MQ00_FAIL_IF QUIESCING);
```

프로세스에 액세스하려면 MQQueueManager의 accessProcess 메소드를 사용하십시오.

accessProcess 메소드에서 MQProcess 클래스의 새 오브젝트를 리턴합니다.

프로세스 오브젝트 사용이 완료되면 다음 예에서와 같이 close() 메소드를 사용하여 닫으십시오.

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

또한 MQProcess 구성자를 사용하여 프로세스를 작성할 수 있습니다. 매개변수는 accessProcess 메소드와 똑같으며 큐 관리자 매개변수가 추가됩니다. 예를 들면, 다음과 같습니다.

```
MQProcess process =
    new MQProcess(queueManager, "PROCESSNAME",
        CMQC.MQ00_FAIL_IF QUIESCING);
```

이 방식으로 프로세스 오브젝트를 구성하면 MQProcess의 고유 서브클래스를 작성할 수 있습니다.

IBM MQ classes for Java의 메시지 핸들링

메시지는 MQMessage 클래스로 표시합니다. MQQueue 및 MQTopic 서브클래스가 있는 MQDestination 클래스의 메소드를 사용하여 메시지를 넣고 가져옵니다.

MQDestination 클래스의 put() 메소드를 사용하여 큐나 토픽에 메시지를 넣습니다. MQDestination 클래스의 get() 메소드를 사용하여 큐나 토픽에서 메시지를 가져옵니다. MQPUT과 MQGET에서 바이트 배열을 넣고 가져오는 프로시저 인터페이스와 달리 Java 프로그래밍 언어는 MQMessage 클래스의 인스턴스를 넣고 가져옵니다. MQMessage 클래스는 실제 메시지 데이터를 포함하는 데이터 버퍼와 해당 메시지를 설명하는 모든 MQMD(메시지 디스크립터) 매개변수와 메시지 특성을 함께 캡슐화합니다.

새 메시지를 빌드하려면 MQMessage 클래스의 새 인스턴스를 작성하고 writeXXX 메소드를 사용하여 메시지 버퍼에 데이터를 넣으십시오.

새 메시지 인스턴스를 작성하면 [MQMD의 초기값 및 언어 선언](#)에 설명된 대로 모든 MQMD 매개변수가 자동으로 기본값으로 설정됩니다. MQDestination의 put() 메소드는 MQPutMessageOptions 클래스의 인스턴스도 매개변수로 받습니다. 이 클래스는 MQPMO 구조를 나타냅니다. 다음 예는 메시지를 작성하여 큐에 넣습니다.

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

String name = "Charlie Jordan";
myMessage.writeInt(name.length());
myMessage.writeBytes(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message
!queue.put(myMessage, pmo);
```

MQDestination의 get() 메소드는 큐에서 방금 받은 메시지를 나타내는 MQMessage의 새 인스턴스를 리턴합니다. 또한 MQGetMessageOptions 클래스의 인스턴스도 매개변수로 받습니다. 이 클래스는 MQGMO 구조를 나타냅니다.

수신 메시지에 맞게 get() 메소드가 내부 버퍼의 크기를 자동으로 조정하므로 최대 메시지 크기를 지정하지 않아도 됩니다. MQMessage 클래스의 readXXX 메소드를 사용하여 리턴된 메시지의 데이터에 액세스하십시오.

다음 예는 큐에서 메시지를 가져오는 방법을 보여줍니다.

```
// Get a message from the queue
MQMessage theMessage = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage, gmo); // has default values

// Extract the message data
int age = theMessage.readInt();
int strlen = theMessage.readInt();
```

```
byte[] strData = new byte[strLen];
theMessage.readFully(strData,0,strLen);
String name = new String(strData,0);
```

encoding 멤버 변수를 설정하여 읽기 및 쓰기 메소드에서 사용하는 번호 형식을 대체할 수 있습니다.

characterSet 멤버 변수를 설정하여 문자열을 읽고 쓰는 데 사용하는 문자 세트를 대체할 수 있습니다.

자세한 정보는 [MQMessage](#) 클래스를 참조하십시오.

참고: *MQMessage*의 *writeUTF()* 메소드는 포함된 유니코드 바이트 외에도 문자열 길이를 자동으로 인코딩합니다. 다른 Java 프로그램에서 메시지를 읽을 때(*readUTF()* 사용) 이 방법을 사용하여 가장 간단하게 문자열 정보를 송신할 수 있습니다.

*IBM MQ classes for Java*에서 비지속 메시지의 성능 향상

클라이언트 애플리케이션에서 메시지를 찾아보거나 비지속 메시지를 이용할 때 성능을 향상시키려면 미리 읽기를 사용할 수 있습니다. *MQGET* 또는 비동기 이용을 사용하는 클라이언트 애플리케이션은 메시지 찾아보기 또는 비지속 메시지 이용 시 성능 개선의 이점이 있습니다.

미리 읽기 기능에 대한 일반 정보는 관련 항목을 참조하십시오.

*IBM MQ classes for Java*에서 *MQQueue* 또는 *MQTopic*의 *CMQC.MQSO_READ_AHEAD* 및 *CMQC.MQSO_NO_READ_AHEAD* 특성을 사용하여 메시지 이용자와 큐 브라우저가 해당 오브젝트에서 미리 읽기를 사용할 수 있는지 판별하십시오.

*IBM MQ classes for Java*를 사용하여 비동기식으로 메시지 넣기

메시지를 비동기식으로 넣으려면 *MQPMO_ASYNC_RESPONSE*를 설정하십시오.

MQDestination 클래스의 *put()* 메소드를 사용하여 큐나 토픽에 메시지를 넣습니다. 메시지를 비동기식으로 넣으려면, 즉 큐 관리자에서 응답을 기다리지 않고 조작이 완료될 수 있게 하려면 *MQPutMessageOptions* 옵션 필드에 *MQPMO_ASYNC_RESPONSE*를 설정할 수 있습니다. 비동기 넣기의 성공 또는 실패를 판별하려면 *MQQueueManager.getAsyncStatus* 호출을 사용하십시오.

***IBM MQ classes for Java*의 발행/구독**

*IBM MQ classes for Java*에서 토픽은 *MQTopic* 클래스로 표시되며 *MQTopic.put()* 메소드를 사용하여 발행합니다.

IBM MQ 발행/구독에 대한 일반 정보는 [발행/구독 메시징](#)을 참조하십시오.

***IBM MQ classes for Java* 를 사용하여 *IBM MQ* 메시지 헤더 처리**

여러 다른 유형의 메시지 헤더를 표시하는 Java 클래스가 제공됩니다. 두 개의 헬퍼 클래스도 제공됩니다.

MQHeader 인터페이스

헤더 오브젝트는 헤더 필드에 액세스하며 메시지 콘텐츠를 읽고 쓰는 범용 메소드를 제공하는 *MQHeader* 인터페이스를 통해 설명합니다. 각 헤더 유형에는 *MQHeader* 인터페이스를 구현하고 개별 필드에 *getter* 및 *setter* 메소드를 추가하는 고유한 클래스가 있습니다. 예를 들어 *MQRFH2* 헤더 유형은 *MQRFH2* 클래스로 표시되며, *MQDLH* 헤더 유형은 *MQDLH* 클래스로 표시되는 식입니다. 헤더 클래스는 필요한 데이터 변환을 자동으로 수행하고 지정된 숫자 인코딩 또는 문자 세트(CCSID)로 된 데이터를 읽거나 쓸 수 있습니다.

중요사항: *MQRFH2* 헤더 클래스는 메시지를 임의 액세스 파일로 처리합니다. 즉, 메시지 시작 부분에 커서가 와야 합니다. *MQRFH*, *MQRFH2*, *MQCIH*, *MQDEAD*, *MQIIH* 또는 *MQXMIT*와 같은 내부 메시지 헤더 클래스를 사용하기 전에 메시지를 클래스에 전달하기 전에 메시지의 커서 위치를 올바른 위치로 업데이트해야 합니다.

헬퍼 클래스

두 개의 헬퍼 클래스인 *MQHeaderIterator* 및 *MQHeaderList*는 메시지에서 헤더 콘텐츠의 읽기 및 디코딩(구문 분석)을 도와 줍니다.

- *MQHeaderIterator* 클래스는 *java.util.Iterator*와 같이 동작합니다. 메시지에 더 많은 헤더가 있으면 *next()* 메소드가 *true*를 리턴하고 *nextHeader()* 또는 *next()* 메소드가 다음 헤더 오브젝트를 리턴합니다.

- MQHeaderList는 java.util.List와 같이 작동합니다. MQHeaderIterator와 마찬가지로 헤더 콘텐츠를 구문 분석할 뿐 아니라 특정 헤더를 검색할 수 있고, 새 헤더를 추가하며, 기존 헤더를 제거하고, 헤더 필드를 업데이트한 다음 헤더 콘텐츠를 다시 메시지에 쓸 수 있습니다. 또는 빈 MQHeaderList를 작성한 다음 이를 헤더 인스턴스로 채우고 메시지에 한 번 또는 반복적으로 쓸 수도 있습니다.

MQHeaderIterator와 MQHeaderList 클래스는 MQHeaderRegistry의 정보를 사용하여 특정 메시지 유형 및 형식과 연관된 IBM MQ 헤더 클래스를 확인합니다. MQHeaderRegistry는 현재 모든 IBM MQ 형식 및 헤더 유형과 구현 클래스에 대한 정보로 구성되며, 사용자가 고유 헤더 유형도 등록할 수 있습니다.

일반적으로 사용되는 다음과 같은 IBM MQ 헤더에 대한 지원이 제공됩니다.

- MQRFH - 규칙 및 형식화 헤더
- MQRFH2 - MQRFH와 마찬가지로 IBM Integration Bus에 속한 메시지 브로커에/에서 메시지를 전달하는 데 사용됩니다. 메시지 특성을 포함하는 데도 사용됩니다.
- MQCIH - CICS 브릿지
- MQDLH - 데드 레터 헤더
- MQIIH - IMS 정보 헤더
- MQRMH - 참조 메시지 헤더
- MQSAPH - SAP 헤더
- MQWIH - 작업 정보 헤더
- MQXQH - 전송 큐 헤더
- MQDH - 분배 헤더
- MQEPH - 캡슐화된 PCF 헤더

사용자 고유의 헤더를 나타내는 클래스를 정의할 수도 있습니다.

MQHeaderIterator를 사용하여 RFH2 헤더를 가져오려면 GetMessageOptions에 MQGMO_PROPERTIES_FORCE_MQRFH2를 설정하거나 큐 특성 PROPCTL을 FORCE로 설정하십시오.

*IBM MQ classes for Java*를 사용하여 메시지의 모든 헤더 인쇄

이 예에서 MQHeaderIterator의 인스턴스는 큐에서 수신한 MQMessage의 헤더를 구문 분석합니다.

nextHeader() 메소드에서 리턴된 MQHeader 오브젝트는 toString 메소드를 호출할 때 해당 구조와 콘텐츠를 표시합니다.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeader;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

while (it.hasNext ())
{
    MQHeader header = it.nextHeader ();

    System.out.println ("Header type " + header.type () + ": " + header);
}
```

*IBM MQ classes for Java*를 사용하여 메시지의 헤더 건너뛰기

이 예에서는 MQHeaderIterator의 skipHeaders() 메소드가 마지막 헤더 바로 다음에 메시지 읽기 커서를 둡니다.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

it.skipHeaders ();
```


*IBM MQ classes for Java*를 사용하여 데드-레터 메시지에서 이유 코드 찾기

이 예에서는 read 메소드가 메시지에서 읽어 MQDLH 오브젝트를 채웁니다. 읽기 조작 후에 메시지 읽기 커서가 MQDLH 헤더 콘텐츠 바로 다음에 옵니다.

큐 관리자의 데드-레터 큐에 있는 메시지는 데드-레터 헤더(MQDLH)로 접두부가 지정됩니다. 이러한 메시지를 핸들링하는 방법을 판별하려면(예: 재시도할지 아니면 제거할지 판별) 데드 레터 핸들링 애플리케이션에서 MQDLH에 포함된 이유 코드를 검색해야 합니다.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;

...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH ();

dlh.read (message);

System.out.println ("Reason: " + dlh.getReason ());
```

모든 헤더 클래스에서는 단일 단계로 메시지에서 해당 클래스를 직접 초기화하는 편리한 구성자도 제공합니다. 따라서 이 예의 코드는 다음과 같이 간단할 수 있습니다.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;

...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH (message);

System.out.println ("Reason: " + dlh.getReason ());
```

*IBM MQ classes for Java*를 사용하여 데드 레터 메시지에서 헤더 읽기 및 제거

이 예에서는 MQDLH를 사용하여 데드-레터 메시지에서 헤더를 제거합니다.

일반적으로 데드-레터 핸들링 애플리케이션은 이유 코드가 임시 오류를 표시하는 경우 거부된 메시지를 다시 제출합니다. 메시지를 다시 제출하기 전에 MQDLH 헤더를 제거해야 합니다.

이 예는 다음 단계를 수행합니다(예제 코드의 주석 참조).

1. MQHeaderList에서 전체 메시지를 읽고, 메시지에 있는 각 헤더가 목록의 항목이 됩니다.
2. 데드-레터 메시지에는 첫 번째 헤더로 MQDLH가 포함되므로 헤더 목록의 첫 번째 항목에서 찾을 수 있습니다. MQHeaderList를 빌드할 때 MQDLH가 메시지에서 이미 입력되었으므로 읽기 메소드를 호출하지 않아도 됩니다.
3. 이유 코드는 MQDLH 클래스에서 제공하는 getReason() 메소드를 사용하여 추출합니다.
4. 이유 코드가 검사되었으며 메시지를 다시 제출하는 것이 적절함을 표시합니다. MQDLH는 MQHeaderList remove() 메소드를 사용하여 제거합니다.
5. MQHeaderList를 통해 나머지 콘텐츠를 새 메시지 오브젝트에 씁니다. 새 메시지에는 이제 원래 메시지에서 MQDLH를 제외한 모든 사항이 포함되며 큐에 쓸 수 있습니다. 구성자와 write 메소드의 **true** 인수는 메시지 본문이 MQHeaderList에 보유되며 다시 기록됨을 표시합니다.
6. 새 메시지의 메시지 디스크립터에 있는 형식 필드에는 이제 MQDLH 형식 필드에 이전에 있던 값이 포함되어 있습니다. 메시지 데이터가 메시지 디스크립터의 숫자 인코딩 및 CCSID 세트와 일치합니다.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
import com.ibm.mq.headers.MQHeaderList;

...
MQMessage message = ... // Message received from the dead-letter queue.
MQHeaderList list = new MQHeaderList (message, true); // Step 1.
MQDLH dlh = (MQDLH) list.get (0); // Step 2.
int reason = dlh.getReason (); // Step 3.
...
list.remove (dlh); // Step 4.

MQMessage newMessage = new MQMessage ();

list.write (newMessage, true); // Step 5.
newMessage.format = list.getFormat (); // Step 6.
```

*IBM MQ classes for Java*를 사용하여 메시지의 콘텐츠 인쇄

이 예에서는 MQHeaderList를 사용하여 헤더를 비롯한 메시지의 콘텐츠를 출력합니다.

출력에는 메시지 본문 외에도 모든 헤더 콘텐츠 보기가 포함되어 있습니다. MQHeaderList 클래스는 한 번에 모든 헤더를 디코딩하는 반면 MQHeaderIterator는 애플리케이션이 제어하는 상태에서 한 번에 하나씩 스텝 스루합니다. WebSphere MQ 애플리케이션을 작성할 때 이 기술을 사용하여 단순 디버깅 도구를 제공할 수 있습니다.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;

...
MQMessage message = ... // Message received from a queue.

System.out.println (new MQHeaderList (message, true));
```

이 예는 MQMD 클래스를 사용하여 메시지 디스크립터 필드도 인쇄합니다. com.ibm.mq.headers.MQMD 클래스의 copyFrom() 메소드는 메시지 본문을 읽지 않고 MQMessage의 메시지 디스크립터 필드에서 헤더 오브젝트를 채웁니다.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQMD;
import com.ibm.mq.headers.MQHeaderList;

...
MQMessage message = ...
MQMD md = new MQMD ();
...
md.copyFrom (message);
System.out.println (md + "\n" + new MQHeaderList (message, true));
```

*IBM MQ classes for Java*를 사용하여 메시지에서 특정 유형의 헤더 찾기

이 예에서는 MQHeaderList의 indexOf(String) 메소드를 사용하여 메시지에서 MQRFH2 헤더(있는 경우)를 찾습니다.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
import com.ibm.mq.headers.MQRFH2;

...
MQMessage message = ...
MQHeaderList list = new MQHeaderList (message);
int index = list.indexOf ("MQRFH2");

if (index >= 0)
{
    MQRFH2 rfh = (MQRFH2) list.get (index);
    ...
}
```

*IBM MQ classes for Java*를 사용하여 MQRFH2 헤더 분석

이 예에서는 MQRFH2 클래스를 사용하여 이름 지정된 폴더의 알려진 필드 값에 액세스하는 방법을 보여줍니다.

MQRFH2 클래스는 구조의 고정 부분에 있는 필드뿐 아니라 NameValueData 필드에서 이동되는 XML 인코딩된 폴더 콘텐츠에 액세스하는 다양한 방법을 제공합니다. 이 예에서는 이름 지정된 폴더의 알려진 필드 값(이 경우 MQ JMS 메시지의 응답 큐 이름을 표시하는 jms 폴더의 Rto 필드)에 액세스하는 방법을 보여줍니다.

```
MQRFH2 rfh = ...

String value = rfh.getStringFieldValue ("jms", "Rto");
```

MQRFH2의 콘텐츠를 검색하려면(특정 필드를 직접 요청하지 않음) MQRFH2.Element 목록을 리턴하는 getFolder 메소드를 사용할 수 있습니다. 이 목록은 필드와 다른 폴더를 포함할 수 있는 폴더의 구조를 나타냅니다. 필드 또는 폴더를 널로 설정하면 MQRFH2에서 제거됩니다. 이 방식으로 NameValueData 폴더 콘텐츠를 조작할 때 StrucLength 필드가 자동으로 적절하게 업데이트됩니다.

*IBM MQ classes for Java*를 사용하여 MQMessage 오브젝트 이외의 바이트 스트림 읽기 및 쓰기

이러한 예에서는 데이터 소스가 MQMessage 오브젝트가 아닐 때 헤더 클래스를 사용하여 IBM MQ 헤더 콘텐츠를 구문 분석하고 조작합니다.

데이터 소스가 MQMessage 오브젝트가 아닌 경우에도 헤더 클래스를 사용하여 IBM MQ 헤더 콘텐츠를 구문 분석하고 조작할 수 있습니다. 모든 헤더 클래스에서 구현한 MQHeader 인터페이스는 int read

(java.io.DataInput message, int encoding, int characterSet) 및 int write (java.io.DataOutput message, int encoding, int characterSet) 메소드를 제공합니다. com.ibm.mq.MQMessage 클래스가 java.io.DataInput and java.io.DataOutput 인터페이스를 구현합니다. 즉, 두 개의 MQHeader 메소드를 사용하여 MQMessage 콘텐츠를 읽고 쓰며, 메시지 디스크립터에 지정된 인코딩과 CCSID를 대체합니다. 다른 인코딩으로 된 헤더 체인을 포함하는 메시지에 유용합니다.

JMS 메시지를 통해 이동되는 바이트 배열 또는 파일이나 소켓 스트림 같은 다른 데이터 스트림에서 DataInput 및 DataOutput 오브젝트도 확보할 수 있습니다. java.io.DataInputStream 클래스가 DataInput을 구현하고 java.io.DataOutputStream 클래스가 DataOutput을 구현합니다. 이 예에서는 바이트 배열에서 IBM MQ 헤더 콘텐츠를 읽습니다.

```
import java.io.*;
import com.ibm.mq.headers.*;
...
byte [] bytes = ...
DataInput in = new DataInputStream (new ByteArrayInputStream (bytes));
MQHeaderIterator it = new MQHeaderIterator (in, CMQC.MQENC_NATIVE,
    CMQC.MQCCSI_DEFAULT);
```

MQHeaderIterator로 시작하는 행은 다음으로 바꿀 수 있습니다.

```
MQDLH dlh = new MQDLH (in, CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
// or any other header type
```

이 예는 DataOutputStream을 사용하여 바이트 배열에 씁니다.

```
MQHeader header = ... // Could be any header type
ByteArrayOutputStream out = new ByteArrayOutputStream ();

header.write (new DataOutputStream (out), CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
byte [] bytes = out.toByteArray ();
```

이 방식으로 스트림에 대해 작업할 때 encoding 및 characterSet 인수에 올바른 값을 사용하도록 주의하십시오. 헤더를 읽을 때 바이트 콘텐츠를 원래 작성할 때 사용한 인코딩과 CCSID를 지정하십시오. 헤더를 쓸 때는 생성할 인코딩 및 CCSID를 지정하십시오. 데이터 변환은 헤더 클래스에 의해 자동으로 수행됩니다.

*IBM MQ classes for Java*를 사용하여 새 헤더 유형의 클래스 작성

'IBM MQ classes for Java'와 함께 제공되지 않는 헤더 유형에 대해 'Java' 클래스를 만들 수 있습니다.

'IBM MQ classes for Java'와 함께 제공된 헤더 클래스와 동일한 방식으로 사용할 수 있는 새 헤더 유형을 나타내는 'Java' 클래스를 추가하려면 MQHeader 인터페이스를 구현하는 클래스를 생성합니다. 가장 간단한 방법은 com.ibm.mq.headers.impl.Header 클래스를 확장하는 것입니다. 이 예에서는 MQTM 헤더 구조를 표시하는 완전한 기능의 클래스를 생성합니다. 각 필드의 개별 getter 및 setter 메소드를 추가하지 않아도 되며 헤더 클래스 사용자에게 매우 유용합니다. 필드 이름으로 문자열을 사용하는 일반 getValue와 setValue 메소드는 헤더 유형에 정의된 모든 필드에 사용할 수 있습니다. 상속된 read, write 및 size 메소드를 사용하면 새 헤더 유형의 인스턴스를 읽고 쓸 수 있으며 필드 정의를 기반으로 헤더 크기를 정확하게 계산합니다. 유형 정의는 한 번만 작성되지 만 이 헤더 클래스의 인스턴스를 여러 개 작성할 수 있습니다. MQHeaderIterator 또는 MQHeaderList 클래스를 사용하여 새 헤더 정의를 디코딩할 수 있도록 MQHeaderRegistry를 사용하여 등록합니다. 그러나 MQTM 헤더 클래스는 이 패키지에 이미 제공되어 기본 레지스트리에 등록되어 있다는 점을 참고하십시오.

```
import com.ibm.mq.headers.impl.Header;
import com.ibm.mq.headers.impl.HeaderField;
import com.ibm.mq.headers.CMQC;

public class MQTM extends Header {
    final static HeaderType TYPE = new HeaderType ("MQTM");
    final static HeaderField StrucId = TYPE.addMQChar ("StrucId", CMQC.MQTM_STRUC_ID);
    final static HeaderField Version = TYPE.addMQLong ("Version", CMQC.MQTM_VERSION_1);
    final static HeaderField QName = TYPE.addMQChar ("QName", CMQC.MQ_Q_NAME_LENGTH);
    final static HeaderField ProcessName = TYPE.addMQChar ("ProcessName",
        CMQC.MQ_PROCESS_NAME_LENGTH);
    final static HeaderField TriggerData = TYPE.addMQChar ("TriggerData",
        CMQC.MQ_TRIGGER_DATA_LENGTH);
    final static HeaderField ApplType = TYPE.addMQLong ("ApplType");
    final static HeaderField ApplId = TYPE.addMQChar ("ApplId", 256);
    final static HeaderField EnvData = TYPE.addMQChar ("EnvData", 128);
    final static HeaderField UserData = TYPE.addMQChar ("UserData", 128);
```

```

protected MQTM (HeaderType type){
    super (type);
}
public String getStrucId () {
    return getStringValue (StrucId);
}
public int getVersion () {
    return getIntValue (Version);
}
public String getQName () {
    return getStringValue (QName);
}
public void setQName (String value) {
    setStringValue (QName, value);
}
// ...Add convenience getters and setters for remaining fields in the same way.
}

```

IBM MQ classes for Java로 PCF 메시지 핸들링

Java 클래스는 PCF 구조화된 메시지를 작성하고 구문 분석하며 쉽게 PCF 요청을 송신하고 PCF 응답을 수집하기 위해 제공됩니다.

PCFMessage 및 MQCFGR 클래스는 PCF 매개변수 구조의 배열을 나타냅니다. PCF 매개변수를 추가하고 검색하기 위한 편리한 메소드를 제공합니다.

PCF 매개변수 구조는 MQCFH, MQCFIN, MQCFIN64, MQCFST, MQCFBS, MQCFIL, MQCFIL64 MQCFSL 및 MQCFGR 클래스로 표시됩니다. 이러한 클래스는 다음과 같은 기본 작동 인터페이스를 공유합니다.

- 메시지 콘텐츠를 읽고 쓸 메소드: read (), write () 및 size ()
- 매개변수를 조정할 메소드: getValue (), setValue (), getParameter () 등
- MQMessage의 PCF 콘텐츠를 구문 분석하는 열거자 메소드 .nextParameter ()

PCF 필터 매개변수는 필터 함수를 제공하기 위해 inquire 명령에서 사용됩니다. 다음 클래스로 캡슐화됩니다.

- MQCFIF - 정수 필터
- MQCFSF - 문자열 필터
- MQCFBF - 바이트 필터

큐 관리자에 대한 연결, 명령 서버 큐 및 관련된 응답 큐를 관리하기 위해 두 개의 에이전트 클래스, PCFAgent 및 PCFMessageAgent가 제공됩니다. PCFMessageAgent는 PCFAgent를 확장하며, 일반적으로 먼저 사용해야 합니다. PCFMessageAgent 클래스는 수신된 MQMessages를 변환하고 PCFMessage 배열로 호출자에 다시 전달합니다. PCFAgent는 사용하기 전에 구문 분석해야 하는 MQMessages 배열을 리턴합니다.

IBM MQ classes for Java에서 메시지 특성 핸들링

IBM MQ classes for Java에는 메시지 핸들을 처리하는 함수 호출이 없습니다. 메시지 핸들 특성을 설정하거나 리턴하거나 삭제하려면 MQMessage 클래스의 메소드를 사용하십시오.

메시지 특성에 대한 일반 정보는 33 페이지의 [『특성 이름』](#)의 내용을 참조하십시오.

IBM MQ classes for Java에서는 MQMessage 클래스를 통해 메시지에 액세스합니다. 따라서 Java 환경에서는 메시지 핸들이 제공되지 않지만 IBM MQ 함수 호출 MQCRTMH, MQDLTMH, MQMHBUF 및 MQBUFMH에 대응하는 호출이 있습니다.

프로시저에 따른 인터페이스에 메시지 핸들 특성을 설정하려면 MQSETMP 호출을 사용할 수 있습니다. IBM MQ classes for Java에서 MQMessage 클래스의 적절한 메소드를 사용하십시오.

- setBooleanProperty
- setByteProperty
- setBytesProperty
- setShortProperty
- setIntProperty
- setInt2Property
- setInt4Property

- setInt8Property
- setLongProperty
- setFloatProperty
- setDoubleProperty
- setStringProperty
- setObjectProperty

이러한 메소드는 집합적으로 *set*property* 메소드라고도 합니다.

프로시저에 따른 인터페이스로 메시지 핸들 특성 값을 리턴하려면 MQINQMP 호출을 사용합니다. IBM MQ classes for Java에서 MQMessage 클래스의 적절한 메소드를 사용하십시오.

- getBooleanProperty
- getByteProperty
- getBytesProperty
- getShortProperty
- getIntProperty
- getInt2Property
- getInt4Property
- getInt8Property
- getLongProperty
- getFloatProperty
- getDoubleProperty
- getStringProperty
- getObjectProperty

이러한 메소드는 집합적으로 *get*property* 메소드라고도 합니다.

프로시저에 따른 인터페이스로 메시지 핸들 특성 값을 삭제하려면 MQDLTMP 호출을 사용합니다. IBM MQ classes for Java에서 MQMessage 클래스의 deleteProperty 메소드를 사용하십시오.

IBM MQ classes for Java의 오류 핸들링

Java try 및 catch 블록을 사용하여 IBM MQ classes for Java 에서 발생하는 오류를 처리합니다.

Java 인터페이스의 메소드는 완료 코드와 이유 코드를 리턴하지 않습니다. 대신 IBM MQ 호출을 통해 생성된 완료 코드와 이유 코드가 둘 다 0이 아닌 경우 예외 처리를 합니다. 그러면 IBM MQ를 각각 호출한 후에 리턴 코드를 확인할 필요가 없도록 프로그램 로직이 간소화됩니다. 프로그램에서 실패의 가능성을 처리할 지점을 결정할 수 있습니다. 다음 예와 같이 이러한 지점의 코드를 try와 catch 블록으로 묶습니다.

```
try {
    myQueue.put(messageA,putMessageOptionsA);
    myQueue.put(messageB,putMessageOptionsB);
}
catch (MQException ex) {
    // This block of code is only executed if one of
    // the two put methods gave rise to a non-zero
    // completion code or reason code.
    System.out.println("An error occurred during the put operation:" +
        "CC = " + ex.completionCode +
        "RC = " + ex.reasonCode);
    System.out.println("Cause exception:" + ex.getCause() );
}
```

z/OS 에 대한 Java 예외에서 다시 보고되는 IBM MQ 호출 이유 코드는 [API 완료 및 이유 코드](#)에 문서화되어 있습니다.

IBM MQ classes for Java 애플리케이션을 실행하는 동안 발생한 예외도 로그에 기록됩니다. 그러나 특정 이유 코드와 연관된 예외가 로깅되지 않도록 애플리케이션에서 MQException.logExclude() 메소드를 호출할 수 있습니다. 특정 이유 코드와 연관된 많은 예외가 발생할 것으로 예상하지만 로그가 이러한 예외로 채워지지 않게 하려

는 경우 이 작업을 수행할 수 있습니다. 예를 들어, 애플리케이션이 루프를 반복할 때 큐에서 메시지를 가져오려고 시도하지만 대부분의 시도에서 큐에 적당한 메시지가 없을 것으로 예상하는 경우 이유 코드 MQRC_NO_MSG_AVAILABLE와 연관된 예외를 방지할 수 있습니다. 애플리케이션이 특정 이유 코드와 연관된 예외가 로깅되지 않도록 이미 방지한 경우 MQException.logInclude() 메소드를 호출하여 이러한 예외를 로깅할 수 있습니다.

이유 코드가 오류와 연관된 세부 정보를 모두 전달하지 못하는 경우도 있습니다. 애플리케이션이 발생한 각 예외에 링크된 예외가 있는지 확인해야 합니다. 링크된 예외 자체에 또 다른 링크된 예외가 있을 수 있으므로, 링크된 예외는 원래의 기본 문제점으로 다시 되돌리는 체인을 작성합니다. 링크된 예외는 java.lang.Throwable 클래스의 체인 예외 메커니즘을 사용하여 구현되며, 애플리케이션은 Throwable.getCause() 메소드를 호출하여 링크된 예외를 확보합니다. MQException 인스턴스인 예외에서 MQException.getCause()가 com.ibm.mq.jmqi.JmqiException의 기본 인스턴스를 검색하고 이 예외의 getCause가 오류를 초래한 기본 java.lang.Exception을 검색합니다.

IBM MQ classes for Java의 속성 값 가져오기 및 설정

여러 공통 속성용으로 getXXX() 및 setXXX() 메소드가 제공됩니다. 다른 속성은 일반 inquire() 및 set() 메소드를 사용하여 액세스할 수 있습니다.

여러 일반적인 속성용으로 MQManagedObject, MQDestination, MQQueue, MQTopic, MQProcess 및 MQQueueManager 클래스에는 getXXX() 및 setXXX() 메소드가 포함되어 있습니다. 이러한 메소드를 사용하면 속성 값을 가져오고 설정할 수 있습니다. MQDestination, MQQueue 및 MQTopic의 경우 오브젝트를 열 때 적절한 inquire 및 set 플래그를 지정하는 경우에만 메소드가 작동합니다.

덜 일반적인 속성용으로는 MQQueueManager, MQDestination, MQQueue, MQTopic 및 MQProcess 클래스가 MQManagedObject라는 클래스에서 모두 상속합니다. 이 클래스는 inquire() 및 set() 인터페이스를 정의합니다.

new 연산자를 사용하여 새 큐 관리자를 작성하면 조회를 위해 자동으로 열립니다. accessProcess() 메소드를 사용하여 프로세스 오브젝트에 액세스하면 조회를 위해 해당 오브젝트가 자동으로 열립니다. accessQueue() 메소드를 사용하여 큐 오브젝트에 액세스하면 해당 오브젝트가 조회 또는 설정 조작을 위해 자동으로 열리지 않습니다. 이러한 옵션을 자동으로 추가하면 일부 리모트 큐 유형에 문제점이 초래될 수 있기 때문입니다. 큐에서 inquire, set, getXXX 및 setXXX 메소드를 사용하려면 accessQueue() 메소드의 openOptions 매개변수에 적절한 inquire 및 set 플래그를 지정해야 합니다. 목적지와 토픽 오브젝트에도 마찬가지입니다.

inquire 및 set 메소드는 다음 세 개의 매개변수를 사용합니다.

- selectors 배열
- intAttrs 배열
- charAttrs 배열

Java의 배열 길이는 항상 공개되므로 MQINQ에 있는 SelectorCount, IntAttrCount 및 CharAttrLength 매개변수가 필요하지 않습니다. 다음 예는 큐에서 조회하는 방법을 보여줍니다.

```
// inquire on a queue
final static int MQIA_DEF_PRIORITY = 6;
final static int MQCA_Q_DESC = 2013;
final static int MQ_Q_DESC_LENGTH = 64;

int[] selectors = new int[2];
int[] intAttrs = new int[1];
byte[] charAttrs = new byte[MQ_Q_DESC_LENGTH]

selectors[0] = MQIA_DEF_PRIORITY;
selectors[1] = MQCA_Q_DESC;

queue.inquire(selectors,intAttrs,charAttrs);

System.out.println("Default Priority = " + intAttrs[0]);
System.out.println("Description : " + new String(charAttrs,0));
```

Java의 멀티스레드 프로그램

Java 런타임 환경은 본질적으로 멀티스레드입니다. IBM MQ classes for Java를 사용하면 여러 스레드에서 큐 관리자 오브젝트를 공유할 수 있지만 대상 큐 관리자에 대한 모든 액세스를 동기화되는지 확인하십시오.

Java로는 멀티스레드 프로그램을 방지하기가 어렵습니다. 큐 관리자에 연결하고 시동 시 큐를 여는 단순 프로그램을 고려하십시오. 프로그램이 화면에 단일 단추를 표시합니다. 사용자가 해당 단추를 클릭하면 프로그램이 큐에서 메시지를 폐치합니다.

Java 런타임 환경은 본질적으로 멀티스레드입니다. 그러므로 애플리케이션 초기화가 한 스레드에서 수행되고, 단추 누르기에 응답하여 실행되는 코드는 별도 스레드(사용자 인터페이스 스레드)에서 실행됩니다.

C 기반 IBM MQ MQI client에서는 여러 스레드에서 핸들을 공유하는 데 제한사항이 있으므로 문제가 초래됩니다. IBM MQ classes for Java는 이 제한조건을 완화하므로 여러 스레드에서 큐 관리자 오브젝트(및 연관 큐, 토픽 및 프로세스 오브젝트)를 공유할 수 있습니다.

IBM MQ classes for Java를 구현하면 특정 연결(MQQueueManager 오브젝트 인스턴스)에서 대상 IBM MQ 큐 관리자에 대한 모든 액세스가 동기화됩니다. 해당 연결을 위해 진행 중인 다른 모든 호출이 완료될 때까지 큐 관리자에 대한 호출을 실행할 스레드는 차단됩니다. 프로그램의 여러 스레드에서 동일한 큐 관리자에 동시에 액세스해야 하는 경우 동기 액세스해야 하는 각 스레드의 새 MQQueueManager 오브젝트를 작성하십시오(각 스레드에 개별 MQCONN 호출을 실행하는 것과 동등).

참고: com.ibm.mq.MQGetMessageOptions 클래스의 인스턴스는 동시에 메시지를 요청하는 스레드 간에 공유하지 않아야 합니다. 해당 MQGET 요청 중에 이 클래스의 인스턴스가 데이터로 업데이트되므로, 오브젝트의 동일한 인스턴스에서 여러 스레드가 동시에 작동할 때 예상치 않은 결과가 발생할 수 있습니다.

IBM MQ classes for Java의 채널 엑시트 사용

IBM MQ classes for Java를 사용하여 애플리케이션에서 채널 엑시트를 사용하는 방법의 개요입니다.

다음 주제는 Java로 채널 엑시트를 작성하는 방법, 지정하는 방법 및 해당 엑시트에 데이터를 전달하는 방식을 설명합니다. 그런 다음 C로 작성된 채널 엑시트를 사용하는 방법과 일련의 채널 엑시트를 사용하는 방법을 설명합니다.

애플리케이션에 채널 엑시트 클래스를 로드하기 위한 정확한 보안 권한이 있어야 합니다.

*IBM MQ classes for Java*에서 채널 액세스 작성

적절한 인터페이스를 구현하는 Java 클래스를 정의하여 고유 채널 엑시트를 제공할 수 있습니다.

엑시트를 구현하기 위해 적절한 인터페이스를 구현하는 새 Java 클래스를 정의합니다. 세 개의 엑시트 인터페이스는 다음 com.ibm.mq.exits 패키지에 정의되어 있습니다.

- WMQSendExit
- WMQReceiveExit
- WMQSecurityExit

참고: 채널 엑시트는 클라이언트 연결용으로만 지원되며, 바인딩 연결용으로는 지원되지 않습니다. 예를 들어 C로 작성된 클라이언트 애플리케이션을 사용하는 경우 'Java 채널 종료는' IBM MQ classes for Java' 외부에서 사용할 수 없습니다.

연결에 정의된 TLS 암호화는 송신 및 보안 엑시트를 호출한 다음 수행됩니다. 마찬가지로 복호화는 수신 및 보안 엑시트를 호출하기 전에 수행합니다.

다음 샘플은 세 가지 인터페이스를 모두 구현하는 클래스를 정의합니다.

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
    public MyMQExits(){
    }
    // This method comes from the send exit interface
    public ByteBuffer channelSendExit(
MQCXP channelExitParms,
                                MQCD channelDefinition,
                                ByteBuffer agentBuffer)
    {
        // Fill in the body of the send exit here
    }
    // This method comes from the receive exit interface
    public ByteBuffer channelReceiveExit(
MQCXP channelExitParms,
                                MQCD channelDefinition,
                                ByteBuffer agentBuffer)
    {
    }
```

```

    // Fill in the body of the receive exit here
}
// This method comes from the security exit interface
public ByteBuffer channelSecurityExit(
MQCXP channelExitParms,
                                MQCD channelDefinition,
                                ByteBuffer agentBuffer)
{
    // Fill in the body of the security exit here
}
}

```

각 엑시트에 MQCXP 오브젝트와 MQCD 오브젝트가 전달됩니다. 이러한 오브젝트는 절차적 인터페이스에 정의된 MQCXP 및 MQCD 구조를 표시합니다.

작성하는 엑시트 클래스에는 구성자가 있어야 합니다. 이 구성자는 기본 구성자이거나 문자열 인수를 사용하는 구성자일 수 있습니다. 문자열을 사용하는 경우 이 구성자를 작성할 때 사용자 데이터가 엑시트에 전달됩니다. 엑시트 클래스에 기본 구성자와 단일 인수 구성자가 모두 포함된 경우 단일 인수 구성에 우선순위가 있습니다.

송신과 보안 엑시트의 경우 엑시트 코드에서 사용자가 서버에 송신할 데이터를 리턴해야 합니다. 송신 엑시트의 경우 엑시트 코드에서 IBM MQ가 해석할 수정된 데이터를 리턴해야 합니다.

가장 간단한 엑시트 본문은 다음과 같습니다.

```

{ return agentBuffer; }

```

채널 엑시트에서 큐 관리자를 닫지 마십시오.

기존 채널 엑시트 클래스 사용

7.0 이전 IBM MQ 버전에서는 다음 예에서와 같이 MQSendExit, MQReceiveExit 및 MQSecurityExit 인터페이스를 사용하여 이러한 엑시트를 구현합니다. 이 메소드는 계속 유효하지만 향상된 기능과 성능을 위해 새 메소드를 사용하는 것이 좋습니다.

```

public class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {
    // Default constructor
    public MyMQExits(){
    }
    // This method comes from the send exit
    public byte[] sendExit(MQChannelExit channelExitParms,
                           MQChannelDefinition channelDefParms,
                           byte agentBuffer[])
    {
        // Fill in the body of the send exit here
    }
    // This method comes from the receive exit
    public byte[] receiveExit(MQChannelExit channelExitParms,
                              MQChannelDefinition channelDefParms,
                              byte agentBuffer[])
    {
        // Fill in the body of the receive exit here
    }
    // This method comes from the security exit
    public byte[] securityExit(MQChannelExit channelExitParms,
                               MQChannelDefinition channelDefParms,
                               byte agentBuffer[])
    {
        // Fill in the body of the security exit here
    }
}

```

*IBM MQ classes for Java*에서 채널 엑시트 지정

IBM MQ classes for Java를 사용하여 채널 엑시트를 지정할 수 있습니다.

예는 채널에 IBM MQ 직접 대응하는 IBM MQ classes for Java 개념이 없습니다. 채널 엑시트가 MQQueueManager에 지정됩니다. 예를 들어, WMQSecurityExit 인터페이스를 구현하는 클래스를 정의한 경우 애플리케이션에서 다음 네 방법 중 하나로 보안 엑시트를 사용할 수 있습니다.

- MQQueueManager 오브젝트를 작성하기 전에 MQEnvironment.channelSecurityExit 필드에 클래스의 인스턴스를 지정

- MQQueueManager 오브젝트를 작성하기 전에 보안 엑시트 클래스를 표시하는 문자열로 MQEnvironment.channelSecurityExit 필드를 설정
- CMQC.SECURITY_EXIT_PROPERTY 키로 MQQueueManager에 전달된 특성 해시 테이블에 키/값 쌍을 작성
- 클라이언트 채널 정의 테이블(CCDT) 사용

MQEnvironment.channelSecurityExit 필드를 문자열로 설정하거나 특성 해시 테이블에 키/값 쌍을 작성하거나 CCDT를 사용하여 지정된 엑시트는 기본 구성자로 작성되어야 합니다. 클래스의 인스턴스로서 지정된 엑시트는 애플리케이션에 따라 기본 구성자가 필요하지 않습니다.

애플리케이션은 비슷한 방법으로 송신 또는 수신 엑시트를 사용할 수 있습니다. 예를 들어 다음 코드 단편에서는 이전에 MQEnvironment를 사용하여 정의된 MyMQExits 클래스에 구현된 보안, 송신 및 수신 엑시트를 사용하는 방법을 보여줍니다.

```
MyMQExits myexits = new MyMQExits();
MQEnvironment.channelSecurityExit = myexits;
MQEnvironment.channelSendExit = myexits;
MQEnvironment.channelReceiveExit = myexits;
:
MQQueueManager jupiter = new MQQueueManager("JUPITER");
```

채널 엑시트를 지정하는 데 두 개 이상의 메소드가 사용된 경우 다음과 같이 우선순위가 지정됩니다.

1. CCDT의 URL을 MQQueueManager에 전달하는 경우 CCDT 콘텐츠에 따라 사용할 채널 엑시트가 결정되며 MQEnvironment 또는 특성 해시 테이블의 엑시트 정의는 무시됩니다.
2. CCDT URL이 전달되지 않으면 MQEnvironment와 해시 테이블의 엑시트 정의가 병합됩니다.
 - 동일한 엑시트 유형이 MQEnvironment와 해시 테이블에 모두 정의되면 해시 테이블의 정의가 사용됩니다.
 - 동등한 이전 및 새 유형의 엑시트가 지정되면(예: IBM WebSphere MQ 7.0 이전 버전에서 사용되는 엑시트 유형에만 사용할 수 있는 sendExit 필드와 모든 송신 엑시트에 사용할 수 있는 channelSendExit 필드) 이전 엑시트가 아니라 새로운 엑시트(channelSendExit)가 사용됩니다.

채널 엑시트를 문자열로 선언한 경우 IBM MQ를 사용하여 채널 엑시트 프로그램을 찾아야 합니다. 애플리케이션이 실행 중인 환경과 채널 엑시트 프로그램이 패키징되는 방식에 따라 다양한 방식으로 수행할 수 있습니다.

- 애플리케이션 서버에서 실행 중인 애플리케이션의 경우 [361 페이지의 표 58](#)에 표시된 디렉토리 또는 **exitClasspath**로 참조되는 JAR 파일에 패키징된 디렉토리에 파일을 저장해야 합니다.
- 애플리케이션 서버에서 실행 중인 애플리케이션의 경우 다음 규칙이 적용됩니다.
 - 채널 엑시트 클래스가 개별 JAR 파일에 패키징되면 해당 JAR 파일은 **exitClasspath**에 포함되어야 합니다.
 - 채널 엑시트가 JAR 파일로 패키징되지 않으면 [361 페이지의 표 58](#)에 표시된 디렉토리 또는 JVM 시스템 클래스 경로나 **exitClasspath**의 디렉토리에 클래스 파일을 저장할 수 있습니다.

exitClasspath 특성은 다음 네 가지 방법으로 지정할 수 있습니다. 이러한 방법을 우선순위 순으로 표시하면 다음과 같습니다.

1. 시스템 특성 com.ibm.mq.exitClasspath(-D 옵션을 사용하여 명령행에 정의됨)
2. mqclient.ini 파일의 exitPath 스탠자
3. CMQC.EXIT_CLASSPATH_PROPERTY 키가 있는 해시 테이블 항목
4. MQEnvironment 변수 **exitClasspath**

java.io.File.pathSeparator 문자를 사용하여 여러 경로를 분리하십시오.

표 58. 채널 엑시트 프로그램의 디렉토리	
플랫폼	디렉토리
 AIX	/var/mqm/exits(32비트 채널 엑시트 프로그램)
 Linux	/var/mqm/exits64(64비트 채널 엑시트 프로그램)

표 58. 채널 엑시트 프로그램의 디렉토리 (계속)	
플랫폼	디렉토리
Windows	install_data_dir\exits

참고: *install_data_dir*은 설치 중에 IBM MQ 데이터 파일용으로 선택하는 디렉토리입니다. 기본 디렉토리는 C:\ProgramData\ IBM \ MQ입니다.

*IBM MQ classes for Java*의 채널 엑시트에 데이터 전달
채널 엑시트에 데이터를 전달하고 채널 엑시트에서 애플리케이션에 데이터를 리턴할 수 있습니다.

agentBuffer 매개변수

송신 엑시트의 경우 *agentBuffer* 매개변수에는 송신할 데이터가 포함되어 있습니다. 수신 엑시트 또는 보안 엑시트의 경우 *agentBuffer* 매개변수에는 방금 수신한 데이터가 포함되어 있습니다. *agentBuffer.limit()* 표현식은 배열의 길이를 표시하므로 길이 매개변수가 필요하지 않습니다.

송신과 보안 엑시트의 경우 엑시트 코드에서 사용자가 서버에 송신할 데이터를 리턴해야 합니다. 송신 엑시트의 경우 엑시트 코드에서 IBM MQ가 해석할 수정된 데이터를 리턴해야 합니다.

가장 간단한 엑시트 본문은 다음과 같습니다.

```
{ return agentBuffer; }
```

채널 엑시트는 백업 어레이가 있는 버퍼와 함께 호출됩니다. 최상의 성능을 위해 엑시트는 백업 어레이가 있는 버퍼를 리턴해야 합니다.

사용자 데이터

애플리케이션에서 *channelSecurityExit*, *channelSendExit* 또는 *channelReceiveExit*를 설정하여 큐 관리자에 연결하는 경우, 적절한 채널 엑시트 클래스를 호출할 때 *channelSecurityExitUserData*, *channelSendExitUserData* 또는 *channelReceiveExitUserData* 필드를 사용하여 해당 채널 엑시트 클래스에 32 바이트의 사용자 데이터를 전달할 수 있습니다. 이 사용자 데이터는 채널 엑시트 클래스에 사용 가능하지만 엑시트를 호출할 때마다 이 데이터를 새로 고칩니다. 따라서 채널 엑시트의 사용자 데이터에 대한 변경사항이 유실됩니다. 채널 엑시트에서 데이터를 지속적으로 변경하려면 MQCXP *exitUserArea*를 사용하십시오. 이 필드의 데이터는 엑시트 호출 간에 유지보수됩니다.

애플리케이션이 *securityExit*, *sendExit* 또는 *receiveExit*를 설정하면 사용자 데이터를 이러한 채널 엑시트 클래스에 전달할 수 없습니다.

애플리케이션에서 클라이언트 채널 정의 테이블(CCDT)을 사용하여 큐 관리자에 연결하는 경우 채널 엑시트 클래스를 호출할 때 클라이언트 연결 채널 정의에 지정된 사용자 데이터가 채널 엑시트 클래스에 전달됩니다. 클라이언트 채널 정의 테이블 사용에 대한 자세한 정보는 346 페이지의 『IBM MQ classes for Java에서 클라이언트 채널 정의 테이블 사용』의 내용을 참조하십시오.

채널 출구를 사용하지 않고 *Java* 작성된 *IBM MQ classes for Java*
Java 애플리케이션에서 C로 작성된 채널 엑시트 프로그램을 사용하는 방법입니다.

IBM MQ에서, C 언어로 작성된 채널 종료 프로그램    (x86 아키텍처의 64비트 환경에서만 지원)의 이름을 MQEnvironment 객체 또는 속성 해시 테이블의 *channelSecurityExit*, *channelSendExit*, 또는 *channelReceiveExit* 필드에 전달되는 문자열로 지정할 수 있습니다. 그러나 다른 언어로 작성된 애플리케이션에서 Java로 작성된 채널 엑시트를 사용할 수 없습니다.

library(function) 형식의 엑시트 프로그램 이름을 지정하고 엑시트 경로에 설명된 대로 엑시트 프로그램의 위치가 지정되었는지 확인하십시오.

C로 채널 엑시트를 작성하는 방법에 대한 정보는 710 페이지의 『메시지 채널에 대한 채널 엑시트 프로그램』의 내용을 참조하십시오.

*IBM MQ classes for Java*에서 일련의 채널 송신 또는 수신 채널 사용

IBM MQ classes for Java 애플리케이션에서 잇달아 실행되는 일련의 채널 송신 또는 수신 엑시트를 사용할 수 있습니다.

일련의 송신 시퀀스를 사용하기 위해 애플리케이션에서 송신 엑시트를 포함하는 List 또는 String을 작성할 수 있습니다. 목록이 사용되면 List의 각 요소는 다음 중 하나일 수 있습니다.

- WMQSendExit 인스턴스를 구현하는 사용자 정의 클래스의 인스턴스
- MQSendExit 인스턴스를 구현하는 사용자 정의 클래스의 인스턴스(Java로 작성된 송신 엑시트의 경우)
- MQExternalSendExit 클래스의 인스턴스(Java로 기록된 송신 엑시트의 경우)
- MQSendExitChain 클래스의 인스턴스
- String 클래스의 인스턴스

List에는 다른 List가 포함될 수 없습니다.

애플리케이션에서 비슷한 방식으로 일련의 수신 엑시트를 사용할 수 있습니다.

String을 사용하는 경우 하나 이상의 쉼표로 구분된 엑시트 정의로 구성되어야 하며 각각은 Java 클래스의 이름이거나 library(function) 형식의 C 프로그램일 수 있습니다.

그런 다음 애플리케이션에서 MQQueueManager 오브젝트를 작성하기 전에 MQEnvironment.channelSendExit 필드에 List 또는 String 오브젝트를 지정합니다.

엑시트에 전달된 정보의 컨텍스트는 단독으로 엑시트의 도메인에 있습니다. 예를 들어, Java 엑시트와 C 엑시트가 체인 연결된 경우 Java 엑시트가 있어도 C 엑시트에 영향을 미치지 않습니다.

엑시트 체인 클래스 사용

IBM WebSphere MQ 7.0 이전 버전에서는 일련의 엑시트를 허용하도록 다음 두 클래스가 제공되었습니다.

- MQSendExit 인터페이스를 구현하는 MQSendExitChain
- MQReceiveExit 인터페이스를 구현하는 MQReceiveExitChain

이러한 클래스 사용은 여전히 유효하지만 새 메소드가 선호됩니다. Java용 IBM MQ 클래스 인터페이스를 사용한다는 것은 애플리케이션에 여전히 com.ibm.mq.jar에 대한 종속성이 있다는 의미입니다.

com.ibm.mq.exits 패키지의 새 인터페이스 세트가 사용되면 com.ibm.mq.jar에 대한 종속성이 없는 것입니다.

일련의 송신 엑시트를 사용하기 위해 애플리케이션에서 오브젝트 목록을 작성합니다. 여기서 각 오브젝트는 다음 중 하나입니다.

- MQSendExit 인스턴스를 구현하는 사용자 정의 클래스의 인스턴스(Java로 작성된 송신 엑시트의 경우)
- MQExternalSendExit 클래스의 인스턴스(Java로 기록된 송신 엑시트의 경우)
- MQSendExitChain 클래스의 인스턴스

애플리케이션에서 이 오브젝트 목록을 구성자의 매개변수로 전달하여 MQSendExitChain 오브젝트를 작성했습니다. 그런 다음 애플리케이션에서 MQQueueManager 오브젝트를 작성하기 전에 MQEnvironment.sendExit 필드에 MQSendExitChain 오브젝트를 지정했습니다.

IBM MQ classes for Java의 채널 압축

채널에서 이동하는 데이터를 압축하면 채널의 성능이 향상되고 네트워크 트래픽이 감소될 수 있습니다. IBM MQ classes for Java에서는 IBM MQ에 빌드된 압축 기능을 사용합니다.

IBM MQ와 함께 제공된 기능을 사용하여 메시지 채널과 MQI 채널에서 이동하는 데이터를 압축할 수 있으며, 채널의 각 유형에서 헤더 데이터와 메시지 데이터를 서로 독립적으로 압축할 수 있습니다. 기본적으로 채널에서 데이터는 압축되지 않습니다. IBM MQ에서 구현된 방식을 비롯하여 채널 압축에 대한 전체 설명은 [데이터 압축 \(COMPMSG\)](#) 및 [헤더 압축 \(COMPHDR\)](#)을 참조하십시오.

IBM MQ classes for Java 애플리케이션은 java.util.Collection 오브젝트를 작성하여 클라이언트 연결 시 헤더 또는 메시지 데이터를 압축하는 데 사용할 수 있는 기술을 지정합니다. 각 압축 기술은 컬렉션의 Integer 오브젝트이고, 애플리케이션이 컬렉션에 압축 기술을 추가하는 순서는 클라이언트 연결을 시작할 때 큐 관리자와 압축 기술을 협상하는 순서입니다. 그런 다음 애플리케이션이 MQEnvironment class 클래스의 hdrCompList 필드(헤

더 데이터의 경우) 또는 msgCompList 필드(메시지 데이터의 경우)에 콜렉션을 지정할 수 있습니다. 애플리케이션이 준비되면 MQQueueManager 오브젝트를 작성하여 클라이언트 연결을 시작할 수 있습니다.

다음 코드 단편은 설명된 접근 방식을 예시합니다. 첫 번째 코드 단편은 헤더 데이터 압축을 구현하는 방법을 보여줍니다.

```
Collection headerComp = new Vector();
headerComp.add(new Integer(CMQXC.MQCOMPRESS_SYSTEM));
:
MQEnvironment.hdrCompList = headerComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

두 번째 코드 단편은 메시지 데이터 압축을 구현하는 방법을 표시합니다.

```
Collection msgComp = new Vector();
msgComp.add(new Integer(CMQXC.MQCOMPRESS_RLE));
msgComp.add(new Integer(CMQXC.MQCOMPRESS_ZLIBHIGH));
msgComp.add(new Integer(CMQXC.MQCOMPRESS_LZ4HIGH));
:
MQEnvironment.msgCompList = msgComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

두 번째 예에서 클라이언트 연결을 시작할 때 압축 기술은 RLE 및 ZLIBHIGH의 순서로 협상합니다. 선택된 압축 기술은 MQQueueManager 오브젝트의 수명 중에 변경할 수 없습니다.

클라이언트 연결 시 클라이언트와 큐 관리자에서 모두 지원되는 헤더와 메시지 데이터의 압축 기술은 MQChannelDefinition 오브젝트의 hdrCompList 및 msgCompList 필드의 콜렉션으로 채널 엑시트에 전달됩니다. 클라이언트 연결 시 헤더와 메시지 데이터를 압축하는 데 현재 사용 중인 실제 기술은 MQChannelExit 오브젝트의 CurHdrCompression 및 CurMsgCompression 필드로 채널 엑시트에 전달됩니다.

클라이언트 연결 시 압축을 사용하는 경우, 채널 송신 엑시트를 처리하기 전에 데이터가 압축되고 채널 수신 엑시트를 처리한 후 데이터가 추출됩니다. 따라서 송신 및 수신 엑시트에 전달된 데이터는 압축된 상태로 있습니다.

압축 기술 지정 및 사용 가능한 압축 기술에 대한 자세한 정보는 [Class com.ibm.mq.MQEnvironment](#) 및 [Interface com.ibm.mq.MQC](#)를 참조하십시오.

IBM MQ classes for Java에서 TCP/IP 연결 공유

단일 TCP/IP 연결을 공유하도록 MQI 채널의 다중 인스턴스를 작성할 수 있습니다.

IBM MQ classes for Java에서 단일 TCP/IP 연결을 공유할 수 있는 대화의 수를 제어하기 위해 MQEnvironment.sharingConversations 변수를 사용합니다.

SHARECNV 속성은 연결 공유에 대한 최상의 접근 방법입니다. 그러므로 IBM MQ classes for Java에서 0보다 큰 SHARECNV 값을 사용하면 새 연결 요청이 이미 설정된 연결을 공유하지 못할 수도 있습니다.

IBM MQ classes for Java의 연결 풀링

IBM MQ classes for Java를 사용하면 여분의 연결을 재사용하기 위해 풀링할 수 있습니다.

IBM MQ classes for Java에서는 IBM MQ 큐 관리자에 대한 여러 연결을 처리하는 애플리케이션에 추가 지원을 제공합니다. 더 이상 연결이 필요하지 않으면, 연결을 영구 삭제하지 않고 풀링한 후 나중에 다시 사용할 수 있습니다. 따라서 임의의 큐 관리자에 연속으로 연결하는 애플리케이션 및 미들웨어의 성능이 현저히 향상됩니다.

IBM MQ에서는 기본 연결 풀을 제공합니다. 애플리케이션은 MQEnvironment 클래스를 통해 토큰을 등록하고 재등록하여 이 연결 풀을 활성화하거나 비활성화할 수 있습니다. IBM MQ classes for Java를 통해 MQQueueManager 오브젝트를 구성할 때 풀이 활성화되면 이 기본 풀을 검색하고 적당한 연결을 재사용합니다. MQQueueManager.disconnect()를 호출하면 기본 연결이 풀에 리턴됩니다.

또는 애플리케이션에서 특정 용도로 MQSimpleConnectionManager 연결 풀을 구성할 수 있습니다. 그런 다음 애플리케이션에서 MQQueueManager 오브젝트를 구성하는 중에 해당 풀을 지정하거나 기본 연결 풀로 사용하도록 MQEnvironment에 풀을 전달할 수 있습니다.

연결에서 너무 많은 자원을 사용하지 않도록 MQSimpleConnectionManager 오브젝트에서 핸들링할 수 있는 총 연결 수를 제한하고 연결 풀의 크기를 제한할 수 있습니다. JVM 내에서 연결 수요가 충돌하는 경우 한계를 설정하면 유용합니다.

기본적으로 `getMaxConnections()` 메소드는 값 0을 리턴합니다. 즉, `MQSimpleConnectionManager` 오브젝트가 핸들링할 수 있는 연결 수가 제한되지 않습니다. `setMaxConnections()` 메소드를 사용하여 한계를 설정할 수 있습니다. 한계를 설정하고 이 한계에 도달하는 경우 추가 연결을 요청하면 `MQException`이 발생하고 이유 코드로 `MQRC_MAX_CONNS_LIMIT_REACHED`가 표시될 수 있습니다.

*IBM MQ classes for Java*에서 기본 연결 풀 제어
이 예는 기본 연결 풀을 사용하는 방법을 보여줍니다.

다음 예제 애플리케이션, `MQApp1`을 고려하십시오.

```
import com.ibm.mq.*;
public class MQApp1
{
    public static void main(String[] args) throws MQException
    {
        for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
        }
    }
}
```

`MQApp1`은 명령행에서 로컬 큐 관리자 목록을 사용하여 차례로 각각에 연결하고 일부 작업을 수행합니다. 그러나 명령행에 동일한 큐 관리자가 여러 번 나열되면 한 번만 연결한 다음 해당 연결을 여러 번 재사용하는 것이 더 효율적입니다.

*IBM MQ classes for Java*는 이 작업을 수행하는 데 사용할 수 있는 기본 연결 풀을 제공합니다. 풀을 사용하려면 `MQEnvironment.addConnectionPoolToken()` 메소드 중 하나를 사용하십시오. 풀을 사용 안함으로 설정하려면 `MQEnvironment.removeConnectionPoolToken()`을 사용하십시오.

다음 예제 애플리케이션, `MQApp2`는 기능 면에서 `MQApp1`과 동일하지만 각 큐 관리자에 한 번만 연결합니다.

```
import com.ibm.mq.*;
public class MQApp2
{
    public static void main(String[] args) throws MQException
    {
        MQPoolToken token=MQEnvironment.addConnectionPoolToken();

        for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
        }

        MQEnvironment.removeConnectionPoolToken(token);
    }
}
```

첫 번째 굵은 선은 `MQEnvironment`에 `MQPoolToken` 오브젝트를 등록하여 기본 연결 풀을 활성화합니다.

이제 `MQQueueManager` 구성자가 이 풀에서 적절한 연결을 검색하고 기존 연결을 찾을 수 없는 경우에만 큐 관리자에 대한 연결을 작성합니다. `qmgr.disconnect()` 호출은 나중에 재사용하도록 풀에 대한 연결을 리턴합니다. 이러한 API 호출은 샘플 애플리케이션 `MQApp1`과 동일합니다.

두 번째 강조표시된 행은 기본 연결 풀을 비활성화하므로, 풀에 저장된 큐 관리자 연결이 영구 삭제됩니다. 그렇지 않으면 풀에 있는 여러 라이브 큐 관리자 연결이 종료되므로 이 내용은 중요합니다. 이 상황으로 인해 큐 관리자 로그에 표시되는 오류가 발생할 수 있습니다.

애플리케이션에서 클라이언트 채널 정의 테이블(CCDT)을 사용하여 큐 관리자에 연결하는 경우 `MQQueueManager` 구성자가 먼저 테이블에서 적당한 클라이언트 연결 채널 정의를 검색합니다. 하나가 발견되면 구성자가 기본 연결 풀에서 채널에 사용할 수 있는 연결을 검색합니다. 구성자가 풀에서 적당한 연결을 찾을 수 없으면 클라이언트 채널 정의 테이블에서 다음으로 적당한 클라이언트 연결 채널 정의를 검색하고 이전에 설명한 대로 진행합니다. 구성자가 클라이언트 채널 정의 테이블 검색을 완료하고 풀에서 적당한 연결을 찾지 못하

면 구성자가 두 번째 테이블 검색을 시작합니다. 이 검색 중에 구성자가 적당한 각 클라이언트 연결 채널 정의에 대한 새 연결을 차례로 작성하고 작성한 첫 번째 연결을 사용합니다.

기본 연결 풀은 최대 10개의 미사용 연결을 저장하고 최대 5분 동안 미사용 연결을 활성으로 유지합니다. 애플리케이션에서 이 동작을 대체합니다(자세한 내용은 [367 페이지의 『IBM MQ classes for Java에서 다른 연결 풀 제공』](#) 참조).

MQEnvironment를 사용하여 MQPoolToken을 제공하지 않고 애플리케이션이 다음과 같이 고유하게 구성합니다.

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

일부 애플리케이션 또는 미들웨어 벤더가 사용자 정의 연결 풀에 정보를 전달하기 위해 MQPoolToken의 서브클래스를 제공합니다. 추가 정보를 연결 풀에 전달할 수 있도록 이 방식으로 구성하여 addConnectionPoolToken()에 전달할 수 있습니다.

IBM MQ classes for Java의 기본 연결 풀 및 다중 컴포넌트

이 예에서는 등록된 MQPoolToken 오브젝트의 정적 세트에서 MQPoolTokens를 추가 또는 제거하는 방법을 보여줍니다.

MQEnvironment는 등록된 MQPoolToken 오브젝트의 정적 세트를 보유합니다. 이 세트에서 MQPoolTokens를 추가 또는 제거하려면 다음 메소드를 사용하십시오.

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

애플리케이션은 독립적으로 존재하고 큐 관리자를 사용하여 작업을 수행하는 여러 컴포넌트로 구성될 수 있습니다. 이러한 애플리케이션에서 각 컴포넌트가 설정된 MQEnvironment에 MQPoolToken을 추가해야 합니다.

예를 들어, 예제 애플리케이션 MQApp3은 10개의 스레드를 작성하고 각 스레드를 시작합니다. 각 스레드가 고유 MQPoolToken을 등록하고 일정 기간 동안 대기한 다음 큐 관리자에 연결합니다. 스레드의 연결을 끊은 다음 고유 MQPoolToken을 제거합니다.

MQPoolTokens 세트에 하나 이상의 토큰이 있는 동안에는 기본 연결 풀이 활성인 상태로 남아 있으므로, 이 애플리케이션의 지속 기간 동안 활성 상태로 남아 있게 됩니다. 애플리케이션은 스레드의 전체 제어에서 마스터 오브젝트를 유지할 필요가 없습니다.

```
import com.ibm.mq.*;
public class MQApp3
{
    public static void main(String[] args)
    {
        for (int i=0; i<10; i++) {
            MQApp3_Thread thread=new MQApp3_Thread(i*60000);
            thread.start();
        }
    }
}

class MQApp3_Thread extends Thread
{
    long time;

    public MQApp3_Thread(long time)
    {
        this.time=time;
    }

    public synchronized void run()
    {
        MQPoolToken token=MQEnvironment.addConnectionPoolToken();
        try {
            wait(time);
            MQQueueManager qmgr=new MQQueueManager("my.qmgr.1");
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
        }
        catch (MQException mqe) {System.err.println("Error occurred!");}
```

```

        catch (InterruptedException ie) {}

        MQEnvironment.removeConnectionPoolToken(token);
    }
}

```

*IBM MQ classes for Java*에서 다른 연결 풀 제공

이 예에서는 **com.ibm.mq.MQSimpleConnectionManager** 클래스를 사용하여 다른 연결 풀을 제공하는 방식을 보여줍니다.

이 클래스에서는 연결 풀링의 기본 기능을 제공하고 애플리케이션에서 이 클래스를 사용하여 풀의 동작을 사용자 정의할 수 있습니다.

MQSimpleConnectionManager를 인스턴스화하고 나면 MQQueueManager 구성자에 지정할 수 있습니다. 그러면 MQSimpleConnectionManager에서 구성된 MQQueueManager의 기본이 되는 연결을 관리합니다.

MQSimpleConnectionManager에 적당하게 풀링된 연결이 포함되어 있으면 MQQueueManager.disconnect() 호출 후에 연결이 재사용되며 MQSimpleConnectionManager에 리턴됩니다.

다음 코드 단편은 이 동작을 보여줍니다.

```

MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr)
:
qmgr.disconnect();

MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr2)
:
qmgr2.disconnect();
myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);

```

첫 번째 MQQueueManager 구성자 호출 중에 작성된 연결은 qmgr.disconnect() 호출 후 myConnMan에 저장됩니다. 그런 다음 두 번째로 MQQueueManager 구성자를 호출하는 동안 연결을 재사용합니다.

두 번째 행에서 MQSimpleConnectionManager를 사용하게 설정합니다. 마지막 행은 MQSimpleConnectionManager를 사용하지 않게 설정하여 풀에 보유된 연결을 영구 삭제합니다. 기본적으로 MQSimpleConnectionManager는 이 섹션의 뒷 부분에 설명된 MODE_AUTO에 있습니다.

MQSimpleConnectionManager는 가장 최근에 사용된 연결을 할당하고 가장 늦게 사용된 연결을 제거합니다. 기본적으로 5분 동안 연결을 사용하지 않거나 풀에 미사용 연결이 10개가 넘으면 연결을 영구 삭제합니다.

MQSimpleConnectionManager.setTimeout()을 호출하여 이러한 값을 변경할 수 있습니다.

또한 기본 연결 풀로 사용할 MQSimpleConnectionManager를 MQQueueManager 구성자에 연결 관리자가 제공되지 않을 때 사용하도록 구성할 수 있습니다.

다음 애플리케이션에서 이를 보여줍니다.

```

import com.ibm.mq.*;
public class MQApp4
{
    public static void main(String []args)
    {
        MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
        myConnMan.setActive(MQSimpleConnectionManager.MODE_AUTO);
        myConnMan.setTimeout(3600000);
        myConnMan.setMaxConnections(75);
        myConnMan.setMaxUnusedConnections(50);
        MQEnvironment.setDefaultConnectionManager(myConnMan);
        MQApp3.main(args);
    }
}

```

짧은 행에서 MQSimpleConnectionManager 오브젝트를 작성하고 구성합니다. 이 구성은 다음을 수행합니다.

- 1시간 동안 사용하지 않은 연결을 종료합니다.
- myConnMan에서 관리하는 연결 수를 75로 제한합니다.

- 풀에서 사용하지 않는 연결 수를 50으로 제한합니다.
- 기본값인 `MODE_AUTO`를 설정합니다. 즉, 풀은 기본 연결 관리자인 경우에만 활성이며, `MQEnvironment`에서 보유한 `MQPoolTokens` 세트에 하나 이상의 토큰이 있습니다.

그런 다음 새 `MQSimpleConnectionManager`가 기본 연결 관리자로 설정됩니다.

마지막 행에서 애플리케이션은 `MQApp3.main()`을 호출합니다. 이는 각 스레드가 독립적으로 IBM MQ을(를) 사용하는 여러 스레드를 실행합니다. 이러한 스레드에서 연결을 작성할 때 `myConnMan`을 사용합니다.

IBM MQ classes for Java를 사용하여 JTA/JDBC 통합

IBM MQ classes for Java는 IBM MQ가 JDBC 유형 2 또는 JDBC 유형 4 준수 드라이버를 제공하는 통합자의 역할을 수행하는 데 사용할 수 있는 `MQQueueManager.begin()` 메소드를 지원합니다.

이 지원은 모든 플랫폼에서 사용할 수 없습니다. JDBC 통합을 지원하는 플랫폼을 확인하려면 [시스템 요구 사항 IBM MQ](#)의 내용을 참조하십시오.

XA-JTA 지원을 사용하려면 특수 JTA 스위치 라이브러리를 사용해야 합니다. 이 라이브러리를 사용하는 메소드는 Windows를 사용하는지 아니면 다른 플랫폼 중 하나를 사용하는지에 따라 달라집니다.

Windows에 JTA/JDBC 통합 구성

XA 라이브러리는 `jdbcxxx.dll` 형식의 이름을 사용하여 DLL로 제공됩니다.

제공된 구성 요소는 `Oracle12Cjdbcora12.dll` 와의 호환성을 제공하여 서버 Windows 설치용 IBM MQ 구성 요소입니다.

Windows 시스템에서 XA 라이브러리는 전체 DLL로 제공됩니다. 이 DLL의 이름은 `jdbcxxx.dll`입니다. 여기서 `xxx`는 스위치 라이브러리가 컴파일된 데이터베이스를 나타냅니다. 이 라이브러리는 설치된 IBM MQ classes for Java 의 또는 `java\lib64\jdbc java\lib\jdbc` 디렉터리에 있습니다. 스위치 로드 파일로도 설명된 XA 라이브러리를 큐 관리자에 선언해야 합니다. IBM MQ Explorer를 사용합니다. XA 자원 관리자 아래에서, 큐 관리자 특성 패널에서 스위치 로드 파일의 세부사항을 지정하십시오. 라이브러리 이름만 지정해야 합니다. 예를 들면, 다음과 같습니다.

Db2 데이터베이스의 경우 SwitchFile 필드를 `dbcdb2`로 설정하십시오.

Oracle 데이터베이스의 경우 SwitchFile 필드를 `jdbcora`로 설정하십시오.

참고:

1. Oracle 12C 에 의해 IBM MQ classes for Java지원되며, 오직 에서 IBM MQ 에 대해서만 지원됩니다 Windows.
2. 지원되는 Oracle 12C 버전은 12.1.0.1.0 Enterprise Edition 및 향후 수정팩입니다.
3. 64비트 Windows의 Oracle 64비트 데이터베이스는 32비트 Oracle 클라이언트를 필요로 합니다.
4. 를 사용하여 IBM MQ classes for Java, 는 트랜잭션 조정자 IBM MQ 역할을 수행할 수 있습니다. 그러나 JTA 스타일의 트랜잭션에는 참여할 수 없습니다.

Windows 이외의 플랫폼에서 JTA/JDBC 통합 구성

오브젝트 파일은 제공됩니다. 제공된 `make` 파일을 사용하여 적절한 파일을 링크하고 구성 파일을 사용하여 큐 관리자에 선언하십시오.

각 데이터베이스 관리 시스템마다, IBM MQ 스위치 라이브러리를 생성하기 위해 링크해야 하는 객체 파일을 제공합니다. 을 위한 Db2 각 객체 파일의 이름은 다음 같습니다. `jdbcdb2.o` 그리고 ~을 위해 Oracle 각 오브젝트 파일의 이름은 다음과 같습니다 `jdbcora.o`.

IBM MQ에서 제공하는 적절한 `make` 파일을 사용하여 각 오브젝트 파일을 링크해야 합니다. 스위치 라이브러리에는 여러 다른 시스템의 여러 다른 위치에 저장해야 하는 다른 라이브러리가 필요합니다. 그러나 스위치 라이브러리는 `setuid` 환경에서 실행되는 큐 관리자에 의해 로드되므로 라이브러리 경로 환경 변수를 사용하여 이러한 라이브러리를 찾을 수 없습니다. 따라서 제공된 `make` 파일을 사용하면 스위치 라이브러리에 이러한 라이브러리의 완전한 경로 이름이 포함됩니다.

스위치 라이브러리를 생성하려면 해당 /java/lib64/jdbc 디렉터리에서 다음 형식의 명령어를 **make** 입력하십시오.

```
make DBMS
```

여기서 **DBMS** 는 스위치 라이브러리를 생성하려는 데이터베이스 관리 시스템입니다. Oracle 유효한 값은 의 경우 Db2 이고, oracle 의 경우 입니다 db2 .

Oracle이 명령은 디렉터리에 /java/lib64/jdbc 라는 jdbcdb2 이름의 스위치 라이브러리를 생성하며, 용으로는 Db2 를, jdbcora용으로는 를 사용합니다.

참고:

- 시스템에서는 어디에서나 Oracle을(를) 설치할 수 있으므로 makefile에서는 **ORACLE_HOME** 환경 변수를 사용하여 Oracle이(가) 설치된 위치를 찾습니다.
- IBM MQ이(가) 기본 위치가 아닌 위치에 설치되면, makefile에서 **MQ_INSTALLATION_PATH** 값을 변경하십시오.

Db2에 대한 스위치 라이브러리, Oracle 또는 이 모두를 작성한 후에는 이를 큐 관리자에 선언해야 합니다. 큐 관리자 구성 파일(qm.ini)에 이미 또는 Oracle Db2 데이터베이스에 대한 스탠자가 XAResourceManager 포함 되어 있는 경우, 각 스탠자 내의 항목을 SwitchFile 다음 중 하나로 대체해야 합니다:

Db2 데이터베이스

```
SwitchFile=jdbcdb2
```

Oracle 데이터베이스의 경우

```
SwitchFile=jdbcora
```

스위치 라이브러리(32비트 또는 64비트)의 전체 경로명을 지정하지 마십시오. 라이브러리의 이름만 지정하십시오.

큐 관리자 구성 파일에 또는 Oracle Db2 데이터베이스에 대한 스탠자가 XAResourceManager 아직 포함되어 있지 않거나, 추가 XAResourceManager 스탠자를 추가하려는 경우, XAResourceManager 스탠자 구성 방법에 대한 정보는 를 [관리 IBM MQ](#) 참조하십시오. 그러나 새 XAResourceManager 스탠자에 있는 각 SwitchFile 항목은 이전에 Db2 또는 Oracle 데이터베이스에 대해 설명한 그대로여야 합니다. ThreadOfControl=PROCESS 항목도 포함해야 합니다.

큐 관리자 구성 파일을 업데이트한 후 해당 데이터베이스 환경 변수를 모두 설정하고 나면 큐 관리자를 다시 시작할 수 있습니다.

JTA/JDBC 통합 사용

제공된 예제와 같이 API 호출을 코딩하십시오.

사용자 애플리케이션의 기본 API 호출 순서는 다음과 같습니다.

```
qMgr = new MQQueueManager("QM1")
Connection con = qMgr.getJDBCConnection( xads );
qMgr.begin()

< Perform MQ and DB operations to be grouped in a unit of work >

qMgr.commit() or qMgr.backout();
con.close()
qMgr.disconnect()
```

getJDBCConnection 호출에서 xads은 XADatasource 인터페이스의 데이터베이스 특정 구현이며, 연결할 데이터베이스의 자세한 내용을 정의합니다. getJDBCConnection에 전달할 적절한 XADatasource 오브젝트를 작성하는 방법을 판별하려면 데이터베이스 문서를 참조하십시오.

JDBC 작업을 수행하는 데 적절한 데이터베이스 특정 jar 파일로 클래스 경로도 업데이트해야 합니다.

여러 애플리케이션에 연결해야 하는 경우 getJDBCConnection을 여러 번 호출하여 여러 다른 연결에서 트랜잭션을 수행해야 합니다.

XADataSource.getXAConnection의 두 양식을 반영하는 두 가지 양식의 getJDBCConnection이 있습니다.

```
public java.sql.Connection getJDBCConnection(javax.sql.XADataSource xads)
    throws MQException, SQLException, Exception

public java.sql.Connection getJDBCConnection(XADataSource dataSource,
                                             String userid, String password)
    throws MQException, SQLException, Exception
```

이러한 메소드는 JTA 함수를 사용하지 않는 고객의 JVM 검증자에 문제점이 발생하지 않도록 throws 절에 Exception을 선언합니다. 발생한 실제 예외는 javax.transaction.xa.XAException이며, 이전에 필요하지 않은 프로그램의 클래스 경로에 jta.jar 파일을 추가해야 합니다.

JTA/JDBC 지원을 사용하려면 다음 명령문을 애플리케이션에 포함시켜야 합니다.

```
MQEnvironment.properties.put(CMQC.THREAD_AFFINITY_PROPERTY, new Boolean(true));
```

JTA/JDBC 통합의 알려진 문제점과 제한사항

JTA/JDBC 지원 문제점과 제한사항은 사용 중인 데이터베이스 관리 시스템에 따라 달라집니다. 예를 들어, 테스트된 JDBC 드라이버는 애플리케이션이 실행 중인 동안 데이터베이스를 시스템 종료할 때 서로 다르게 작동합니다. 애플리케이션에서 사용 중인 데이터베이스에 대한 연결이 중단되면 애플리케이션이 큐 관리자와 데이터베이스에 대한 새 연결을 재설정하기 위해 수행할 수 있는 단계가 있습니다. 그러면 이 새 연결을 사용하여 필수 트랜잭션 작업을 수행할 수 있습니다.

JTA/JDBC 지원에서 JDBC 드라이버를 호출하므로 해당 JDBC 드라이버를 구현하면 시스템 동작에 상당한 영향을 미칠 수 있습니다. 특히 테스트된 JDBC 드라이버는 애플리케이션을 실행하는 동안 데이터베이스를 종료할 때 서로 다르게 작동합니다.

중요사항: 애플리케이션에서 열린 연결을 보유 중인 동안에는 갑자기 데이터베이스를 종료하지 마십시오.

참고: 애플리케이션은 IBM MQ classes for Java 바인딩 모드를 사용하여 연결해야만 데이터베이스 코디네이터 역할을 수행할 IBM MQ 수 있습니다.

다중 XAResourceManager 스탠자

큐 관리자 구성 파일, qm.ini에서 둘 이상의 XAResourceManager 스탠자를 사용하는 것은 지원되지 않습니다. 첫 번째를 제외한 모든 XAResourceManager 스탠자는 무시됩니다.

DB2

Db2에서 때때로 SQL0805N 오류를 리턴합니다. 이 문제점은 다음 CLP 명령을 사용하여 해결할 수 있습니다.

```
DB2 bind @db2cli.lst blocking all grant public
```

자세한 정보는 Db2 문서를 참조하십시오.

ThreadOfControl=PROCESS를 사용하도록 XAResourceManager 스탠자를 구성해야 합니다. Db2 8.1 이상에서 Db2의 기본 제어 설정 스레드와 일치하지 않으므로 XA Open String에 toc=p를 지정해야 합니다. JTA/JDBC 통합을 사용하는 Db2의 예제 XAResourceManager 스탠자는 다음과 같습니다.

```
XAResourceManager:
  Name=jdbcdb2
  SwitchFile=jdbcdb2
  XAOpenString=uid=userid,db=dbalias,pwd=password,toc=p
  ThreadOfControl=PROCESS
```

JTA/JDBC 통합을 사용하는 Java 애플리케이션이 여전히 멀티스레드를 사용합니다.

Oracle

MQQueueManager.disconnect()에서 SQLException을 생성한 후 JDBC Connection.close() 메소드를 호출합니다. MQQueueManager.disconnect() 전에 Connection.close()를 호출하거나 Connection.close() 호출을 생략하십시오.

데이터베이스 연결 문제 핸들링

IBM MQ classes for Java 애플리케이션에서 IBM MQ가 제공하는 JTA/JDBC 지원을 사용할 때 일반적으로 다음 단계를 수행합니다.

1. 트랜잭션 관리자로 작동할 큐 관리자에 대한 연결을 표시하는 새로운 MQQueueManager 오브젝트를 작성합니다.
2. 트랜잭션에서 수집될 데이터베이스에 연결하는 방법에 관한 자세한 내용을 포함하는 XADataSource 오브젝트를 구성합니다.
3. MQQueueManager.getJDBCConnection(XADataSource) 메소드를 호출하여 이전에 작성된 XADataSource를 전달합니다. 그러면 IBM MQ classes for Java가 데이터베이스 연결합니다.
4. MQQueueManager.begin() 메소드를 호출하여 XA 트랜잭션을 시작합니다.
5. 메시징 및 데이터베이스 작업을 수행합니다.
6. 모든 필수 작업이 완료되면 MQQueueManager.commit() 메소드를 호출합니다. 그러면 XA 트랜잭션이 완료됩니다.
7. 이때 새로운 XA 트랜잭션이 필요하면 애플리케이션에서 4, 5 및 6단계를 반복할 수 있습니다.
8. 애플리케이션이 완료되며 3단계에서 작성한 데이터베이스 연결을 닫은 다음 MQQueueManager.disconnect() 메소드를 호출하여 큐 관리자에서 연결을 끊어야 합니다.

IBM MQ classes for Java는 애플리케이션에서 MQQueueManager.getJDBCConnection(XADataSource)를 호출할 때 작성된 모든 데이터베이스 연결의 내부 목록을 유지보수합니다. 큐 관리자가 XA 트랜잭션을 처리하는 중에 데이터베이스와 통신해야 하는 경우 다음이 처리됩니다.

1. 큐 관리자가 IBM MQ classes for Java를 호출하여 데이터베이스에 전달해야 하는 XA 호출의 자세한 내용을 전달합니다.
2. 그런 다음 IBM MQ classes for Java가 목록에서 적절한 연결을 검색한 후 해당 연결을 사용하여 데이터베이스에 대한 XA 호출 플로우를 이행합니다.

이 처리 중에 데이터베이스에 대한 연결이 유실되면 애플리케이션이 다음을 수행해야 합니다.

1. MQQueueManager.backout() 메소드를 호출하여 트랜잭션을 통해 수행된 기존 작업을 백아웃합니다.
2. 데이터베이스 연결을 닫습니다. 그러면 IBM MQ classes for Java가 내부 목록에서 중단된 데이터베이스 연결의 자세한 내용을 제거해야 합니다.
3. MQQueueManager.disconnect() 메소드를 호출하여 큐 관리자에서 연결을 끊습니다.
4. 새 MQQueueManager 오브젝트를 구성하여 큐 관리자에 대한 새 연결을 설정합니다.
5. MQQueueManager.getJDBCConnection(XADataSource) 메소드를 호출하여 새로운 데이터베이스 연결을 작성합니다.
6. 트랜잭션 작업을 다시 수행합니다.

그러면 애플리케이션이 큐 관리자와 데이터베이스에 대한 새 연결을 재설정한 다음 해당 연결을 사용하여 필요한 트랜잭션 작업을 수행할 수 있습니다.

IBM MQ classes for Java의 TLS(Transport Layer Security)

IBM MQ classes for Java 클라이언트 애플리케이션은 TLS 암호화를 지원합니다. JSSE 제공자가 TLS 암호화를 사용해야 합니다.

TRANSPORT(CLIENT)를 사용하는 IBM MQ classes for Java 클라이언트 애플리케이션에서 TLS 암호화를 지원합니다. TLS는 통신 암호화, 인증 및 메시지 무결성을 제공합니다. 이는 일반적으로 인트라넷 내의 또는 인터넷의 두 피어 간의 보안 통신에 사용됩니다.

IBM MQ classes for Java TLS 암호화를 처리하기 위해 (JSSE)를 Java Secure Socket Extension 사용하므로 JSSE provider가 필요합니다. Java SE v1.4Java Virtual Machine (JVM)에는 내장된 제공자가 JSSE 있습니다. 인증서를 관리하고 저장하는 방법의 세부사항은 제공자마다 다를 수 있습니다. 이에 대한 자세한 내용은 JSSE 공급업체의 설명서를 참조하십시오.

이 섹션은 공급자가 JSSE 올바르게 설치 및 구성되었으며, 적절한 인증서가 설치되어 공급자가 JSSE 사용할 수 있도록 설정되었음을 전제로 합니다.

IBM MQ classes for Java 클라이언트 애플리케이션에서 클라이언트 채널 정의 테이블(CCDT)을 사용하여 큐 관리자에 연결하는 경우 [346 페이지의 『IBM MQ classes for Java에서 클라이언트 채널 정의 테이블 사용』](#)의 내용을 참조하십시오.

IBM MQ classes for Java에서 TLS 사용

TLS를 사용하기 위해 CipherSuite를 지정합니다. 두 가지 방법으로 CipherSuite를 지정할 수 있습니다.

TLS는 클라이언트 연결에만 지원됩니다. TLS를 사용하려면 큐 관리자와 통신할 때 사용할 CipherSuite를 지정해야 하며, 이 CipherSuite는 대상 채널에 설정된 CipherSpec과 일치해야 합니다. 또한, 명시된 CipherSuite 은 귀하의 Java Secure Socket Extension (JSSE) 제공자가 지원해야 합니다. 그러나 CipherSuites는 CipherSpecs과 구분되도록 다른 이름이 지정됩니다. [375 페이지의 『IBM MQ classes for Java의 TLS CipherSpecs 및 CipherSuites』](#) CipherSpecs 가 IBM MQ 지원하는 객체 유형을 해당 객체 유형의 동등한 JSSECipherSuites 로 매핑하는 테이블을 포함합니다.

TLS를 사용하려면 MQEnvironment의 sslCipherSuite 정적 멤버 변수를 사용하여 CipherSuite를 지정하십시오. 다음 예는 TLS_RSA_WITH_AES_128_CBC_SHA256의 CipherSpec인 TLS가 필요하도록 설정된 SECURE.SVRCONN.CHANNEL이라고 이름 지정된 SVRCONN 채널에 첨부됩니다.

```
MQEnvironment.hostname      = "your_hostname";
MQEnvironment.channel       = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_WITH_AES_128_CBC_SHA256";
MQQueueManager qmgr = new MQQueueManager("your_q_manager");
```

채널이 TLS_RSA_WITH_AES_128_CBC_SHA256의 CipherSpec을 가지고 있더라도 Java 애플리케이션은 SSL_RSA_WITH_AES_128_CBC_SHA256의 CipherSuite를 지정해야 합니다. CipherSpec 및 CipherSuite 사이의 매핑 목록을 보려면 [375 페이지의 『IBM MQ classes for Java의 TLS CipherSpecs 및 CipherSuites』](#)의 내용을 참조하십시오.

환경 특성 CMQC.SSL_CIPHER_SUITE_PROPERTY를 설정하는 방법을 통해서도 애플리케이션이 CipherSuite를 지정할 수 있습니다.

또는 클라이언트 채널 테이블(CCDT)을 사용하십시오. 자세한 정보는 [346 페이지의 『IBM MQ classes for Java에서 클라이언트 채널 정의 테이블 사용』](#)의 내용을 참조하십시오.

CipherSuite 을 사용하기 위해 클라이언트 연결이 필요한 경우, 해당 IBMJavaJSSEFIPS 제공자(IBMJSSEFIPS)가 지원하는 애플리케이션은 MQEnvironment 클래스의 sslFipsRequired 필드를 true로 설정할 수 있습니다. 또는 애플리케이션이 환경 특성 CMQC.SSL_FIPS_REQUIRED_PROPERTY를 설정할 수도 있습니다. 기본값은 false이며 클라이언트 연결이 IBM MQ에 의해 지원되는 모든 CipherSuite를 사용할 수 있음을 의미합니다.

애플리케이션이 두 개 이상의 클라이언트 연결을 사용하는 경우, 애플리케이션이 첫 번째 클라이언트 연결을 작성할 때 사용되던 sslFipsRequired 필드의 값이 모든 후속 클라이언트 연결을 작성할 때 사용되는 값을 결정합니다. 따라서 애플리케이션이 후속 클라이언트 연결을 작성할 때 sslFipsRequired 필드의 값은 무시됩니다. sslFipsRequired 필드에 다른 값을 사용하려면 애플리케이션을 재시작해야 합니다.

TLS 를 사용하여 성공적으로 연결하려면, 큐 관리자에서 제시하는 인증서를 인증할 수 있는 인증 기관 루트 인증서로 JSSE 트러스트스토어를 설정해야 합니다. 마찬가지로 SVRCONN 채널의 SSLClientAuth가 MQSSL_CLIENT_AUTH_REQUIRED로 설정된 경우 JSSE 키 저장소에는 큐 관리자가 신뢰하는 식별 인증서가 있어야 합니다.

관련 참조

[Federal Information Processing Standards \(FIPS\)에 대해 AIX, Linux, and Windows](#)

IBM MQ classes for Java에서 큐 관리자의 식별 이름 사용

큐 관리자는 식별 이름(DN)을 포함하는 TLS 인증서를 사용하여 식별됩니다. IBM MQ classes for Java 클라이언트 애플리케이션은 이 DN을 사용하여 올바른 큐 관리자와 통신 중인지 확인할 수 있습니다.

DN 패턴은 MQEnvironment의 sslPeerName 변수를 사용하여 지정됩니다. 예를 들어,

```
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSphere";
```

큐 관리자가 QMGR로 시작하는 공통 이름의 인증서를 제공하는 경우에만 연결이 성공할 수 있습니다. 최소한 두 개의 조직 단위 이름이 있어야 합니다. 첫 번째 이름은 IBM 이고 두 번째 이름은 WebSphere이어야 합니다.

sslPeerName이 설정된 경우, 올바른 패턴으로 설정되고 큐 관리자가 일치하는 인증서를 제공하는 경우에만 연결에 성공합니다.

애플리케이션에서 CMQC.SSL_PEER_NAME_PROPERTY 환경 속성도 설정하여 큐 관리자의 식별 이름을 지정할 수 있습니다. 식별 이름에 대한 자세한 정보는 [식별 이름](#)을 참조하십시오.

*IBM MQ classes for Java*에서 인증서 폐기 목록 사용

java.security.cert.CertStore 클래스를 통해 사용하려면 인증서 폐기 목록을 지정하십시오. 그러면 IBM MQ classes for Java에서 지정된 CRL에 대해 인증서를 검사합니다.

인증서 폐기 목록(CRL)은 인증 기관을 발행하거나 로컬 조직을 통해 폐기된 인증서 세트입니다. 일반적으로 CRL은 LDAP 서버에서 호스팅됩니다. Java 2 v1.4를 사용하면 연결 시 CRL 서버를 지정할 수 있고 연결을 허용하기 전에 큐 관리자가 제공한 인증서를 CRL과 비교하여 확인할 수 있습니다. 인증서 폐기 목록 및 IBM MQ에 대한 자세한 정보는 [인증서 폐기 목록 및 권한 폐기 목록에 대한 작업](#) 및 [IBM MQ classes for Java](#) 및 [IBM MQ classes for JMS](#)로 CRL 및 ARL에 액세스를 참조하십시오.

참고: LDAP 서버에서 호스팅된 CRL에서 CertStore를 성공적으로 사용하려면 Java SDK(Software Development Kit)가 CRL과 호환 가능한지 확인하십시오. 일부 SDK에서는 LDAP v2에 대한 스키마를 정의하는 RFC 2587을 CRL이 준수하도록 요구합니다. 대부분의 LDAP v3 서버는 RFC 2256을 대신 사용합니다.

사용할 CRL은 java.security.cert.CertStore 클래스를 통해 지정됩니다. CertStore의 인스턴스를 확보하는 방법에 대한 자세한 정보를 보려면 이 클래스의 문서를 참조하십시오. LDAP 서버를 기반으로 하여 CertStore를 작성하려면 먼저 사용할 서버 및 포트 설정을 사용하여 초기화된 LDAPCertStoreParameters 인스턴스를 작성해야 합니다. 예를 들면, 다음과 같습니다.

```
import java.security.cert.*;
CertStoreParameters csp = new LDAPCertStoreParameters("crl_server", 389);
```

CertStoreParameters 인스턴스를 작성하고 CertStore에서 정적 구성자를 사용하여 LDAP 유형의 CertStore를 작성하십시오.

```
CertStore cs = CertStore.getInstance("LDAP", csp);
```

기타 CertStore 유형(예를 들어, 콜렉션)도 지원됩니다. 일반적으로 중복을 제공하기 위해 동일한 CRL 정보를 사용하여 설정된 CRL 서버가 여러 개 있습니다. 이러한 각 CRL 서버에 대해 CertStore 오브젝트가 있는 경우, 적당한 콜렉션에 모두 배치하십시오. 다음은 ArrayList에 배치된 CertStore 오브젝트를 표시하는 예제입니다.

```
import java.util.ArrayList;
Collection crls = new ArrayList();
crls.add(cs);
```

이 콜렉션은 CRL 검사를 사용하기 전에 MQEnvironment 정적 변수, sslCertStores에 설정할 수 있습니다.

```
MQEnvironment.sslCertStores = crls;
```

연결이 설정될 때 큐 관리자에 의해 표시되는 인증서는 다음과 같이 유효성 검증됩니다.

1. sslCertStores에 의해 식별되는 콜렉션 내의 첫 번째 CertStore 오브젝트는 CRL 서버를 식별하는 데 사용됩니다.
2. CRL 서버에 접속하려는 시도가 이루어집니다.
3. 시도가 성공하면 일치하는 인증서를 찾기 위해 서버가 검색됩니다.
 - a. 인증서가 폐기된 것으로 밝혀지면 검색 프로세스가 종료되며 이유 코드 MQRC_SSL_CERTIFICATE_REVOKED와 함께 연결 요청이 실패합니다.
 - b. 인증서를 찾지 못하는 경우에는 검색 프로세스가 종료되며 연결을 진행하도록 허용됩니다.
4. 서버에 접속하려는 시도가 실패하면 다음 CertStore 오브젝트를 사용하여 CRL 서버를 식별하며 프로세스가 2단계부터 반복됩니다.

콜렉션 내의 마지막 CertStore인 경우 또는 콜렉션에 CertStore 오브젝트가 없는 경우, 검색 프로세스가 실패하고 이유 코드 MQRC_SSL_CERT_STORE_ERROR와 함께 연결 요청이 실패합니다.

콜렉션 오브젝트는 CertStores가 사용되는 순서를 판별합니다.

CertStores의 컬렉션은 CMQC.SSL_CERT_STORE_PROPERTY를 사용하는 방법으로도 설정할 수 있습니다. 편의상 이 특성 또한 컬렉션의 멤버가 되지 않고 단일 CertStore가 지정될 수 있도록 허용합니다.

sslCertStores가 널로 설정되면 CRL 검사가 수행되지 않습니다. CipherSuite가 설정되지 않으면 특성이 무시됩니다.

*IBM MQ classes for Java*에서 비밀 키 재협상

IBM MQ classes for Java 클라이언트 애플리케이션은 클라이언트 연결의 암호화에 사용되는 비밀 키를 재협상할 때 송신 및 수신한 총 바이트 수를 제어할 수 있습니다.

애플리케이션은 다음 방식으로 이 작업을 수행할 수 있습니다. 애플리케이션에서 이러한 방법을 두 개 이상 사용하는 경우 일반적인 서열 규칙이 적용됩니다.

- MQEnvironment 클래스에 sslResetCount 필드 설정
- 해시 테이블 오브젝트에 환경 특성 MQC.SSL_RESET_COUNT_PROPERTY 설정. 그러면 애플리케이션이 해시 테이블을 MQEnvironment 클래스의 properties 필드에 지정하거나 해시 테이블을 해당 구성자의 MQQueueManager 오브젝트로 전달합니다.

sslResetCount 필드 또는 환경 변수 MQC.SSL_RESET_COUNT_PROPERTY의 값은 비밀 키를 협상하기 전에 IBM MQ classes for Java 클라이언트 코드가 송신하고 수신한 총 바이트 수를 나타냅니다. 송신된 바이트 수는 암호화 전의 바이트 수이며, 수신된 바이트 수는 복호화 후의 바이트 수입니다. 바이트 수에는 IBM MQ classes for Java 클라이언트가 송신하고 수신한 제어 정보도 포함됩니다.

재설정 수가 기본값인 0인 경우 비밀 키는 재협상되지 않습니다. CipherSuite가 지정되지 않으면 재설정 수가 무시됩니다.

*IBM MQ classes for Java*에서 사용자 정의된 SSLSocketFactory 제공

사용자 정의 소켓 Java Secure Socket Extension 팩토리(JSSE)를 사용하는 경우, `MQEnvironment.sslSocketFactory`를 사용자 정의 팩토리 객체로 설정하십시오. 구현 방식에 JSSE 따라 세부 사항이 다릅니다.

서로 다른 JSSE 구현 방식은 서로 다른 기능을 제공할 수 있습니다. 예를 들어, 특정 암호화 하드웨어 모델의 구성을 허용하는 특수화된 JSSE 구현이 있을 수 있습니다. 또한 일부 JSSE 공급자는 프로그램별로 키스토어 및 트러스트스토어를 사용자 정의하거나, 키스토어에서 선택할 신원 인증서를 변경할 수 있도록 허용합니다.

javax.net.ssl.SSLSocketFactory 팩토리 클래스에서 이러한 모든 사용자 정의 JSSE 사항이 추상화됩니다.

사용자 정의된 `SSLSocketFactory` 구현을 생성하는 방법에 대한 자세한 내용은 해당 JSSE 문서에서 확인하십시오. 자세한 내용은 제공자별로 다르지만, 일반적인 단계는 다음과 같습니다.

1. SSLContext에 정적 메소드를 사용하여 SSLContext 오브젝트 작성
2. 적절한 KeyManager 및 TrustManager 구현(고유 팩토리 클래스에서 작성)을 통해 이 SSLContext 초기화
3. SSLContext에서 SSLSocketFactory 작성

SSLSocketFactory 오브젝트가 있으면 MQEnvironment.sslSocketFactory를 사용자 정의된 팩토리 오브젝트로 설정하십시오. 예를 들면, 다음과 같습니다.

```
javax.net.ssl.SSLSocketFactory sf = sslContext.getSocketFactory();
MQEnvironment.sslSocketFactory = sf;
```

IBM MQ classes for Java에서는 이 SSLSocketFactory를 사용하여 IBM MQ 큐 관리자에 연결합니다. 이 특성은 CMQC.SSL_SOCKET_FACTORY_PROPERTY를 사용해서도 설정할 수 있습니다. sslSocketFactory가 널로 설정되면 JVM의 기본 SSLSocketFactory가 사용됩니다. CipherSuite가 설정되지 않으면 특성이 무시됩니다.

사용자 정의 SSLSocketFactories를 사용하는 경우 TCP/IP 연결 공유의 영향을 고려하십시오. 연결 공유가 가능하면 새 소켓에서 제공된 SSLSocketFactory를 요청하지 않습니다. 단 후속 연결 요청 컨텍스트에서는 생성된 소켓이 다를 수 있습니다. 예를 들어 후속 연결에서 다른 클라이언트 인증서를 제공하는 경우 연결 공유가 허용되지 않아야 합니다.

키스토어 JSSE 또는 트러스트스토어에 변경 사항을 적용하는 *IBM MQ classes for Java*

(JSSE) 키스토어 또는 Java Secure Socket Extension 트러스트스토어를 변경하는 경우, 변경 사항이 적용되도록 특정 작업을 수행해야 합니다.

키스토어 JSSE 또는 트러스트스토어의 내용을 변경하거나, 키스토어 또는 트러스트스토어 파일의 위치를 변경하는 경우, 당시 실행 IBM MQ classes for Java 중인 애플리케이션은 변경 사항을 자동으로 반영하지 않습니다. 변경사항을 적용하려면 다음 조치를 수행해야 합니다.

- 애플리케이션이 모든 자체 연결을 닫고 연결 풀의 사용되지 않는 연결을 영구 삭제해야 합니다.
- 제공자가 JSSE 키스토어 및 트러스트스토어의 정보를 캐싱하는 경우, 이 정보를 새로 고쳐야 합니다.

이 조치가 수행된 후에는 애플리케이션이 자체 연결을 재작성할 수 있습니다.

응용 프로그램의 설계 방식과 공급자가 JSSE 제공하는 기능에 따라, 응용 프로그램을 중지하고 다시 시작하지 않고도 이러한 작업을 수행할 수 있습니다. 그러나 애플리케이션을 중지하고 재시작하는 것이 가장 단순한 솔루션일 수 있습니다.

*IBM MQ classes for Java*와 TLS를 사용할 때 오류 핸들링

TLS를 사용하여 큐 관리자에 연결할 때 IBM MQ classes for Java에서 여러 이유 코드가 발행될 수 있습니다.

이러한 이유 코드는 다음 목록에 설명되어 있습니다.

MQRC_SSL_NOT_ALLOWED

sslCipherSuite 특성이 설정되어 있지만 바인딩 연결이 사용되었습니다. 클라이언트 연결에서만 TLS를 지원 합니다.

MQRC_JSSE_ERROR

JSSE 제공자가 IBM MQ에서 핸들링할 수 없는 오류를 보고했습니다. 이는 (JSSE) Java Secure Socket Extension 의 구성 문제로 인해 발생할 수 있으며, 또는 큐 관리자가 제시한 인증서를 검증할 수 없었기 때문 일 수 있습니다. `getCause()` 메서드를 사용하여 `MQException`에서 예외를 검색할 수 있습니다 JSSE .

MQRC_SSL_INITIALIZATION_ERROR

지정된 TLS 구성 옵션으로 MQCONN 또는 MQCONNX 호출이 발행되었지만 TLS 환경의 초기화 중에 오류가 발생했습니다.

MQRC_SSL_PEER_NAME_MISMATCH

sslPeerName 특성에 지정된 DN 패턴이 큐 관리자에서 제공하는 DN과 일치하지 않습니다.

MQRC_SSL_PEER_NAME_ERROR

sslPeerName 특성에 지정된 DN 패턴이 올바르지 않습니다.

MQRC_UNSUPPORTED_CIPHER_SUITE

sslCipherSuite 에 명시된 CipherSuite 는 공급자에 JSSE 의해 인식되지 않았습니다. 제공자가 JSSE 지원하는 CipherSuites 의 전체 목록은 `SSLFactory.getSupportedCipherSuites()` 메서드를 사용하여 프로그램에서 얻을 수 있습니다. IBM MQ와 함께 사용할 수 있는 CipherSuites 목록은 375 페이지의 『IBM MQ classes for Java의 TLS CipherSpecs 및 CipherSuites』에 있습니다.

MQRC_SSL_CERTIFICATE_REVOKED

큐 관리자가 제공하는 인증서가 sslCertStores 특성으로 지정된 CRL에 있습니다. 신뢰할 수 있는 인증서를 사용하도록 큐 관리자를 업데이트하십시오.

MQRC_SSL_CERT_STORE_ERROR

제공된 CertStores에서 큐 관리자가 제공한 인증서를 검색할 수 없습니다. `MQException.getCause()` 메소드는 처음으로 CertStore를 검색할 때 발생한 오류를 리턴합니다. 원인이 되는 예외가 `NoSuchElementException`, `ClassCastException` 또는 `NullPointerException`이면 sslCertStores 특성에 지정된 Collection에 하나 이상의 올바른 CertStore 오브젝트가 있는지 확인하십시오.

*IBM MQ classes for Java*의 TLS CipherSpecs 및 CipherSuites

애플리케이션이 큐 매니저와 IBM MQ classes for Java 연결을 설정할 수 있는지 여부는 MQI 채널의 서버 측에 지정된 CipherSpec 와 클라이언트 측에 지정된 CipherSuite 에 따라 달라집니다.

FIPS 지원

참고:

-  AIX, Linux, 및 Windows에서는 IBM MQ 암호 IBM Crypto for C (ICC) 모듈을 GSKit 9 통해 규정 준수를 제공합니다 FIPS 140-3 . 해당 FIPS 140-3 모듈과 관련된 NIST 인증서는 <https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4755> 에서 확인할 수 있습니다.

-  Linux s390xFIPS 현재.에 대한 지원이 제공되지 않습니다. IBM MQ이 플랫폼에서 지원이 필요한 FIPS 고객은 이전 버전을 계속 사용해야 합니다. 해당 IBM Crypto for C (ICC) 암호화 모듈이 이 플랫폼에서 인증을 받으면, 향후 FixPack 에서 FIPS 140-3 지원이 활성화될 예정입니다.
-  이 안에 포함된 암호화 FIPS 140-3 모듈은 IBM Semeru Runtime 2024년 8월 NIST의 승인을 받았습니다. IBM Semeru Runtime 11+IBM MQ 10.0.0 TLS 를 FIPS 140-3 사용하여 및 IBM MQ classes for Java 의 IBM MQ classes for JMS 및 내에서 Java 8 클라이언트 연결 처리를 지원합니다. <https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4755> 해당 FIPS 140-3 모듈과 관련된 NIST 인증서는 에서 확인할 수 있습니다.
- 컨테이너 환경의 IBM MQ 경우, IBM MQ Operator3.2.0 및 9.4.0.0 이후 버전의 큐 매니저 컨테이너 이미지는 UBI 9를 기반으로 합니다. FIPS 140-3 컨테이너 내에서의 IBM MQ 규정 준수 여부는 현재 검토 중입니다.
-  이 기능이 FIPS 활성화되어 있으면, 컨테이너 IBM MQ 제어 프로세스에서 인증된 OpenSSLFIPS 140-3 모듈을 사용합니다. NIST 인증에 대한 자세한 내용은 다음 링크에서 확인하실 수 있습니다: <https://access.redhat.com/compliance/fips>. IBM MQ 컨테이너 이미지에서 실행되는 큐 관리자는 기본 이미지 플랫폼 버전의 인증 수준과 FIPSIBM MQ 동일합니다.

암호 모음이 준수 FIPS 140-2 (compliant)로 표시된 경우 애플리케이션이 준수(compliance) FIPS 140-2 를 강제하도록 구성되지 않았다면 사용할 수 있습니다. 그러나 애플리케이션에 대해 준수(FIPS 140-2 compliance)가 구성된 경우(이 항목의 "암호 모음 및 FIPS 준수 구성" 섹션 참조), 호환(FIPS 140-2 compatible)으로 표시된 암호 모음(CipherSuites)만 구성할 수 있습니다. 다른 암호 모음(CipherSuites)을 사용하려고 하면 오류가 발생합니다.

참고: 각각은 JRE 여러 암호화 보안 공급자를 가질 수 있으며, 각 공급자는 동일한 CipherSuite 의 구현을 제공할 수 있습니다. 그러나 모든 보안 업체가 FIPS 140-2 또는 FIPS 140-3 인증을 받은 것은 아닙니다. 특정 애플리케이션에 대해 FIPS 140-2 또는 FIPS 140-3 준수 요건이 적용되지 않는 경우, 인증되지 않은 보안 모듈(CipherSuite)이 사용될 가능성이 있습니다. 인증되지 않은 구현의 경우, 해당 보안 모듈(CipherSuite)이 이론적으로는 표준에서 요구하는 최소 보안 수준을 충족하더라도 FIPS 140-2 또는 FIPS 140-3을 준수하지 않을 수 있습니다. IBM MQJMS 애플리케이션에서 FIPS 140-2 적용을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오 380 페이지의 『CipherSuites 구성 및 애플리케이션 IBM MQ classes for Java 내 FIPS 규정 준수 설정』.

CipherSpecs 및 FIPS 140-2CipherSuites, 의 Suite-B 준수 관련 자세한 내용은 CipherSpecs 지정 항목을 참조하십시오. 미국과 관련된 정보도 알아야 Federal Information Processing Standards 할 수 있습니다.

CipherSuites 의 모든 기능을 활용하고 인증된 FIPS 140-2 환경 및/또는 Suite-B 규격에 부합하는 환경에서 운영하려면 적절한 JRE 장비가 필요합니다. IBM CipherSpecs 다음과 그와 동등한 것들에 의해 IBM MQ 지원된 CipherSuitesJava 7 서비스 릴리스 4 픽스 팩 2 이상은 에 IBM 나열된 항목에 CipherSuites 대한 JRE 적절한 지원을 제공합니다.

 IBM MQ 10.0.0TLS 를 사용하여 및 IBM MQ classes for Java 클라이언트 연결을 IBM MQ classes for JMS 처리하는 기능은 8.0.8.30, 11.0.24, 17.0.12, Java21.0.4 및 25.0.2 버전 이상에서 IBM Semeru Runtime 다음 플랫폼에 대한 FIPS 140-3 구현을 지원합니다:

-  AIX
-  Linux for x86-64
-  Linux on Power Systems - Little Endian
-  Windows

참고:

- 32비트 운영 체제는 지원되지 않습니다.
- Semeru 에서는 25가 FIPS 140-3 기본값이며, FIPS 140-2 이 옵션은 사용할 수 없습니다.
- 다른 런타임에 FIPS 140-2 대한 연결을 구성할 수 있습니다.

IBM MQ에서 지원하는 CipherSpec 및 이와 동등한 CipherSuite

다음 표는 지원되는 IBM MQ CipherSpecs 및 해당 CipherSuites 를 나열합니다. 이 표는 또한 통신에 사용되는 프로토콜과, 해당 통신 장치(CipherSuite)가 및 FIPS 140-3 FIPS 140-2 표준을 준수하는지 여부를 나타냅니다.

Deprecated 사용되지 않는 CipherSpec 주제를 검토하여 다음 표에 나열된 CipherSpecs이 IBM MQ에서 더 이상 사용되지 않는지 확인하고, 그렇다면 CipherSpec이 더 이상 사용되지 않는 업데이트가 있는지 확인해야 합니다.

중요사항: 나열된 CipherSuites 는 (JRE)와 함께 IBMJava runtime environment 제공되는 것으로 IBM MQ 지원됩니다. 나열된 CipherSuites 에는 JavaJREOracle 에서 지원하는 항목들이 포함됩니다. 애플리케이션이 를 사용하도록 구성하는 OracleJavaJRE 방법에 대한 자세한 내용은 를 참조하십시오 애플리케이션이 또는 CipherSuiteJavaOracle IBMJava 매핑을 사용하도록 구성하기.

TLS1.3 암호화를 사용하려면 애플리케이션을 실행하는 JRE 시스템이 TLS1.3 를 지원해야 합니다.

암호화 Java 11방식 접두사는 상호 교환이 불가능하므로, 채널 및 클라이언트 정의 모두에 올바른 SSL_ 또는 TLS_ 접두사가 필요합니다.

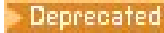
참고: CipherSuites, 를 사용하려면 '제한 없는' 정책 파일을. JRE에 구성해야 합니다. SDK 또는 에서 정책 파일이 설정되는 방법에 대한 자세한 내용은 JRE 사용 중인 버전의 보안 참조: *IBM SDK, Java 기술 에디션* 에서 를 IBM SDK 보안 정책 파일 참조하십시오.

표 59. IBM MQ에서 지원하는 CipherSpec 및 이와 동등한 CipherSuite					
CipherSpec 379 페이지의 『1』	동등한 CipherSuite (IBM JRE)	동등한 CipherSuite (Oracle JRE)	프로토콜	FIPS 140-2 호환 가능	V 10.0.0 FIPS 140-3 호환 되는
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	예	아니오
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	예	예
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	예	예
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	예	예
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	예	예
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	아니오	아니오
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	아니오	아니오
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	예	아니오
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	예	예
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	예	예
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	예	예
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	예	예

표 59. IBM MQ에서 지원하는 CipherSpec 및 이와 동등한 CipherSuite (계속)					
CipherSpec 379 페이지의 『1』	동등한 CipherSuite (IBM JRE)	동등한 CipherSuite (Oracle JRE)	프로토콜	FIPS 140-2 호환 가능	V1000 FIPS 140-3 호환 되는
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	아니오	아니오
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	아니오	아니오
TLS_RSA_WITH_3DES_EDE_CBC_SHA 379 페이지의 『2』 380 페이지의 『5』	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	아니 380 페이지의 『4』	아니오
TLS_RSA_WITH_AES_128_CBC_SHA 380 페이지의 『5』	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	아니 380 페이지의 『4』	아니오
TLS_RSA_WITH_AES_128_CBC_SHA256 380 페이지의 『5』	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	아니 380 페이지의 『4』	아니오
TLS_RSA_WITH_AES_128_GCM_SHA256 380 페이지의 『5』	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	아니 380 페이지의 『4』	아니오
TLS_RSA_WITH_AES_256_CBC_SHA 380 페이지의 『5』	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	아니 380 페이지의 『4』	아니오
TLS_RSA_WITH_AES_256_CBC_SHA256 380 페이지의 『5』	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	아니 380 페이지의 『4』	아니오

표 59. IBM MQ에서 지원하는 CipherSpec 및 이와 동등한 CipherSuite (계속)					
CipherSpec 379 페이지의 『1』	동등한 CipherSuite (IBM JRE)	동등한 CipherSuite (Oracle JRE)	프로토콜	FIPS 140-2 호환 가능	 FIPS 140-3 호환되는
TLS_RSA_WITH_AES_256_GCM_SHA384 380 페이지의 『5』	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	아니오 380 페이지의 『4』	아니오
TLS_RSA_WITH_DES_CBC_SHA 380 페이지의 『5』	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	아니오	아니오
TLS_RSA_WITH_NULL_SHA256 380 페이지의 『5』	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	아니오	아니오
TLS_RSA_WITH_RC4_128_SHA256 380 페이지의 『5』	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	아니오	아니오
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	예	아니오
TLS_AES_128_GCM_SHA256 380 페이지의 『3』	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS 1.3	아니오	예
TLS_AES_256_GCM_SHA384 380 페이지의 『3』	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS 1.3	아니오	예
TLS_CHACHA20_POLY1305_SHA256 380 페이지의 『3』	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	아니오	아니오
TLS_AES_128_CCM_SHA256 380 페이지의 『3』	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS 1.3	아니오	아니오
TLS_AES_128_CCM_8_SHA256 380 페이지의 『3』	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS 1.3	아니오	아니오
임의 380 페이지의 『3』	*ANY	*ANY	다중	아니오	아니오
ANY_TLS13 380 페이지의 『3』	*TLS13	*TLS13	TLS 1.3	아니오	아니오
ANY_TLS12_OR_HIGHER 380 페이지의 『3』	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 이상	아니오	아니오
ANY_TLS13_OR_HIGHER 380 페이지의 『3』	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 이상	아니오	아니오

참고:

1. 이는 채널에 구성된 IBM MQ값으로, CCDT(바이너리 또는 JSON) 내에서도 포함됩니다.
2.  CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA는 더 이상 사용되지 않습니다. 그러나 연결이 오류로 종료되기 전까지 최대 32GB의 AMQ9288 데이터를 전송하는 데 여전히 사용할 수 있습니다. 이 오류를 방지하려면 3중 DES를 사용하지 않거나 이 CipherSpec을 사용할 때 비밀 키 재설정을 사용 가능하게 하십시오.

3. TLS1.3 암호화를 사용하려면 애플리케이션을 실행하는 (JRE) Java runtime environment 이 TLS1.3 를 지원해야 합니다.
4. 부터 IBM MQ 9.4.0, FIPS 모드에서 작동할 때는 RSA 키 교환에 대한 지원을 제거합니다 IBMJava 8JRE .
5. V 10.0.0에서 TLS_RSA_* CipherSpecs IBM MQ 10.0.0가 비활성화되었습니다 Java 25. 자세한 내용은 ‘개발’ [Jakarta MessagingJMS](#) 및 Java ‘애플리케이션’을 참조하십시오.

CipherSuites 구성 및 애플리케이션 IBM MQ classes for Java 내 FIPS 규정 준수 설정

CipherSuite 를 사용하는 애플리케이션은 연결의 를 설정하기 위해 다음 IBM MQ classes for Java 두 가지 방법 중 하나를 사용할 수 있습니다:

- MQEnvironment 클래스의 sslCipherSuite 필드를 CipherSuite 이름으로 설정합니다.
- MQQueueManager 구성자에 전달된 특성 해시 테이블의 CMQC.SSL_CIPHER_SUITE_PROPERTY 특성을 CipherSuite 이름으로 설정합니다.

해당 기능을 사용하는 애플리케이션은 다음 두 가지 방법 IBM MQ classes for Java 중 하나로 규정 준수를 FIPS 140-2 강제할 수 있습니다:

- MQEnvironment 클래스에서 sslFipsRequired 필드를 true로 설정합니다.
- MQQueueManager 생성자에 전달되는 속성 해시 테이블에서 CMQC.SSL_FIPS_REQUIRED_PROPERTY 속성을 true로 설정하십시오.

V 10.0.0의 보안 제공자는 OpenJCEPlusFIPSFIPS 140-3 입니다. 다음 Java 속성을 제공함으로써 구현을 FIPS 140-3 활성화합니다:

```
-Dsemeru.fips=true -Dsemeru.customprofile=OpenJCEPlusFIPS
```

이렇게 하면 기본 보안 공급자가 OpenJCEPlusFIPS 로 설정됩니다. 연결은 해당 FIPS 140-3 구현에서 지원하는 CipherSuite 를 사용해야 합니다([CipherSpecs](#) 및 [이에 IBM MQ 상응하는 CipherSuites](#) 참조).

참고:

- 클라이언트 연결이 사용하는 FIPS 버전을 확인하려면 시스템 속성 Java 을 `com.ibm.fips.mode` 쿼리하십시오. 이것은 또는 140-2 140-3 을 반환합니다.
- 로 마이그레이션할 때 기존 IBM MQ classes for JMS 및 IBM MQ classes for Java IBM MQ 10.0.0 클라이언트의 동작에는 아무런 변화가 없습니다. 그러나 FIPS 클라이언트를 에서 Java 8 로 마이그레이션하는 경우, 보안 FIPS 140-3 공급자를 활성화하기 위해 반드시 지정해야 하는 IBM Semeru Runtime 새로운 Java 속성을 반영하기 위해 클라이언트에 변경 사항을 적용해야 합니다.
- 해당 **SSLFIPSREQUIRED** 속성은 지원되지 IBM Semeru Runtime 않으며, FIPS 인증 연결을 생성하는 데 더 이상 필요하지 않습니다. 클라이언트가 IBM Semeru Runtime 11+ 런타임으로 연결 팩토리에 **set**을 **SSLFIPSREQUIRED** 설정하여 연결하지만 속성이 IBM Semeru RuntimeJava 지정되지 않은 경우, JMSSSL 구성 예외가 발생합니다. 이 예외는 적절한 보안 공급자가 사용 가능한 상태에서 인증된 FIPS 연결이 요청되었음을 나타냅니다. 자세한 정보는 [JMS Exception Messages](#)의 내용을 참조하십시오.
- CipherSuites 에서 Java 11 SSL_ 및 TLS_ 접두사를 상호 교환하여 사용할 수 있는 지원이 제거되었으므로 암호화 방식 정의가 올바르게 지정되어야 합니다.
- FIPS 140-3 IBM Semeru Runtime 32비트 클라이언트를 지원하지 않습니다. 32비트 클라이언트를 업데이트할 수 없는 경우, 오버라이드 파일을 제공하여 하드코딩된 연결 팩토리 속성을 재정의할 수 있습니다.

IBM Java 또는 Oracle Java CipherSuite 맵핑을 사용하도록 애플리케이션 구성

에서 IBM MQ 9.4.0, 암호(Cipher)는 암호화(CipherSpec) 또는 복호화(CipherSuite) 이름으로 정의될 수 있으며 에 의해 올바르게 IBM MQ 처리됩니다.

참고: Removed 이전 버전의 에서 어떤 매핑을 사용할지 제어하던 시스템 `com.ibm.mq.cfg.useIBMCipherMappings` Java 속성은 IBM MQ 더 이상 필요하지 않아 에서 제품에서 IBM MQ 9.4.0 제거되었습니다.

상호 운용성 제한사항

사용 중인 프로토콜에 따라 특정 CipherSuite는 둘 이상의 IBM MQ CipherSpec과 호환 가능할 수 있습니다. 하지만 표 1에 지정된 TLS 버전을 사용하는 CipherSuite/CipherSpec 조합만 지원됩니다. CipherSuite 및 CipherSpec의 지원되지 않는 조합을 사용하려고 시도하면 해당되는 예외로 실패합니다. 이러한 CipherSuite/CipherSpec 조합을 사용한 설치는 지원되는 조합으로 이동되어야 합니다.

다음 표에는 이 제한사항이 적용되는 CipherSuite가 표시되어 있습니다.

표 60. CipherSuite 및 이의 지원 및 지원되지 않는 CipherSpec		
CipherSuite	지원되는 TLS CipherSpec	지원되지 않는 SSL CipherSpec
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA A 381 페이지의 『1』	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

참고:

1. **Deprecated** 이 CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA는 더 이상 사용되지 않습니다. 그러나 AMQ9288 오류로 인해 연결이 종료되기 전까지는 이 CipherSpec을 사용하여 최대 32GB의 데이터를 전송할 수 있습니다. 이 오류를 방지하려면 3중 DES를 사용하지 않거나 이 CipherSpec을 사용할 때 비밀 키 재설정을 사용 가능하게 하십시오.

관련 개념

253 페이지의 『IBM MQ classes for JMS의 TLS CipherSpecs 및 CipherSuites』

애플리케이션이 큐 매니저와 연결을 설정할 수 있는지 IBM MQ classes for JMS 여부는 MQI 채널의 서버 측에 지정된 CipherSpec 와 클라이언트 측에 지정된 CipherSuite 에 따라 달라집니다.

IBM MQ classes for Java 애플리케이션 실행

클라이언트 또는 바인딩 모드를 사용하여 애플리케이션을 작성하는 경우(main() 메소드를 포함하는 클래스) Java 해석기를 사용하여 프로그램을 실행하십시오.

다음 명령을 사용하십시오.

```
java -Djava.library.path= library_path MyClass
```

여기서 *library_path*는 IBM MQ classes for Java 라이브러리의 경로입니다. 추가 정보는 329 페이지의 『IBM MQ classes for Java 라이브러리』의 내용을 참조하십시오.

관련 태스크

[IBM MQ classes for Java 애플리케이션 추적](#)

[IBM MQ 자원 어댑터 추적](#)

IBM MQ classes for Java 환경 종속 동작

IBM MQ classes for Java를 사용하여 여러 다른 버전의 IBM MQ를 실행할 수 있는 애플리케이션을 작성할 수 있습니다. 이 주제 컬렉션에서는 이 여러 다른 버전에 따라 다른 Java 클래스의 동작을 설명합니다.

IBM MQ classes for Java에서는 모든 환경에서 일관된 기능과 동작을 제공하는 코어 클래스를 제공합니다. 이 코어에 포함되지 않은 기능은 애플리케이션이 연결된 큐 관리자의 기능에 따라 달라집니다.

여기에 명시된 내용을 제외하고 나타나는 동작은 큐 관리자에 적절한 [MQI 애플리케이션 참조](#)에 설명되어 있습니다.

IBM MQ classes for Java의 코어 클래스

IBM MQ classes for Java에는 모든 환경에서 사용할 수 있는 코어 클래스 세트가 포함되어 있습니다.

다음 클래스 세트는 코어 클래스로 간주되며 모든 환경에서 사용할 수 있습니다. 단, 383 페이지의 『IBM MQ classes for Java 코어 클래스의 제한사항 및 변형』에 나열된 대로 약간 변화될 수 있습니다.

- MQEnvironment
- MQException
- MQGetMessageOptions

제외:

- MatchOptions
- GroupStatus
- SegmentStatus
- Segmentation

- MQManagedObject

제외:

- inquire()
- set()

- MQMessage

제외:

- groupId
- messageFlags
- messageSequenceNumber
- 오프셋
- originalLength

- MQPoolServices
- MQPoolServicesEvent
- MQPoolServicesEventListener
- MQPoolToken
- MQPutMessageOptions

제외:

- knownDestCount
- unknownDestCount
- invalidDestCount
- recordFields

- MQProcess
- MQQueue
- MQQueueManager

제외:

- begin()
- accessDistributionList()

- MQSimpleConnectionManager
- MQTopic
- MQC

참고:

1. 일부 상수는 코어에 포함되지 않습니다(자세한 내용은 383 페이지의 『IBM MQ classes for Java 코어 클래스의 제한사항 및 변형』 참조). 완전히 이식 가능한 프로그램에서는 사용하지 마십시오.
2. 일부 플랫폼에서는 일부 연결 방식이 지원되지 않습니다. 해당 플랫폼에서는 지원 모드와 연관된 코어 클래스와 옵션만 사용할 수 있습니다.

IBM MQ classes for Java 코어 클래스의 제한사항 및 변형

동등한 MQI 호출은 대개 환경에 따라 차이가 있지만 코어 클래스는 일반적으로 모든 환경에서 일관됩니다. 이는 약간의 제한사항과 변동을 제외하고 AIX, Linux 또는 Windows 큐 관리자가 사용되는 경우와 동일하게 작동합니다.

IBM MQ classes for Java의 MQGMO_* 값 제한사항

특정 MQGMO_* 값은 모든 큐 관리자에서 지원되지 않습니다.

다음 MQGMO_* 값을 사용하면 MQQueue.get()에서 MQException이 발생할 수 있습니다.

```
MQGMO_SYNCPOINT_IF_PERSISTENT
MQGMO_MARK_SKIP_BACKOUT
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_LOCK
MQGMO_UNLOCK
MQGMO_LOGICAL_ORDER
MQGMO_COMPLETE_MESSAGE
MQGMO_ALL_MSGS_AVAILABLE
MQGMO_ALL_SEGMENTS_AVAILABLE
MQGMO_UNMARKED_BROWSE_MSG
MQGMO_MARK_BROWSE_HANDLE
MQGMO_MARK_BROWSE_CO_OP
MQGMO_UNMARK_BROWSE_HANDLE
MQGMO_UNMARK_BROWSE_CO_OP
```

또한 Java에서 사용할 때 MQGMO_SET_SIGNAL은 지원되지 않습니다.

IBM MQ classes for Java의 MQPMRF_* 값 제한사항

이 제한사항은 분배 목록에 메시지를 넣을 때만 사용하며 분배 목록을 지원하는 큐 관리자에서만 지원됩니다. 예를 들어 z/OS 큐 관리자는 분배 목록을 지원하지 않습니다.

IBM MQ classes for Java의 MQPMO_* 값 제한사항

특정 MQPMO_* 값은 모든 큐 관리자에서 지원되지 않습니다.

다음 MQPMO_* 값을 사용하면 MQQueue.put() 또는 MQQueueManager.put()에서 MQException이 발생할 수 있습니다.

```
MQPMO_LOGICAL_ORDER
MQPMO_NEW_CORREL_ID
MQPMO_NEW_MESSAGE_ID
MQPMO_RESOLVE_LOCAL_Q
```

IBM MQ classes for Java의 MQCNO_* 값 제한사항 및 차이점

특정 MQCNO_* 값은 지원되지 않습니다.

- 자동 클라이언트 다시 연결은 IBM MQ classes for Java에서 지원되지 않습니다. 설정하는 MQCNO_RECONNECT_* 값에 상관없이 MQCNO_RECONNECT_DISABLED를 설정한 것처럼 연결이 계속 작동합니다.
- MQCNO_FASTPATH는 MQCNO_FASTPATH를 지원하지 않는 큐 관리자에서 무시됩니다. 클라이언트 연결에서도 무시됩니다.

IBM MQ classes for Java의 MQRO_* 값 제한사항

다음과 같은 보고서 옵션을 설정할 수 있습니다.

MQRO_EXCEPTION_WITH_FULL_DATA
MQRO_EXPIRATION_WITH_FULL_DATA
MQRO_COA_WITH_FULL_DATA
MQRO_COD_WITH_FULL_DATA
MQRO_DISCARD_MSG
MQRO_PASS_DISCARD_AND_EXPIRY

자세한 정보는 [보고서](#)를 참조하십시오.

z/OS의 *IBM MQ classes for Java*와 기타 플랫폼 간의 기타 차이점
IBM MQ for z/OS는 다른 플랫폼의 IBM MQ와 일부 영역에서 다르게 작동합니다.

BackoutCount

메시지가 255번이 넘게 백아웃되었어도 z/OS 큐 관리자가 최대 255인 BackoutCount를 리턴합니다.

기본 동적 큐 접두부

바인딩 연결을 사용하여 z/OS 큐 관리자에 연결하는 경우 기본 동적 큐 접두부는 CSQ.*입니다. 그렇지 않으면 기본 동적 큐 접두부는 AMQ.*입니다.

MQQueueManager 구성자

클라이언트 연결은 z/OS에서 지원되지 않습니다. 클라이언트 옵션과 연결하려고 하면 MQException이 발생하고 MQCC_FAILED 및 MQRC_ENVIRONMENT_ERROR가 표시됩니다.

MQQueueManager 구성자도 MQRC_CHAR_CONVERSION_ERROR(IBM-1047 및 ISO8859-1 코드 페이지 간 변환을 초기화하는 데 실패하는 경우) 또는 MQRC_UCS2_CONVERSION_ERROR(큐 관리자의 코드 페이지와 유니코드 간 변환을 초기화하는 데 실패하는 경우)로 인해 실패할 수 있습니다. 애플리케이션이 이러한 이유 코드 중 하나를 표시하며 실패하는 경우 언어 환경의 자국어(NL) 자원 컴포넌트가 설치되었는지 확인하고 올바른 변환 테이블을 사용할 수 있는지 확인하십시오.

유니코드의 변환 테이블은 z/OS C/C++ 선택적 기능의 일부로 설치됩니다. UCS-2 변환 사용에 대한 자세한 정보는 *z/OS C/C++ 프로그래밍 안내서*, SC09-4765를 참조하십시오.

IBM MQ classes for Java 코어 클래스 외부의 기능

IBM MQ classes for Java에는 모든 큐 관리자가 지원하지 않는 API 확장 기능을 사용하도록 특별히 디자인된 특정 기능이 포함되어 있습니다. 이 주제 컬렉션에서는 해당 확장 기능을 지원하지 않는 큐 관리자를 사용할 때 이 확장 기능이 동작하는 방식을 설명합니다.

MQQueueManager 구성자 옵션의 변형

일부 MQQueueManager 구성자에는 선택적 정수 인수가 포함됩니다. 이 인수의 일부 값은 모든 플랫폼에서 허용되지 않습니다.

MQQueueManager 구성자에 선택적 정수 인수가 포함된 경우, MQI의 MQCNO 옵션 필드에 맵핑되고 정상 경로 연결과 빠른 경로 연결 사이를 전환하는 데 사용됩니다. 사용된 옵션이 MQCNO_STANDARD_BINDING 또는 MQCNO_FASTPATH_BINDING뿐이면 이 확장된 양식의 구성자가 모든 환경에서 허용됩니다. 다른 옵션을 사용하면 구성자가 MQRC_OPTIONS_ERROR로 인해 실패하게 됩니다. 빠른 경로 옵션 CMQC.MQCNO_FASTPATH_BINDING은 지원되는 큐 관리자에 바인딩 연결하는 데만 사용됩니다. 다른 환경에서는 무시됩니다.

MQQueueManager.begin() 메소드의 제한사항

이 방법은 바인딩 모드에서 실행 중인 AIX, Linux, and Windows 시스템의 큐 IBM MQ 관리자에 대해서만 사용할 수 있습니다. 그렇지 않으면 MQRC_ENVIRONMENT_ERROR로 인해 실패합니다.

자세한 내용은 368 페이지의 [『IBM MQ classes for Java를 사용하여 JTA/JDBC 통합』](#)의 내용을 참조하십시오.

MQGetMessageOptions 필드의 변형

일부 큐 관리자에서는 버전 2 MQGMO 구조를 지원하지 않으므로, 일부 필드를 기본값으로 설정해야 합니다.

버전 2 MQGMO 구조를 지원하지 않는 큐 관리자를 사용할 때 다음 필드를 기본값으로 설정된 상태로 두십시오.

GroupStatus
SegmentStatus

Segmentation

또한 MatchOptions 필드는 MQMO_MATCH_MSG_ID와 MQMO_MATCH_CORREL_ID만 지원합니다. 지원되지 않는 값을 이러한 필드에 넣으면 후속 MQDestination.get()이 MQRC_GMO_ERROR로 인해 실패합니다. 큐 관리자가 버전 2 MQGMO 구조를 지원하지 않으면 이러한 필드는 MQDestination.get()에 성공한 후에도 업데이트되지 않습니다.

*IBM MQ classes for Java*의 분배 리스트 제한사항

일부 큐 관리자에서는 MQDistributionList를 열 수 없습니다.

다음 클래스를 사용하여 분배 리스트를 작성합니다.

```
MQDistributionList
MQDistributionListItem
MQMessageTracker
```

모든 환경에서 MQDistributionLists 및 MQDistributionListItems를 작성하고 채울 수 있지만, 일부 큐 관리자에서는 MQDistributionList를 열 수 없습니다. 특히 z/OS 큐 관리자는 분배 목록을 지원하지 않습니다. 이러한 큐 관리자를 사용할 때 MQDistributionList를 열려고 하면 MQRC_OD_ERROR가 발생합니다.

MQPutMessageOptions 필드의 변형

큐 관리자가 분배 목록을 지원하지 않으면 특정 MQPMO 필드는 다르게 처리됩니다.

MQPMO의 네 필드는 MQPutMessageOptions 클래스의 다음 멤버 변수로 렌더링됩니다.

```
knownDestCount
unknownDestCount
invalidDestCount
recordFields
```

이러한 필드는 주로 분배 목록과 사용됩니다. 그러나 분배 목록을 지원하는 큐 관리자가 단일 큐에 MQPUT을 수행한 후 DestCount 필드를 채웁니다. 예를 들어 큐가 로컬 큐로 해석되면 knownDestCount가 1로 설정되고 다른 두 count 필드는 0으로 설정됩니다.

큐 관리자가 분배 목록을 지원하지 않으면 이러한 값은 다음과 같이 시뮬레이션됩니다.

- put()에 성공하면 unknownDestCount가 1로 설정되고 나머지는 0으로 설정됩니다.
- put()에 실패하면 invalidDestCount가 1로 설정되고 나머지는 0으로 설정됩니다.

recordFields 변수는 분배 목록과 함께 사용됩니다. 환경과 상관없이 언제든지 값을 recordFields에 기록할 수 있습니다. MQPutMessageOptions 오브젝트가 MQDistributionList.put()이 아니라 후속 MQDestination.put() 또는 MQQueueManager.put()에서 사용되는 경우에는 무시됩니다.

*IBM MQ classes for Java*에서 MQMD 필드의 제한사항

세그먼트화를 지원하지 않는 큐 관리자를 사용할 때 메시지 세그먼트화와 관련된 특정 MQMD 필드는 기본값으로 남겨 두어야 합니다.

다음 MQMD 필드는 주로 메시지 세그먼트화와 관련됩니다.

```
GroupId
MsgSeqNumber
오프셋
MsgFlags
OriginalLength
```

애플리케이션이 MQMD 필드를 기본값 이외의 값으로 설정한 다음 put() 또는 get()을 지원하지 않는 큐 관리자에서 해당 함수를 수행하면 put() 또는 get()에서 MQException가 발생하고 MQRC_MD_ERROR가 표시됩니다. 이러한 큐 관리자에서 put() 또는 get()을 성공적으로 수행하려면 MQMD 필드를 기본값으로 설정하십시오. 메시지 그룹화와 세그먼트화를 지원하지 않는 큐 관리자에 대해 실행하는 Java 애플리케이션에 그룹화되었거나 세그먼트화된 메시지를 송신하지 마십시오.

Java 애플리케이션이 이러한 필드를 지원하지 않는 큐 관리자에서 메시지 get()을 시도하고 검색된 실제 메시지가 세그먼트화된 메시지 그룹의 일부이면(즉, MQMD 필드의 기본이 아닌 값 포함) 오류 없이 검색됩니다. 그러나

MQMessage의 MQMD 필드가 업데이트되지 않고 MQMessage 형식 특성이 MQFMT_MD_EXTENSION으로 설정되며 새 필드의 값을 포함하는 MQMDE 구조가 true 메시지 데이터의 접두부로 지정됩니다.

CICS Transaction Server에서 IBM MQ classes for Java에 대한 제한사항

CICS Transaction Server for z/OS 환경에서는 기본 (첫 번째) 스레드만 CICS 또는 IBM MQ 호출을 발행할 수 있습니다.

IBM MQ JMS 클래스는 CICS Java 애플리케이션에서 사용하도록 지원되지 않습니다.

그러므로 이 환경에서 스레드 간에 MQQueueManager 또는 MQQueue 오브젝트를 공유하거나 하위 스레드에서 새 MQQueueManager를 작성할 수 없습니다.

z/OS 384 페이지의 『z/OS의 IBM MQ classes for Java와 기타 플랫폼 간의 기타 차이점』에서는 z/OS 큐 관리자에 대해 실행할 때 IBM MQ classes for Java에 적용되는 일부 제한사항 및 변형을 식별합니다. 또한 CICS에서 실행할 때 MQQueueManager의 트랜잭션 제어 메소드는 지원되지 않습니다. MQQueueManager.commit() 또는 MQQueueManager.backout()을 발행하는 대신에, 애플리케이션은 JCICS 태스크 동기화 메소드, Task.commit() 및 Task.rollback()을 사용합니다. 태스크 클래스는 com.ibm.cics.server 패키지에 JCICS을(를) 통해 제공됩니다.

리소스 IBM MQ 어댑터 사용

리소스 IBM MQ 어댑터는 애플리케이션 서버에서 실행 중인 애플리케이션이 리소스에 IBM MQ 접근할 수 있도록 합니다. 리소스 어댑터는 인바운드 및 아웃바운드 통신을 지원합니다.

리소스 어댑터가 포함하는 내용

새로운 애플리케이션 개발을 위해 가 지원됩니다 IBM MQ 9.3.0Jakarta Messaging 3.0 . IBM MQ 9.3.0 기존 애플리케이션에 대한 JMS 2.0 지원을 계속 제공합니다. 와 를 Java EE 지원하는 JMS 2.0리소스 어댑터 외에도, IBM MQ 9.3.0 및 이후 버전에서는 를 지원하는 리소스 어댑터를 Jakarta Messaging 제공합니다.

JM 3.0 IBM MQ 리소스 어댑터 Jakarta Messaging

이 표준은 환경에서 Jakarta EE 실행 중인 애플리케이션을 또는 IBM MQ 과 같은 Jakarta Connectors Architecture 기업 정보 시스템(EIS)에 Db2 연결하는 표준화된 방법을 제공합니다. 리소스 IBM MQ 어댑터는 Jakarta MessagingJakarta Connectors 2.0.0 인터페이스를 구현하며 를 포함합니다 IBM MQ classes for Jakarta Messaging. 애플리케이션 서버에서 실행 중인 애플리케이션 및 메시지 주도 빈(MDB)이 큐 IBM MQ Jakarta Messaging 매니저의 리소스에 접근할 수 있도록 합니다. 자원 어댑터는 포인트-투-포인트 도메인과 발행/구독 도메인을 모두 지원합니다.

JMS 2.0 IBM MQ 리소스 어댑터 JMS 2.0

Java Platform, Enterprise Edition Connector Architecture(JCA)에서는 Java EE 환경에서 실행 중인 애플리케이션을 IBM MQ 또는 Db2 등의 EIS(Enterprise Information System)에 연결하는 표준 방식을 제공합니다. 리소스 IBM MQ 어댑터는 JMS 2.0JCA 1.7 인터페이스를 구현하며 를 포함합니다 IBM MQ classes for JMS. 애플리케이션 서버에서 실행 중인 JMS 애플리케이션 및 메시지 구동 Bean(MDB)이 IBM MQ 큐 관리자의 자원에 액세스할 수 있게 합니다. 자원 어댑터는 포인트-투-포인트 도메인과 발행/구독 도메인을 모두 지원합니다.

IBM MQ 자원 어댑터는 애플리케이션과 큐 관리자 사이의 두 통신 유형을 지원합니다.

아웃바운드 통신

애플리케이션에서 큐 관리자와 연결을 시작한 다음 동기적인 방식으로 JMS 메시지를 JMS 목적지에 송신하고 JMS 목적지에서 JMS 메시지를 수신합니다.

인바운드 통신

JMS 목적지에 도착한 JMS 메시지는 메시지를 비동기식으로 처리하는 MDB에 전달됩니다.

자원 어댑터에는 IBM MQ classes for Java도 포함되어 있습니다. 리소스 어댑터가 배포된 애플리케이션 서버에서 실행 중인 애플리케이션은 해당 클래스를 자동으로 사용할 수 있으며, 해당 애플리케이션 서버에서 실행 중인 애플리케이션은 큐 IBM MQ 매니저의 리소스에 접근할 때 API를 IBM MQ classes for Java 사용할 수 있습니다.

제한 사항이 적용된 상태에서 환경 IBM MQ classes for JavaJava EE 내에서의 사용이 지원됩니다. 이러한 제한 사항에 관한 정보는 322 페이지의 『애플리케이션 IBM MQ classes for Java 실행 Java EE』의 내용을 참조하십시오.

사용할 자원 어댑터 버전

사용하는 리소스 어댑터 버전은 해당 어댑터를 지원하는 애플리케이션 서버에 배포하는지 Jakarta EEJava EE 여부에 따라 달라집니다

JM 3.0 Jakarta EE

부터 IBM MQ 9.3.0, Jakarta Messaging 3.0 가 지원됩니다. 의 IBM MQ 리소스 어댑터는 를 지원하는 애플리케이션 Jakarta Messaging 서버 내에 Jakarta EE 배포되어야 합니다.

JMS 2.0 Java EE 7

IBM MQ 8.0 이상 자원 어댑터는 JCA v1.7을 지원하고 JMS 2.0 지원을 제공합니다. 이 자원 어댑터는 Java EE 7 이상 애플리케이션 서버에 배치해야 합니다(388 페이지의 『지원의 IBM MQ 자원 어댑터 명령문』 참조).

해당 Java Platform, Enterprise Edition 7 사양을 준수하는 것으로 인증된 모든 애플리케이션 서버에 버전 또는 이후 버전의 IBM MQ 8.0 리소스 어댑터를 설치할 수 있습니다. 또는 이후 버전의 IBM MQ 8.0 리소스 어댑터를 사용하여 애플리케이션은 BINDINGS 또는 CLIENT 전송을 통해 큐 관리자에 연결할 수 있습니다.

중요사항: IBM MQ 8.0 이상 자원 어댑터는 JMS 2.0을 지원하는 애플리케이션 서버에만 배치할 수 있습니다.

WebSphere Application Server와 자원 어댑터 사용

리소스 IBM MQ 어댑터는 또는 이후 WebSphere Application Server 9.0 버전 내에 사전 설치되어 있습니다. 그러므로 새 자원 어댑터를 설치할 필요가 없습니다.

JM 3.0 Jakarta EEWebSphere Application Server 현재는 지원하지 않습니다. 388 페이지의 『지원의 IBM MQ 자원 어댑터 명령문』의 내용을 참조하십시오.

참고: IBM MQ 9.0 이상 자원 어댑터는 서비스 중인 모든 IBM MQ 큐 관리자에 CLIENT 또는 BINDINGS 전송 모드로 연결될 수 있습니다.

WebSphere Liberty와 자원 어댑터 사용

에 IBM MQ 연결하려면 리소스 WebSphere LibertyIBM MQ 어댑터를 사용해야 합니다. 에는 리소스 IBM MQ 어댑터가 포함되어 있지 Liberty 않으므로, 별도로 에서 획득해야 Fix Central 합니다.

사용하는 리소스 어댑터 버전은 해당 어댑터를 또는 Jakarta EE 을 지원하는 Liberty 버전의 에 Java EE배포하는 지에 따라 달라집니다.

자원 어댑터를 다운로드하여 설치하는 방법에 대한 자세한 정보는 398 페이지의 『리소스 IBM MQ 어댑터 설치』의 내용을 참조하십시오.

관련 개념

409 페이지의 『인바운드 통신용 자원 어댑터 구성』

인바운드 통신을 구성하려면 하나 이상의 ActivationSpec 오브젝트에 대한 특성을 정의하십시오.

425 페이지의 『아웃바운드 통신용 자원 어댑터 구성』

아웃바운드 통신을 구성하려면 ConnectionFactory 오브젝트와 관리 대상 목적지 오브젝트의 특성을 정의하십시오.

83 페이지의 『사용 IBM MQ classes for JMS/Jakarta Messaging』

IBM MQ classes for JMS 그리고 IBM MQ classes for Jakarta Messaging 는 와 함께 제공되는 IBM MQ메시징 Java 공급자입니다. 이 메시징 공급자들은 및 Jakarta Messaging JMS 사양에서 정의된 인터페이스를 구현하는 것 외에도, 메시징 Java API에 두 가지 확장 기능을 추가합니다.

320 페이지의 『IBM MQ classes for Java 사용』

사용 IBM MQJava 환경에서 사용하십시오. IBM MQ classes for Java를 사용하면 Java 애플리케이션을 IBM MQ 클라이언트로 IBM MQ에 연결하거나 IBM MQ 큐 관리자에 직접 연결할 수 있습니다.

관련 참조

최신 자원 어댑터 유지보수 레벨을 사용하도록 애플리케이션 서버 구성

IBM MQ 자원 어댑터의 문제점 판별

WebSphere Application Server 토포픽

IBM MQ 자원 어댑터 유지보수

[Liberty에 애플리케이션을 JMS 배포하여 메시징 IBM MQ 공급자를 사용하기](#)

지원의 IBM MQ 자원 어댑터 명령문

응용 프로그램과 큐 관리자 간 통신에 사용해야 하는 리소스 IBM MQ 어댑터는 API를 Jakarta Messaging 3.0 사용하는지 아니면 API를 JMS 2.0 사용하는지에 따라 달라집니다.

JMS 2.0 IBM MQ 8.0 또는 이후 버전에는 해당 JMS 2.0 사양을 구현하는 리소스 어댑터가 포함됩니다. 이는 Java Platform, Enterprise Edition 7(Java EE 7) 호환 애플리케이션 서버에만 배치될 수 있으므로 JMS 2.0을 지원합니다. [Oracle 웹사이트](#)에서 인증된 애플리케이션 서버 Java Platform, Enterprise Edition 목록을 관리하고 있습니다.

JM 3.0 새로운 애플리케이션 개발을 위해 가 지원됩니다 IBM MQ 9.3.0 Jakarta Messaging 3.0 . IBM MQ 9.3.0 기존 애플리케이션에 대한 JMS 2.0 지원을 계속 제공합니다. 와 를 Java EE 지원하는 JMS 2.0 리소스 어댑터 외에도, IBM MQ 9.3.0 및 이후 버전에서는 를 지원하는 리소스 어댑터를 Jakarta Messaging 제공합니다. 동일한 애플리케이션에서 API와 Jakarta Messaging 3.0 API를 JMS 2.0 동시에 사용하는 것은 지원되지 않습니다. 자세한 정보는 [JMS용 IBM MQ 클래스 사용](#)을 참조하십시오.

WebSphere Liberty 내에 배치

WebSphere Liberty 8.5.5 Fix Pack 6 이상 및 WebSphere Application Server Liberty 9.0 이상은 Java EE 7 인증 애플리케이션 서버이므로 IBM MQ 9.0 자원 어댑터를 이에 배치할 수 있습니다.

리소스 IBM MQ 어댑터를 와 Jakarta Messaging 함께 사용하려면, 를 Liberty 지원하는 Liberty 버전의 를 Jakarta EE 사용해야 합니다.

WebSphere Liberty 리소스 어댑터 작업에 사용할 수 있는 다음과 같은 기능을 제공합니다:

- **JM 3.0** messaging-3.0 는 리소스 Jakarta Messaging 3.0 어댑터와 함께 작업할 수 있도록 하는 기능을 제공합니다.
- wmqJmsClient-2.0 기능 - JMS 2.0 자원 어댑터와 함께 사용할 수 있습니다.
- wmqJmsClient-1.1 기능 - JMS 1.1 자원 어댑터와 함께 사용할 수 있습니다.

중요사항:

- **JM 3.0** 해당 IBM MQ 리소스 어댑터는 를 지원하는 Jakarta Messaging Liberty 버전의 에 Jakarta EE 배포되어야 합니다. 이 리소스 어댑터는 이전 Java EE 사양을 지원하는 버전의 Liberty 와 함께 사용할 수 없습니다 Jakarta EE.
- **JMS 2.0** wmqJmsClient-2.0 기능을 지원하는 또는 이후 버전의 JMS 2.0 IBM MQ 8.0 리소스 어댑터는 반드시 해당 기능과 함께 배포되어야 합니다.

WebSphere Application Server 내에 배치

WebSphere Application Server 9.0은 IBM MQ 9.0 자원 어댑터가 이미 설치되어 제공됩니다. 그러므로 새 자원 어댑터를 설치할 필요가 없습니다. 설치된 리소스 어댑터는 CLIENT 또는 BINDINGS 전송 모드로 지원되는 버전의 IBM MQ.에서 실행 중인 모든 큐 관리자에 연결할 수 있습니다. 자세한 정보는 [389 페이지의 『IBM MQ 8.0 이상 큐 관리자에 대한 연결성』](#)의 내용을 참조하십시오.

중요사항:

- 리소스 IBM MQ 9.0 어댑터는 인증되지 Java EE 7 애플리케이션 버전이기 때문에 이전 WebSphere Application Server 버전에는 배포할 IBM MQ 9.0 수 없습니다.
- **JM 3.0** Jakarta EE WebSphere Application Server 현재는 지원하지 않습니다.

[WebSphere MQ](#) 과 WebSphere Application Server 함께 제공되는 리소스 어댑터 버전에 대한 자세한 내용은 기술 노트 'WebSphere Application Server 과 함께 제공되는 리소스 어댑터(RA) 버전은 무엇인가요?'를 참조하십시오

다른 애플리케이션 서버와 함께 자원 어댑터 사용

기타 모든 Java EE 7 또는 Jakarta EE 호환 애플리케이션 서버의 경우, 리소스 IBM MQ 어댑터 설치 검증 테스트 (IVT)의 성공적인 완료 후 발생하는 문제는 제품 추적 및 기타 IBM MQ IBM MQ 진단 정보 조사를 위해 해당 IBM 부서에 보고할 수 있습니다. 리소스 어댑터 IBM MQ IVT를 성공적으로 실행할 수 없는 경우, 발생하는 모든 문제는 애플리케이션 서버별 잘못된 배포 또는 잘못된 리소스 정의로 인해 발생할 가능성이 높으며, 해당 애플리케이션 서버의 문서 및 지원 조직을 활용하여 문제를 조사해야 합니다.

Java 런타임

애플리케이션 서버를 실행하는 데 사용되는 (JRE) Java runtime environment 는 또는 이후 IBM MQ 9.0 버전의 클라이언트에서 지원되는 버전이어야 합니다. 자세한 정보는 시스템 요구 사항 IBM MQ의 내용을 참조하십시오. (보려는 운영 체제 또는 컴포넌트 보고서의 버전을 선택한 후 지원되는 소프트웨어 탭 아래에 나열된 Java 링크를 따르십시오.)

IBM MQ 8.0 이상 큐 관리자에 대한 연결성

인증된 애플리케이션 Java EE 7 서버에 배포된 리소스 어댑터를 사용하여 버전 이상의 큐 IBM MQ 8.0 관리자에 연결할 경우 모든 기능 JMS 2.0 범위를 사용할 수 있습니다. JMS 2.0 기능을 사용하려면, IBM MQ 메시징 제공자 정상 모드를 사용하여 자원 어댑터를 큐 관리자에 연결해야 합니다. 자세한 정보는 [JMS PROVIDERVERSION](#) 특성 구성을 참조하십시오.

JM 3.0 인증된 애플리케이션 Jakarta EE 서버에 배포된 리소스 어댑터를 사용하여 버전 이상의 큐 IBM MQ 9.3 관리자에 연결할 경우 모든 기능 Jakarta Messaging 3.0 범위를 사용할 수 있습니다.

MQ 확장

JMS 2.0 스펙은 특정 동작이 작동하는 방식에 대한 변경사항을 소개합니다. 이 스펙은 IBM MQ 8.0 이상에서 구현되므로, IBM MQ 8.0 이상과 제품의 이전 버전 간에는 동작 변경사항이 있습니다. IBM MQ 8.0 이상에서, IBM MQ classes for JMS는 Java 시스템 특성 com.ibm.mq.jms.SupportMQExtensions에 대한 지원을 포함하며, 이 특성이 TRUE로 설정되면, 이러한 버전의 IBM MQ가 이러한 작동을 IBM WebSphere MQ 7.5 또는 이전 버전으로 되돌립니다. 특성의 기본값은 FALSE입니다.

또는 이후 버전의 IBM MQ 9.0 리소스 어댑터에는 시스템 com.ibm.mq.jms.SupportMQExtensionsJava 속성과 동일한 효과 및 기본값을 가진 이라는 supportMQExtensions 리소스 어댑터 속성도 포함됩니다. 이 자원 어댑터 특성은 기본적으로 ra.xml에서 false로 설정됩니다.

자원 어댑터 특성 및 Java 시스템 특성이 둘 다 설정된 경우, 시스템 특성이 우선순위를 가집니다.

WebSphere Application Server 9.0 내에 이미 배치된 자원 어댑터 내에서 이 특성이 자동으로 마이그레이션을 지원하기 위해 TRUE로 설정됨에 유의하십시오.

자세한 정보는 [302 페이지의 『SupportMQExtensions 특성』](#)의 내용을 참조하십시오.

일반 문제

세션 인터리빙이 지원되지 않음

일부 애플리케이션 서버는 세션 인터리빙이라는 기능을 제공하며, 여기서는 한 번에 하나에만 나열되더라도 동일한 JMS 세션이 다중 트랜잭션에서 사용될 수 있습니다. IBM MQ 자원 어댑터는 다음 문제를 유발할 수 있으므로 이 기능을 지원하지 않습니다.

An attempt to put a message to a MQ queue fails with reason code 2072 (MQRC_SYNCPOINT_NOT_AVAILABLE).

xa_close()에 대한 호출이 이유 코드 -3(XAER_PROTO)과 함께 실패했으며 프로브 ID가 AT040010인 FDC가 애플리케이션 서버로부터 액세스 중인 IBM MQ 큐 관리자에 생성됩니다. 이 기능을 사용 안함으로 설정하는 방법에 대한 정보는 애플리케이션 서버 문서를 참조하십시오.

XA 트랜잭션 복구를 위해 XA 자원이 복구되는 방법에 대한 Java Transaction API(JTA) 스펙

JTA 스펙의 섹션 3.4.8은 XA 트랜잭션 복구를 수행하기 위해 재작성된 XA 자원에 의해 특정 메커니즘을 정의하지 않습니다. 예를 들어 XA 트랜잭션에 포함된 XA 자원이 복구되는 방법은 각 개별 트랜잭션 관리자(따

라서 애플리케이션 서버)에게 달려있습니다. 일부 애플리케이션 서버의 경우 IBM MQ 9.0 자원 어댑터가 XA 트랜잭션 복구를 사용하기 위해 사용된 애플리케이션 서버별 메커니즘을 구현하지 않을 수 있습니다.

ManagedConnectionFactory에서의 연결 일치

애플리케이션 서버는 IBM MQ 자원 어댑터가 제공하는 ManagedConnectionFactory 인스턴스에서 matchManagedConnections 메소드를 호출할 수 있습니다. ManagedConnectionFactory은 애플리케이션 서버에 의해 메소드에 전달된 **javax.security.auth.Subject** 및

javax.resource.spi.ConnectionRequestInfo 인수 모두와 일치하는 항목을 메소드가 찾은 경우에만 리턴됩니다.

리소스 IBM MQ 어댑터의 한계 및 IBM MQ 리소스 어댑터에 대한 Jakarta Messaging

리소스 어댑터와 IBM MQ 리소스 어댑터는 모든 IBM MQ Jakarta Messaging 플랫폼에서 지원됩니다. 그러나 리소스 어댑터를 사용할 경우, 일부 기능이 IBM MQ 사용 불가능하거나 제한됩니다.

리소스 어댑터 및 IBM MQ 리소스 어댑터는 다음과 같은 Jakarta Messaging 제한 사항이 있습니다:

- ▶ **JMS 2.0** 리소스 IBM MQ 어댑터 (wmqjms.ra.rar)는 Java Platform, Enterprise Edition (Java EE) 기능을 JMS 2.0 제공하는 () 리소스 어댑터입니다. 이 리소스 어댑터는 이상 인증 애플리케이션 Java EE 7 서버에 설치해야 합니다. 클라이언트 또는 바인딩 전송 모드로 서비스 중인 모든 큐 관리자에 연결할 수 있습니다.
- ▶ **JM 3.0** (wmq.jakarta.jmsra.rar)용 리소스 IBM MQ Jakarta Messaging 어댑터는 (Jakarta EE) 기능을 Jakarta Messaging 제공하는 (Jakarta Platform, Enterprise Edition) 리소스 어댑터입니다. 이 리소스 어댑터는 이상 인증 애플리케이션 Jakarta EE 9 서버에 설치해야 합니다. 클라이언트 또는 바인딩 전송 모드로 서비스 중인 모든 큐 관리자에 연결할 수 있습니다.
- WebSphere Liberty 애플리케이션 서버에서 실행할 때 안정화된 IBM MQ classes for Java는 지원되지 않습니다. 다른 애플리케이션 서버 내에서는 사용을 IBM MQ classes for Java 권장하지 않습니다. 에 대한 고려 사항에 IBM MQ classes for Java 대한 자세한 Java EE 내용은 을 참조하십시오 [322 페이지의 『애플리케이션 IBM MQ classes for Java 실행 Java EE』](#).
- ▶ **JMS 2.0** 애플리케이션 WebSphere Liberty 서버에서 리소스 IBM MQ 어댑터를 사용할 때 여러 가지 제한 사항이 있습니다. 자세한 정보는 [392 페이지의 『메시징 WebSphere Liberty IBM MQ 공급자와 함께 리소스 IBM MQ 어댑터 사용』](#) 및 [394 페이지의 『리소스 IBM MQ 어댑터를 일반 리소스 어댑터로 사용하기』](#)의 내용을 참조하십시오.
- ▶ **JM 3.0** 애플리케이션 WebSphere Liberty 서버에서 Jakarta Messaging 리소스 IBM MQ 어댑터를 사용할 때 여러 가지 제한 사항이 있습니다. 자세한 정보는 [393 페이지의 『메시징 WebSphere Liberty IBM MQ 공급자와 함께 Jakarta Messaging 리소스 IBM MQ 어댑터 사용』](#) 및 [396 페이지의 『리소스 IBM MQ Jakarta Messaging 어댑터로 사용하기 위한 리소스 어댑터』](#)의 내용을 참조하십시오.
- 리소스 IBM MQ 어댑터는 C++ 이외의 언어로 작성된 채널 종료 프로그램을 Java 지원하지 않습니다.
- 애플리케이션 서버가 실행 중인 동안 sslFipsRequired 특성의 값은 모든 JCA 자원의 경우 true여야 하고 모든 JCA 자원의 경우에는 false여야 합니다. 이 요구사항은 JCA 자원을 동시에 사용하지 않는 경우에도 해당합니다. sslFipsRequired 특성의 값이 각기 다른 JCA 자원마다 서로 다른 경우 TLS 연결을 사용 중이지 않아도 IBM MQ에서 이유 코드 MQRC_UNSUPPORTED_CIPHER_SUITE를 발행합니다.
- 애플리케이션 서버의 키 저장소를 두 개 이상 지정할 수 없습니다. 두 개 이상의 큐 관리자에 연결하는 경우 모든 연결에서 동일한 키 저장소를 사용해야 합니다. 이 제한사항은 WebSphere Application Server에 적용되지 않습니다.
- 두 개 이상의 적당한 클라이언트 연결 채널 정의가 포함된 클라이언트 채널 정의 테이블(CCDT)을 사용하는 경우, 장애가 발생하면 자원 어댑터에서 다른 채널 정의를 선택할 수 있으므로, CCDT의 다른 큐 관리자로 인해 트랜잭션 복구 문제가 초래될 수 있습니다. 자원 어댑터는 이러한 구성을 사용하지 못하게 만드는 조치를 취하지 않으므로, 사용자가 트랜잭션 복구 문제를 초래할 수 있는 구성을 방지해야 합니다.
- 아웃바운드 연결의 연결 재시도 기능은 또는 Jakarta EE Java EE 컨테이너(EJB/Servlet)에서 실행 시 지원되지 않습니다. 어댑터가 JEE 컨테이너 컨텍스트에서 사용될 경우, 트랜잭션 구성 여부나 트랜잭션이 JMS 없는 사용과 관계없이 아웃바운드에 대한 연결 재시도는 전혀 지원되지 않습니다.
- Java EE Connector Architecture 버전 1.7 스펙의 9.1.9 섹션에 정의한 대로 JMS 연결은 다시 인증할 수 없습니다. 리소스 IBM MQ 어댑터 내부의 파일과 에 IBM MQ 대한 ra.xml 리소스 어댑터는 라는 속성을 Jakarta Messaging false **reauthentication-support** 값으로 설정해야 합니다. 애플리케이션 서버가 또는

Jakarta Messaging JMS 연결을 재인증하려는 시도는 리소스 IBM MQ 어댑터가 메시지 코드 MQJCA1028 를 가진 javax.resource.spi.SecurityException 를 발생시키는 결과를 초래합니다.

관련 태스크

[MQI 클라이언트에서 실행 시 인증된 CipherSpecs 만 FIPS 사용하도록 지정](#)

관련 참조

[Federal Information Processing Standards \(FIPS\)에 대해 AIX, Linux, and Windows](#)

▶ JMS 2.0 WebSphere Application Server 그리고 IBM MQ 리소스 어댑터

WebSphere Application Server 9 리소스 IBM MQ 9.0 어댑터를 임베드합니다. 메시징 IBM MQ 공급자와 메시지를 JMS 수행하는 애플리케이션에서 사용됩니다.

애플리케이션 서버에 내장된 리소스 IBM MQ 어댑터는 픽스 WebSphere Application Server 팩을 통해 업데이트됩니다. 와 WebSphere Application Server 함께 제공되는 리소스 IBM MQ 어댑터 버전에 대한 정보는 기술 노트 " [WebSphere Application Server 와 함께 제공되는 WebSphere MQ 리소스 어댑터\(RA\) 버전은 무엇인가요?](#) "를 참조하십시오.

특정 버전의 리소스 IBM MQ 어댑터를 에 WebSphere Application Server 9 설치하고, 메시징 IBM MQ 공급자가 내장된 리소스 어댑터 대신 이를 사용하도록 구성할 수 있습니다. 이 작업을 수행하는 방법에 대한 자세한 내용은 제품 WebSphere Application Server 설명서의 " [IBM MQ 리소스 어댑터의 특정 유지 관리 수준 설치](#) " 항목을 참조하십시오.

WebSphere Application Server 9 와 Java EE 8Java EE 7 사양을 지원합니다. (또는 이후 Jakarta EE 9) 사양을 지원하지 않습니다. 이는 해당 애플리케이션 서버 버전에 Jakarta Messaging 리소스 IBM MQ 어댑터를 설치할 수 없음을 의미합니다.

WebSphere Application Server 8.5.5 사양을 Java EE 6 지원합니다. 해당 IBM MQ 9.0 리소스 어댑터는 및 을 Java EE 7Java EE 8 지원하므로, 이 버전의 애플리케이션 서버에는 설치할 수 없습니다.

중요사항: 에 설치되는 엔터프라이즈 애플리케이션 WebSphere Application Server 내부에는 IBM MQ classes for JMS 또는 IBM MQ classes for Java 를 포함하지 마십시오. 리소스 IBM MQ 어댑터가 내장된 에는 와 WebSphere Application Server 도 포함되어 있으므로, IBM MQ classes for Java 애플리케이션 서버에 IBM MQ classes for JMS 해당 클래스의 복수 사본이 로드될 수 있습니다. 이로 인해 애플리케이션 서버에서 예기치 않은 동작(예: 애플리케이션 서버에서 예기치 않은 종료(ClassCastExceptions) 발생)이 발생할 수 있습니다. 리소스 어댑터에 포함된 클래스를 사용해야 하는 IBM MQ classes for Java 엔터프라이즈 애플리케이션은 해당 클래스를 사용할 IBM MQ classes for JMS 수 있어야 합니다.

WebSphere Liberty 그리고 IBM MQ 리소스 어댑터

리소스 어댑터와 IBM MQIBM MQ 리소스 어댑터는 메시징 공급자 또는 WebSphere LibertyIBM MQ 일반적인 Java Platform, Enterprise Edition Connector Architecture (Java EE JCA) 지원을 사용하여 에 WebSphere Liberty 설치할 수 있습니다 Jakarta Messaging .

메시징 WebSphere LibertyIBM MQ 공급자를 사용하여 리소스 어댑터를 설치하는 것이 권장됩니다. 이 구성에서 리소스 어댑터는 와 긴밀하게 WebSphere Liberty 결합되어 있으며 다음과 같은 기능을 활용할 수 있습니다:

- 연속적인 전달 실패가 발생한 후 메시징 엔드포인트를 일시 중지하는 기능.
- 아웃바운드 메시징과 함께 클라이언트 채널 정의 테이블(CCDT)을 사용할 때 ‘XA’ 트랜잭션을 완벽하게 지원합니다(단, 이 기능은 활성화 사양에서는 사용할 수 없습니다).
- ▶ **z/OS** 에 배치 Java 애플리케이션을 z/OS 실행하는 기능.
- 전체 추적 통합으로, 리소스 IBM MQ 어댑터 추적이 추적 WebSphere Liberty 파일에 기록됩니다.

리소스 어댑터와 IBM MQIBM MQ 리소스 어댑터가 일반 리소스 Jakarta Messaging 어댑터로 설치된 경우, 이를 표준 WebSphere Liberty 타사 어댑터로 취급합니다. 이는 앞서 언급된 기능이 사용할 수 없음을 의미합니다.

참고: ▶ **z/OS** 리소스 어댑터 및 IBM MQ 리소스 어댑터를 IBM MQ 일반 WebSphere Liberty 리소스로 Jakarta Messaging 설치하는 것은 에서 지원되지 z/OS 않습니다.

▶ JMS 2.0 메시징 WebSphere Liberty IBM MQ 공급자와 함께 리소스 IBM MQ 어댑터 사용

메시징 WebSphere Liberty IBM MQ 공급자는 애플리케이션 서버 내에서 실행되는 (JMS) 애플리케이션이 메시징 JMS 리소스의 외부 Java Message Service 공급자로서 시스템을 IBM MQ 사용할 수 있도록 합니다.

이 구성에서 리소스 IBM MQ 어댑터는 메시징 WebSphere Liberty IBM MQ 공급자와 긴밀하게 결합되어 다음과 같은 기능을 제공합니다:

- 연속적인 전달 실패가 발생한 후 메시징 엔드포인트를 일시 중지하는 기능.
- 아웃바운드 메시징과 함께 클라이언트 채널 정의 테이블(CCDT)을 사용할 때 ‘XA’ 트랜잭션을 완벽하게 지원 합니다(단, 이 기능은 활성화 사양에서는 사용할 수 없습니다).
- ▶ **z/OS**에 배치 Java 애플리케이션을 z/OS 실행하는 기능.
- 전체 추적 통합으로, 리소스 IBM MQ 어댑터 추적이 추적 WebSphere Liberty 파일에 기록됩니다.

메시징 WebSphere Liberty IBM MQ 공급자와 리소스 IBM MQ 어댑터를 사용하기 위해 필요한 기능

메시징 WebSphere Liberty IBM MQ 공급자와 리소스 IBM MQ 어댑터를 사용하려면 wmqJmsClient-2.0 WebSphere Liberty 기능을 활성화해야 합니다. 자세한 정보는 WebSphere Liberty 제품 문서에서 다음 토픽을 참조하십시오.

- [IBM MQ 용 JMS 2.0 클라이언트](#)
- [IBM MQ 메시징 공급자를 사용하기 위해 Liberty에 JMS 애플리케이션 배포](#)

리소스 IBM MQ 어댑터가 큐 관리자에 대한 보안 TLS 연결을 생성해야 하는 경우, WebSphere LibertytransportSecurity-1.0 기능도 활성화해야 합니다. 이 기능에 대한 자세한 내용은 제품 WebSphere Liberty 설명서의 다음 항목을 참조하십시오:

- [Liberty에서 SSL 통신 사용](#)
- [SSL Liberty의 기본값](#)
- [Transport Security 1.0](#)

참고: 응용 프로그램의 기능에 따라 다른 기능이 필요할 수 있습니다. 자세한 정보는 WebSphere Liberty 제품 문서를 참조하십시오.

리소스 어댑터 설치 IBM MQ 검증 테스트 프로그램을 사용하여 리소스 IBM MQ 어댑터가 올바르게 설치되었는지 확인할 수 있습니다. 자세한 정보는 [442 페이지의 『리소스 어댑터 IBM MQ 설치 확인』](#)의 내용을 참조하십시오.

wmqJmsClient-2.0 기능 사용 시 제한 사항

wmqJmsClient-2.0 기능을 사용할 때 리소스 어댑터에는 다음 제한 사항이 적용됩니다:

- 해당 항목은 IBM MQ classes for Java 지원되지 않습니다. 자세한 정보는 [322 페이지의 『애플리케이션 IBM MQ classes for Java 실행 Java EE』](#)의 내용을 참조하십시오.
- IBM MQ 자원 어댑터에는 BINDINGS_THEN_CLIENT의 전송 유형이 있습니다. 이 운송 유형은 지원되지 않습니다.
- 외부 JNDI 저장소(예: LDAP 또는 파일 시스템 저장소)에서 관리되는 객체(연결 팩토리 JMS 및 대상)를 조회하는 기능은 지원되지 않습니다. 애플리케이션은 구성에 WebSphere Liberty 정의된 관리 객체를 JMS 사용해야 합니다.

연속적인 전달 실패가 발생한 후 메시징 엔드포인트 일시 중지

메시징 WebSphere Liberty IBM MQ 공급자는 연속적인 전달 실패가 발생할 경우 메시지 엔드포인트를 일시 중지하고 메시지 주도형 빈(MDB) 애플리케이션으로의 메시지 전달을 중지하도록 구성할 수 있습니다. 이 설정 방법에 대한 자세한 내용은 [441 페이지의 『IBM MQ 메시지에 의한 WebSphere Liberty의 Bean 일시정지』](#).

XA 클라이언트 채널 정의 테이블을 통한 트랜잭션 지원

메시징 IBM MQ 공급자를 WebSphere Liberty 사용하여 아웃바운드 메시징을 수행할 때는, 클라이언트 채널 정의 테이블(CCDT) 내의 큐 관리자 그룹을 XA 트랜잭션과 함께 활용할 수 있습니다. 이는 큐 관리자 그룹이 제공하는 워크로드 분산 및 가용성을 활용하면서도 트랜잭션 무결성을 유지할 수 있음을 의미합니다.

큐 관리자에 대한 연결 오류가 발생할 경우 큐 관리자를 다시 사용 가능하게 해야 트랜잭션을 해결할 수 있습니다. 트랜잭션 복구는 WebSphere Liberty에서 관리하므로, 큐 관리자가 다시 사용 가능해질 적절한 시간이 허용 되도록 트랜잭션 관리자를 구성해야 합니다. 자세한 내용은 제품 WebSphere Liberty 설명서의 [트랜잭션 관리자 \(트랜잭션\)](#) 항목을 참조하십시오.

배치 애플리케이션 Java 실행 z/OS



에서 z/OS, 메시징 WebSphere Liberty IBM MQ 공급자를 배치 Java 환경에서 사용할 수 있습니다. 자세한 정보는 WebSphere Liberty 제품 문서에서 다음 토픽을 참조하십시오.

- [IBM MQ 메시징 공급자를 사용하여 다중 서버 지원 활성화](#)
- [IBM MQ 메시징 공급자를 사용하여 다중 서버 파티션 지원 활성화](#)

JM 3.0 메시징 WebSphere Liberty IBM MQ 공급자와 함께 Jakarta Messaging 리소스

IBM MQ 어댑터 사용

에 대해 구성된 Jakarta Messaging 서버의 WebSphere Liberty 경우, 메시징 WebSphere Liberty IBM MQ 공급자는 애플리케이션이 외부 제공자로서 를 IBM MQ 사용할 수 있도록 Jakarta Messaging 합니다.

이 구성에서 의 리소스 IBM MQ 어댑터는 Jakarta Messaging 메시징 WebSphere Liberty IBM MQ 공급자와 긴밀하게 결합되어 다음과 같은 기능을 제공합니다:

- 연속적인 전달 실패가 발생한 후 메시징 엔드포인트를 일시 중지하는 기능.
- 아웃바운드 메시징과 함께 클라이언트 채널 정의 테이블(CCDT)을 사용할 때 ‘XA’ 트랜잭션을 완벽하게 지원합니다(단, 이 기능은 활성화 사양에서는 사용할 수 없습니다).
- 전체 추적 통합, 여기서 추적용 리소스 IBM MQ Jakarta Messaging 어댑터가 추적 WebSphere Liberty 파일에 기록됩니다.

메시징 WebSphere Liberty IBM MQ 공급자와 함께 리소스 IBM MQ 어댑터를 Jakarta Messaging 사용하기 위해 필요한 기능

메시징 WebSphere Liberty IBM MQ 공급자와 함께 Jakarta Messaging 리소스 IBM MQ 어댑터를 사용하려면 다음 WebSphere Liberty 기능을 활성화해야 합니다:

- jakartaee-9.1 또는 jakartaee-10.0
- wmqMessagingClient-3.0

자세한 정보는 WebSphere Liberty 제품 문서에서 다음 토픽을 참조하십시오.

- [Jakarta EE 플랫폼 9.1](#)
- [Jakarta EE 플랫폼 10.0](#)
- [3.0 용 메시징 클라이언트 \(IBM MQ\)](#)

서버는 WebSphere Liberty 또한 또는 이후 Java SE 11 버전으로 실행되어야 합니다.

리소스 IBM MQ 어댑터가 큐 관리자에 대한 보안 TLS Jakarta Messaging 연결을 생성해야 하는 경우, WebSphere Liberty transportSecurity-1.0 기능도 활성화해야 합니다. 이 기능에 대한 자세한 내용은 제품 WebSphere Liberty 설명서의 다음 항목을 참조하십시오:

- [Liberty에서 SSL 통신 사용](#)
- [SSL Liberty의 기본값](#)
- [Transport Security 1.0](#)

참고: 응용 프로그램의 기능에 따라 다른 기능이 필요할 수 있습니다. 자세한 정보는 WebSphere Liberty 제품 문서를 참조하십시오.

설치 검증 테스트 Jakarta Messaging 프로그램용 리소스 IBM MQ 어댑터를 사용하여 해당 IBM MQ 리소스 Jakarta Messaging 어댑터가 올바르게 설치되었는지 확인할 수 있습니다. 자세한 정보는 [442 페이지의 『리소스 어댑터 IBM MQ 설치 확인』](#)의 내용을 참조하십시오.

wmqMessagingClient-3.0 기능 사용 시 제한 사항

wmqMessagingClient-3.0 기능을 사용할 때 리소스 어댑터에는 다음 제한 사항이 적용됩니다:

- 해당 항목은 IBM MQ classes for Java 지원되지 않습니다. 자세한 정보는 [322 페이지의 『애플리케이션 IBM MQ classes for Java 실행 Java EE』](#)의 내용을 참조하십시오.
- 의 IBM MQ 리소스 어댑터는 전송 유형이 Jakarta Messaging BINDINGS_THEN_CLIENT입니다. 이 운송 유형은 지원되지 않습니다.
- 외부 JNDI 저장소(예: LDAP 또는 파일 시스템 저장소)에서 관리되는 객체(연결 팩토리 Jakarta Messaging 및 대상)를 조회하는 기능은 지원되지 않습니다. 애플리케이션은 구성에 WebSphere Liberty 정의된 관리 객체를 Jakarta Messaging 사용해야 합니다.

연속적인 전달 실패가 발생한 후 메시징 엔드포인트 일시 중지

메시징 WebSphere Liberty IBM MQ 공급자는 연속적인 전달 실패가 발생할 경우 메시지 엔드포인트를 일시 중지하고 메시지 주도형 빈(MDB) 애플리케이션으로의 메시지 전달을 중지하도록 구성할 수 있습니다. 이 설정 방법에 대한 자세한 내용은 [441 페이지의 『IBM MQ 메시지에 의한 WebSphere Liberty의 Bean 일시정지』](#).

XA 클라이언트 채널 정의 테이블을 통한 트랜잭션 지원

메시징 IBM MQ 공급자를 WebSphere Liberty 사용하여 발신 메시징을 수행할 때는, 클라이언트 채널 정의 테이블(CCDT) 내의 큐 관리자 그룹을 XA 트랜잭션과 함께 활용할 수 있습니다. 이는 큐 관리자 그룹이 제공하는 워크로드 분산 및 가용성을 활용하면서도 트랜잭션 무결성을 유지할 수 있음을 의미합니다.

큐 관리자에 대한 연결 오류가 발생할 경우 큐 관리자를 다시 사용 가능하게 해야 트랜잭션을 해결할 수 있습니다. 트랜잭션 복구는 WebSphere Liberty에서 관리하므로, 큐 관리자가 다시 사용 가능해질 적절한 시간이 허용되도록 트랜잭션 관리자를 구성해야 합니다. 자세한 내용은 제품 WebSphere Liberty 설명서의 [트랜잭션 관리자\(트랜잭션\)](#) 항목을 참조하십시오.

배치 애플리케이션 Java 실행 z/OS



에서 z/OS, 메시징 WebSphere Liberty IBM MQ 공급자를 배치 Java 환경에서 사용할 수 있습니다. 자세한 정보는 WebSphere Liberty 제품 문서를 참조하십시오.

- [IBM MQ 메시징 공급자를 사용하여 다중 서버 지원 활성화](#)
- [IBM MQ 메시징 공급자를 사용하여 다중 서버 파티션 지원 활성화](#)

JMS 2.0 리소스 IBM MQ 어댑터를 일반 리소스 어댑터로 사용하기

리소스 IBM MQ 어댑터를 일반 리소스 WebSphere Liberty 어댑터로 설치할 수 있습니다.

리소스 IBM MQ 어댑터를 일반 리소스 어댑터로 사용하기 위해 필요한 기능

리소스 IBM MQ 어댑터를 일반 리소스 어댑터로 사용하려면 다음 WebSphere Liberty 기능을 활성화해야 합니다:

- jndi-1.0
- jms-2.0

중요사항: wmqJmsClient-2.0 기능은 활성화되어서는 안 됩니다.

자세한 정보는 WebSphere Liberty 제품 문서에서 다음 토픽을 참조하십시오.

- [자바 네이밍 및 디렉터리 인터페이스 1.0](#)
- [Java 메시지 서비스 2.0](#)

리소스 IBM MQ 어댑터가 큐 관리자에 대한 보안 TLS 연결을 생성해야 하는 경우, WebSphere LibertytransportSecurity-1.0 기능을 활성화해야 합니다. 이 기능에 대한 자세한 내용은 제품 WebSphere Liberty 설명서의 다음 항목을 참조하십시오:

- [Liberty에서 SSL 통신 사용](#)
- [SSL Liberty의 기본값](#)
- [Transport Security 1.0](#)

참고: 응용 프로그램의 기능에 따라 다른 기능이 필요할 수 있습니다. 자세한 정보는 WebSphere Liberty 제품 문서를 참조하십시오.

리소스 IBM MQ 어댑터를 로드하도록 WebSphere Liberty 구성

리소스 IBM MQ 어댑터를 일반 리소스 어댑터로 정의할 때는 다음 XML 요소를 사용하여 어디에서 로드할지 WebSphere Liberty 지정해야 합니다:

```
<resourceAdapter id="mqJms" location="<directory_path>/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

<directory_path>는 리소스 IBM MQ 어댑터가 위치한 디렉터리의 경로입니다.

중요사항: 태그의 id 값은 를 제외한 모든 값이 wmqJms 될 수 있습니다. 태그는 wmqJms wmqJmsClient-2.0 기능 전용으로 예약되어 있습니다. 리소스 IBM MQ 어댑터가 일반 리소스 어댑터로 구성된 경우에 사용하면, 이를 올바르게 WebSphere Liberty 로드할 수 없습니다.

그런 다음 엔터프라이즈 애플리케이션을 구성할 때 id 태그의 값을 사용하여 해당 애플리케이션이 리소스 IBM MQ 어댑터가 제공하는 클래스에 접근할 수 있도록 할 수 있습니다. 이를 보여주는 예는 다음과 같습니다:

```
<application id="WMQHTTP" location="${server.config.dir}/apps/WMQHTTP.war"
name="WMQHTTP" type="war">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"
classProviderRef="mqJms"/>
</application>
```

리소스 IBM MQ 어댑터를 일반 리소스 어댑터로 사용할 때의 제한 사항

리소스 IBM MQ 어댑터를 일반 리소스 어댑터로 사용할 때 다음 제한 사항이 적용됩니다:

-  리소스 IBM MQ 어댑터를 일반 리소스 어댑터로 사용하는 것은 에서 지원되지 z/OS않습니다.
- 연속적인 전달 실패가 발생한 후 메시징 엔드포인트를 일시 중지하는 기능은 제공되지 않습니다.
- XA 큐 매니저 그룹에 대한 트랜잭션 지원은 제공되지 않습니다.
- 추적 및 로깅은, WebSphere LibertyNET 프레임워크와 통합되지 않습니다.
- IBM MQ 자원 어댑터에는 BINDINGS_THEN_CLIENT의 전송 유형이 있습니다. 이 운송 유형은 지원되지 않습니다.
- 외부 JNDI 저장소(예: LDAP 또는 파일 시스템 저장소)에서 관리되는 객체(연결 팩토리 JMS 및 대상)를 조회하는 기능은 지원되지 않습니다. 애플리케이션은 구성에 WebSphere Liberty 정의된 관리 객체를 JMS 사용해야 합니다.

리소스 IBM MQ 어댑터를 일반 리소스 어댑터로 추적하기

리소스 IBM MQ 어댑터가 일반 리소스 WebSphere Liberty 어댑터로 배포될 경우, 해당 어댑터의 추적 정보는 추적 WebSphere Liberty 메커니즘과 통합되지 않습니다. 대신, [JMS 애플리케이션을 위한 IBM MQ 클래스](#) 추적에 설명된 대로 시스템 속성 Java 또는 구성 IBM MQ classes for JMS 파일을 사용하여 리소스 IBM MQ 어댑터 추

적을 활성화해야 합니다. WebSphere Liberty 에서 시스템 속성을 Java 설정하는 방법에 대한 자세한 WebSphere Liberty 내용은 해당 문서에서 확인하십시오.

예를 들어, `19.0.0.9` WebSphere Liberty 에서 리소스 IBM MQ 어댑터의 추적을 활성화하려면 다음 단계를 수행하십시오:

1. 해당 `${server.config.dir}` 디렉토리에 `jvm.options` 라는 이름의 텍스트 파일이 아직 존재하지 않는다면 생성하십시오.
2. 파일을 `jvm.options` 편집하고 다음 줄을 추가하십시오:

```
-Dcom.ibm.msg.client.commonservices.trace.status=ON
-Dcom.ibm.msg.client.commonservices.trace.outputName=<path_to_trace_directory>\MQRA-WLP_%PID%.trc
```

where는 추적 파일이 `<path_to_trace_directory>` 작성될 디렉터리입니다.

3. 애플리케이션 서버를 다시 시작하십시오.

그러면 디렉토리 `<path_to_trace_directory>`의 `MQRA-WLP_<process identifier>.trc(이)`라는 추적 파일에 추적이 기록됩니다.

JM 3.0 를 일반 리소스 **IBM MQ Jakarta Messaging** 어댑터로 사용하기 위한 리소스 어댑터 리소스 IBM MQ 어댑터를 일반 리소스 Jakarta Messaging WebSphere Liberty 어댑터로 설치할 수 있습니다.

일반 리소스 어댑터로 사용하기 Jakarta Messaging 위한 리소스 IBM MQ 어댑터의 필수 기능

리소스 IBM MQ 어댑터를 일반 리소스 Jakarta Messaging 어댑터로 사용하려면 활성화해야 하는 기능은 사용 중인 플랫폼 Jakarta EE 버전에 따라 다릅니다.

플랫폼을 Jakarta EE 9.1 위해:

- jakartaee-9.1
- messaging-3.0

플랫폼을 Jakarta EE 10 위해:

- jakartaee-10.0
- messaging-3.1

중요사항: `wmqMessagingClient-3.0` 기능은 활성화되어서는 안 됩니다.

서버는 WebSphere Liberty 또한 또는 이후 Java SE 11 버전으로 실행되어야 합니다.

자세한 정보는 WebSphere Liberty 제품 문서에서 다음 토픽을 참조하십시오.

- [Jakarta EE 플랫폼 9.1](#)
- [Jakarta EE 플랫폼 10.0](#)
- [자카르타 메시징 3.0](#)
- [자카르타 메시징 3.1](#)

리소스 IBM MQ 어댑터가 큐 관리자에 대한 보안 TLS Jakarta Messaging 연결을 생성해야 하는 경우, WebSphere Liberty `transportSecurity-1.0` 기능을 활성화해야 합니다. 이 기능에 대한 자세한 내용은 제품 WebSphere Liberty 설명서의 다음 항목에서 확인할 수 있습니다:

- [Liberty에서 SSL 통신 사용](#)
- [SSL Liberty의 기본값](#)
- [Transport Security 1.0](#)

참고: 응용 프로그램의 기능에 따라 다른 기능이 필요할 수 있습니다. 자세한 정보는 WebSphere Liberty 제품 문서를 참조하십시오.

리소스 IBM MQ 어댑터를 로드하도록 WebSphere Liberty 구성 Jakarta Messaging

리소스 IBM MQ 어댑터를 일반 리소스 Jakarta Messaging 어댑터로 정의할 때는 다음 XML 요소를 사용하여 해당 어댑터를 어디에서 로드할지 WebSphere Liberty 지정해야 합니다:

```
<resourceAdapter id="mqJakartaRa" location="<directory_path>/wmq.jakarta.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

<directory_path>는 의 리소스 IBM MQ Jakarta Messaging 어댑터가 위치한 디렉터리의 경로입니다.

중요사항: 태그의 id 값은 를 제외한 모든 값이 wmqJms 될 수 있습니다. 태그는 wmqJms wmqMessagingClient-3.0 기능 전용으로 예약되어 있습니다. 리소스 IBM MQ 어댑터가 일반 리소스 어댑터로 Jakarta Messaging 구성된 경우, WebSphere Liberty 는 이를 올바르게 로드할 수 없습니다.

그런 다음 엔터프라이즈 애플리케이션을 구성할 때 id 태그의 값을 사용하여 해당 애플리케이션이 리소스 IBM MQ 어댑터가 제공하는 클래스에 접근할 수 있도록 할 수 Jakarta Messaging 있습니다. 이를 보여주는 예는 다음과 같습니다:

```
<application id="WMQHTTP" location="${server.config.dir}/apps/WMQHTTP.war" name="WMQHTTP"
type="war">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"
classProviderRef="mqJakartaRa"/>
</application>
```

일반 리소스 IBM MQ Jakarta Messaging 어댑터로 사용할 때의 제한 사항

다음 제한 사항은 리소스 IBM MQ 어댑터를 일반 리소스 Jakarta Messaging 어댑터로 사용할 때 적용됩니다:

-  리소스 IBM MQ 어댑터를 일반 리소스 Jakarta Messaging 어댑터로 사용하는 것은 에서 지원되지 않습니다.
- 연속적인 전달 실패가 발생한 후 메시징 엔드포인트를 일시 중지하는 기능은 제공되지 않습니다.
- XA 큐 매니저 그룹에 대한 트랜잭션 지원은 제공되지 않습니다.
- 추적 및 로깅은. WebSphere LibertyNET 프레임워크와 통합되지 않습니다.
- 의 IBM MQ 리소스 어댑터는 전송 유형이 Jakarta Messaging BINDINGS_THEN_CLIENT입니다. 이 운송 유형은 지원되지 않습니다.
- 외부 JNDI 저장소(예: LDAP 또는 파일 시스템 저장소)에서 관리되는 객체(연결 팩토리 Jakarta Messaging 및 대상)를 조회하는 기능은 지원되지 않습니다. 애플리케이션은 구성에 WebSphere Liberty 정의된 관리 객체를 Jakarta Messaging 사용해야 합니다.

일반 리소스 Jakarta Messaging 어댑터로서의 리소스 IBM MQ 어댑터 추적

리소스 IBM MQ 어댑터가 일반 리소스 Jakarta Messaging WebSphere Liberty 어댑터로 배포될 경우, 해당 어댑터의 추적 정보는 추적 WebSphere Liberty 메커니즘과 통합되지 않습니다. 대신, [JMS 애플리케이션용 IBM MQ 클래스](#) 추적에 설명된 대로 시스템 속성 Java 또는 구성 IBM MQ classes for JMS 파일을 사용하여 추적용 Jakarta Messaging 리소스 IBM MQ 어댑터를 활성화해야 합니다. [WebSphere Liberty](#) 에서 시스템 속성을 Java 설정하는 방법에 대한 자세한 WebSphere Liberty 내용은 해당 문서에서 확인하십시오.

예를 들어, ` 19.0.0.9WebSphere Liberty ` Jakarta Messaging 에서 리소스 IBM MQ 어댑터의 추적을 활성화하려면 다음 단계를 수행하십시오:

1. 해당 \${server.config.dir} 디렉토리에 jvm.options 라는 이름의 텍스트 파일이 아직 존재하지 않는다면 생성하십시오.
2. 파일을 jvm.options 편집하고 다음 줄을 추가하십시오:

```
-Dcom.ibm.msg.client.commonservices.trace.status=ON
-Dcom.ibm.msg.client.commonservices.trace.outputName=<path_to_trace_directory>\MQRA-WLP_%PID%.trc
```

where는 추적 파일이 <path_to_trace_directory> 작성될 디렉터리입니다.

3. 애플리케이션 서버를 다시 시작하십시오.

그러면 디렉토리 <path_to_trace_directory>의 MQRA-WLP_<process identifier>.trc(이)라는 추적 파일에 추적이 기록됩니다.

리소스 IBM MQ 어댑터 설치

리소스 어댑터 및 IBM MQ 용 IBM MQ 리소스 어댑터는 리소스 Jakarta Messaging 아카이브(RAR) 파일로 제공됩니다. 리소스 IBM MQ 어댑터(wmq.jmsra.rar)용 RAR 파일은 J2EE Java Platform, Enterprise Edition 7 1Java EE 7.0 사양 또는 Java Platform, Enterprise Edition 8 J2EE 1Java EE 8.1 사양을 지원하는 모든 애플리케이션 서버에 배포할 수 있습니다. (wmq.jakarta.jmsra.rar) Jakarta Messaging 용 리소스 IBM MQ 어댑터의 RAR 파일은 (또는 이후 Jakarta EE 9.1 버전) 사양을 지원하는 애플리케이션 서버에 배포할 수 있습니다.

리소스 어댑터 및 IBM MQ 리소스 어댑터 획득 Jakarta Messaging

메시징 Java 구성 요소의 일부로 설치된 IBM MQ 환경에서 Jakarta Messaging 또는 에서 제공되는 다운로드를 통해 해당 IBM MQ 리소스 Fix Central 어댑터와 IBM MQ 리소스 어댑터를 얻을 수 있습니다.

이 태스크 정보

IBM MQ 애플리케이션 서버에 설치할 수 있는 두 가지 리소스 어댑터를 제공합니다:

- **JMS 2.0** 리소스 IBM MQ 어댑터는 호출됩니다 `wmq.jmsra.rar`. 리소스 어댑터(RAR) 파일에는 (JCA) Java EE Connector Architecture 인터페이스의 및 IBM MQ classes for JMS IBM MQ 구현이 포함되어 있습니다. RAR 파일은 또는 Java EE 8 Java EE 7 사양을 지원하는 애플리케이션 서버에 설치할 수 있습니다.
- **JM 3.0** 의 IBM MQ 리소스 어댑터는 Jakarta Messaging 라고 합니다 `wmq.jakarta.jmsra.rar`. 리소스 어댑터(RAR) 파일에는 (JCA) Jakarta Connectors Architecture 인터페이스의 및 IBM MQ classes for Jakarta Messaging IBM MQ 구현이 포함되어 있습니다. 이 리소스 어댑터는 (또는 이후 Jakarta EE 9.1 버전) 사양을 지원하는 애플리케이션 서버에 설치할 수 있습니다.

리소스 어댑터는 메시징 구성 Java 요소의 일부로 설치됩니다. 이 구성 요소가 설치되면 RAR 파일은 에 표시된 디렉터리에서 [398 페이지의 표 61](#) 찾을 수 있습니다.

표 61. IBM MQ 각 플랫폼별 RAR 파일이 포함된 디렉터리	
플랫폼	디렉터리
AIX and Linux	<code>MQ_INSTALLATION_PATH/java/lib/jca</code>
IBM i	<code>/QIBM/ProdData/mqm/java/lib/jca</code>
Windows	<code>MQ_INSTALLATION_PATH\java\lib\jca</code>
z/OS	<code>MQ_INSTALLATION_PATH/java/lib/jca</code>

`MQ_INSTALLATION_PATH`은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

해당 디렉터리에는 두 개의 설치 검증 테스트(IVT) 프로그램이 포함되어 있습니다:

- 리소스 IBM MQ 어댑터에 대한 IVT 프로그램은 입니다 `wmq.jmsra.ivt.ear`.
- 리소스 IBM MQ 어댑터의 IVT 프로그램은 Jakarta Messaging 입니다 `wmq.jakarta.jmsra.ivt.ear`.

이러한 IVT 프로그램은 애플리케이션 서버에 리소스 어댑터가 올바르게 설치되었는지 확인하는 데 사용할 수 있습니다. 자세한 정보는 [442 페이지의 『리소스 어댑터 IBM MQ 설치 확인』](#)의 내용을 참조하십시오.

리소스 어댑터 및 IVT 프로그램은 별도로 다운로드할 수도 있습니다 Fix Central. 리소스 어댑터 및 IVT를 다운로드하고 추출하는 Fix Central 단계는 [절차](#)에 설명되어 있습니다.

참고: 다음 작업을 시작하기 전에 컴퓨터에 (JRE) Java runtime environment 가 설치되어 있는지, 그리고 시스템 경로에 ()가 JRE 추가되었는지 확인하십시오.

이 설치 프로세스에서 사용된 Java 설치 프로그램은 루트 또는 특성 사용자로 실행 중일 필요가 없습니다. 유일한 요구 사항은 파일을 저장할 디렉터리에 대해 실행 중인 사용자가 쓰기 권한을 가져야 한다는 것입니다.

프로시저

1. 리소스 어댑터 IBM MQ 설치 패키지를 에서 Fix Central다운로드하십시오.

- a) 다음 링크를 클릭하십시오. [IBM MQ 자원 어댑터](#).
- b) 사용 가능한 수정 사항 목록에서 해당 IBM MQ 버전의 리소스 어댑터를 찾으십시오.
예를 들면, 다음과 같습니다.

```
release level: 10.0.0.0-IBM-MQ-Java-InstallRA
Long Term Support Release: 10.0.0.0 IBM MQ Resource Adapter for use with Application
Servers
```

그런 다음 자원 어댑터 파일 이름을 클릭하고 다운로드 프로세스를 수행하십시오.

2. 리소스 어댑터 IBM MQ 설치 패키지를 다운로드한 디렉터리에서 다음 명령을 입력하여 설치를 시작하십시오:

```
java -jar V.R.M.F-IBM-MQ-Java-InstallRA.jar
```

여기서, *V.R.M.F*는 버전, 릴리스, 수정 및 수정팩 번호이고 *V.R.M.F-IBM-MQ-Java-InstallRA.jar*은 (는) Fix Central에서 다운로드한 파일의 이름입니다.

예를 들어, IBM MQ 10.0.0 릴리스용 리소스 IBM MQ 어댑터를 설치하려면 다음 명령을 사용합니다:

```
java -jar 10.0.0.0-IBM-MQ-Java-InstallRA.jar
```

3. 라이선스 계약서 표시 여부를 선택하십시오.

리소스 어댑터를 설치하는 명령을 입력하면 라이선스 계약이 표시됩니다:

<버전 번호 및 날짜>를 사용, 압축 해제 또는 설치하기 전에 다음 사항에 동의해야 합니다

1.의 조건 IBM [제품명] 평가용 국제 라이선스 계약서

프로그램 2. IBM 국제 프로그램 라이선스 계약 및 추가 사항

license information. Please read the following license agreements carefully.

The license agreement is separately viewable using the

--viewLicenseAgreement option.

Press Enter to display the license terms now, or 'x' to skip.

- 라이선스를 표시하려면 Enter를 누르십시오.
- 또는 x를 누르면 라이선스의 표시를 건너뛵니다.

4. 추가 라이선스 약관을 표시할지 여부를 선택하십시오.

라이선스를 확인하거나 X를 누르면, 추가 라이선스 약관을 표시할 수 있다는 다음 메시지가 나타납니다:

Additional license information is separately viewable using the

--viewLicenseInfo option.

Press Enter to display additional license information now, or 'x' to skip.

- 추가 라이선스 조항을 표시하려면 Enter를 누르십시오.
- 또는 x를 누르면 추가 라이선스 조항의 표시를 건너뛵니다.

5. 라이선스 계약에 동의하십시오.

다음으로, 라이선스 계약에 동의하라는 메시지가 표시됩니다:

By choosing the "I Agree" option below, you agree to the terms of the license agreement and non-IBM terms, if applicable. 이 매개변수를 agree, select "I do not Agree".

Select [1] I Agree, or [2] I do not Agree:

- 라이선스 계약을 수락하고 설치 디렉터리 선택을 계속하려면 값 1을 입력하십시오.
- 대신 값 2를 입력하면 설치가 즉시 종료됩니다.

6. 설치 대상 디렉터를 선택하십시오.

라이선스에 동의하면 다음 메시지가 표시되며, 대상 설치 디렉터를 선택하라는 요청을 받게 됩니다:

Enter directory for product files or leave blank to accept the default value.

기본 대상 디렉터리는 <현재 디렉터리>입니다

Target directory for product files?

where는 리소스 어댑터 IBM MQ 설치 패키지가 <current directory> 다운로드된 디렉터리입니다.

- 기본 위치에 자원 어댑터를 설치하려는 경우, 값을 지정하지 않고 Enter를 누르십시오.
- 기본값과 다른 위치에 자원 어댑터를 설치하려는 경우, 자원 어댑터를 설치하려는 디렉토리의 이름을 지정한 후 Enter를 누르십시오.

선택된 위치에 파일을 설치한 후에 다음 예에 표시된 대로 확인 메시지가 표시됩니다.

```
Extracting files to H:\Liberty\WMQ\wmq
Successfully extracted all product files.
```

설치 중 이름이 wmq인 새 디렉토리가 선택된 설치 디렉토리 내에 작성되고 다음 파일이 wmq 디렉토리에 설치됩니다.

- IBM MQ 리소스 어댑터 (wmq.jmsra.rar).
- IBM MQ 리소스 어댑터 IVT 프로그램 (wmq.jmsra.ivt.ear).
- (wmq.jakarta.jmsra.rar) Jakarta Messaging 용 IBM MQ 리소스 어댑터.
- IVT Jakarta Messaging 프로그램용 IBM MQ 리소스 어댑터 (wmq.jakarta.jmsra.ivt.ear).

다음에 수행할 작업

시스템에 리소스 IBM MQ 어댑터 및 IBM MQ 리소스 어댑터가 Jakarta Messaging 설치된 후(설치 IBM MQ 시 메시징 구성 Java 요소의 일부로 또는 에서 제공되는 다운로드를 통해 설치된 경우), 해당 Fix Central 어댑터들은 애플리케이션 서버에 배포할 준비가 완료됩니다.

JMS 2.0 메시징 WebSphere Liberty IBM MQ 공급자를 사용하여 WebSphere Liberty 리소스 IBM MQ 어댑터를 설치합니다

리소스 IBM MQ 어댑터를 사용하는 권장 방법은 메시징 WebSphere Liberty WebSphere Liberty IBM MQ 공급자를 사용하여 설치하는 것입니다.

이 태스크 정보

이 구성에서 리소스 어댑터는 애플리케이션 서버와 긴밀하게 결합되어 다음과 같은 기능을 제공합니다:

- 연속적인 전달 실패가 발생한 후 메시징 엔드포인트를 일시 중지하는 기능.
- 아웃바운드 메시징과 함께 클라이언트 채널 정의 테이블(CCDT)을 사용할 때 'XA' 트랜잭션을 완벽하게 지원합니다(단, 이 기능은 활성화 사양에서는 사용할 수 없습니다).
- **z/OS**에 배치 Java 애플리케이션을 z/OS 실행하는 기능.
- 전체 추적 통합으로, 리소스 IBM MQ 어댑터 추적이 추적 WebSphere Liberty 파일에 기록됩니다.

자세한 정보는 392 페이지의 『메시징 WebSphere Liberty IBM MQ 공급자와 함께 리소스 IBM MQ 어댑터 사용』의 내용을 참조하십시오.

프로시저

1. 리소스 IBM MQ 어댑터를 동일한 시스템에 설치하십시오 WebSphere Liberty.
이 작업을 수행하는 방법에 대한 자세한 내용은 을 참조하십시오 398 페이지의 『리소스 어댑터 및 IBM MQ IBM MQ 리소스 어댑터 획득 Jakarta Messaging』.
2. WebSphere Liberty 서버를 중지하십시오.
3. 파일을 WebSphere Libertyserver.xml 편집하고, <featureManager> 섹션에 다음 항목이 포함되어 있는지 확인하십시오:

```
<feature>wmqJmsClient-2.0</feature>
<feature>jndi-1.0</feature>
<feature>jsp-2.3</feature>
<feature>mdb-3.2</feature>
```

4. server.xml 파일에 다음 행을 추가하십시오.

```
<variable name="wmqJmsClient.rar.location" value="<directory_path>\wmq.jmsra.rar"/>
```


where <directory_path> 는 리소스 IBM MQ 어댑터가 포함된 디렉터리입니다. 이 설정은 메시징 WebSphere LibertyIBM MQ 공급자가 리소스 어댑터를 찾기 위해 사용하는 변수를 wmqJmsClient.rar.location 지정합니다.

5. 와 동일한 머신에 IBM MQ 설치된 WebSphere Liberty 경우, 애플리케이션 서버 내부에서 실행 중인 애플리케이션이 BINDINGS 전송을 사용하여 큐 관리자에 연결하려는 경우 다음 줄을 파일에 server.xml 추가하십시오:

```
<wmqJmsClient nativeLibraryPath="<MQ_INSTALLATION_PATH>/java/lib64"/>
```

는 가 IBM MQ<MQ_INSTALLATION_PATH> 설치된 디렉터리입니다. 이를 통해 메시징 WebSphere LibertyIBM MQ 공급자가 BINDINGS 전송을 사용하여 연결하는 데 필요한 네이티브 IBM MQ 라이브러리를 로드하도록 보장합니다.

6. 서버를 WebSphere Liberty 시작하십시오.

리소스 IBM MQ 어댑터가 올바르게 로드되고 있는지 확인하려면 다음 메시지가 파일에 WebSphere Libertymessages.log 있는지 확인하십시오:

■ V 10.0.0

WMQ 메시징 : '<버전 번호>'.
IBM MQ 리소스 어댑터가 WebSphere Liberty 를 사용하여 WebSphere Liberty 에 배포되었습니다. IBM MQ 메시징 제공업체.

where <version_number> 는 리소스 IBM MQ 어댑터의 버전입니다.

■ JM 3.0 메시징 WebSphere LibertyIBM MQ 공급자를 사용하여 WebSphere Liberty 에 대한 Jakarta Messaging 리소스 IBM MQ 어댑터 설치

리소스 IBM MQ 어댑터를 사용하기 위한 Jakarta Messaging 권장 방법은 메시징 WebSphere LibertyIBM MQ 공급자를 WebSphere Liberty 사용하여 설치하는 것입니다.

이 태스크 정보

이 구성에서 리소스 어댑터는 애플리케이션 서버와 긴밀하게 결합되어 다음과 같은 기능을 제공합니다:

- 연속적인 전달 실패가 발생한 후 메시징 엔드포인트를 일시 중지하는 기능.
- 아웃바운드 메시징과 함께 클라이언트 채널 정의 테이블(CCDT)을 사용할 때 'XA' 트랜잭션을 완벽하게 지원합니다(단, 이 기능은 활성화 사양에서는 사용할 수 없습니다).
- **z/OS** 에 배치 Java 애플리케이션을 z/OS 실행하는 기능.
- 전체 추적 통합, 여기서 추적용 리소스 IBM MQJakarta Messaging 어댑터가 추적 WebSphere Liberty 파일에 기록됩니다.

자세한 정보는 393 페이지의 『메시징 WebSphere LibertyIBM MQ 공급자와 함께 Jakarta Messaging 리소스 IBM MQ 어댑터 사용』의 내용을 참조하십시오.

프로시저

1. 리소스 IBM MQ 어댑터를 에 Jakarta Messaging 설치하십시오 WebSphere Liberty.
이 작업을 수행하는 방법에 대한 자세한 내용은 을 참조하십시오 398 페이지의 『리소스 어댑터 및 IBM MQIBM MQ 리소스 어댑터 획득 Jakarta Messaging』.
2. WebSphere Liberty 서버를 중지하십시오.
3. 파일을 WebSphere Libertyserver.xml 편집하고, 사용 중인 Jakarta EE 버전에 따라 해당 <featureManager> 섹션에 다음 항목이 포함되어 있는지 확인하십시오.
 - 기능은 Jakarta EE 9.1 다음과 같습니다:

```
<feature>jakartaee-9.1</feature>  
<feature>jndi-1.0</feature>  
<feature>mdb-4.0</feature>  
<feature>wmqMessagingClient-3.0</feature>
```

- 기능은 Jakarta EE 10 다음과 같습니다:

```
<feature>jakartaee-10.0</feature>
<feature>jndi-1.0</feature>
<feature>mdb-4.0</feature>
<feature>wmqMessagingClient-3.0</feature>
```

4. server.xml 파일에 다음 행을 추가하십시오.

```
<variable name="wmqJmsClient.rar.location" value="<directory_path>\wmq.jakarta.jmsra.rar"/>
```

where <directory_path> 는 에 대한 리소스 IBM MQ 어댑터가 Jakarta Messaging 포함된 디렉터리입니다. 이 설정은 메시징 WebSphere Liberty IBM MQ 공급자가 리소스 어댑터를 찾기 위해 사용하는 변수를 wmqJmsClient.rar.location 지정합니다.

5. 와 동일한 머신에 IBM MQ 설치된 WebSphere Liberty 경우, 애플리케이션 서버 내부에서 실행 중인 애플리케이션이 BINDINGS 전송을 사용하여 큐 관리자에 연결하려는 경우 다음 줄을 파일에 server.xml 추가하십시오:

```
<wmqJmsClient nativeLibraryPath="<MQ_INSTALLATION_PATH>/java/lib64"/>
```

는 가 IBM MQ<MQ_INSTALLATION_PATH> 설치된 디렉터리입니다. 이를 통해 메시징 WebSphere Liberty IBM MQ 공급자가 BINDINGS 전송을 사용하여 연결하는 데 필요한 네이티브 IBM MQ 라이브러리를 로드하도록 보장합니다.

6. 서버를 WebSphere Liberty 시작하십시오.

리소스 IBM MQ 어댑터가 올바르게 Jakarta Messaging 로드되고 있는지 확인하려면 다음 메시지가 파일에 WebSphere Liberty messages.log 있는지 확인하십시오:

WMQ 1000

WMQ 메시징 : '<버전 번호>'.
IBM MQ 리소스 어댑터가 WebSphere Liberty 를 사용하여 WebSphere Liberty 에 배포되었습니다. IBM MQ 메시징 제공업체.

where <version_number> 는 리소스 IBM MQ 어댑터의 버전입니다.

▶ JMS 2.0 리소스 IBM MQ 어댑터를 일반 리소스 WebSphere Liberty 어댑터로 설치

리소스 IBM MQ 어댑터는 일반 리소스 WebSphere Liberty 어댑터로 설치될 수 있습니다.

이 태스크 정보

이 구성에서는 리소스 IBM MQ 어댑터를 표준 타사 어댑터로 취급하므로 WebSphere Liberty 여러 기능이 사용할 수 없습니다. 자세한 정보는 394 페이지의 『리소스 IBM MQ 어댑터를 일반 리소스 어댑터로 사용하기』의 내용을 참조하십시오.

참고: ▶ z/OS 리소스 IBM MQ 어댑터를 일반 리소스 어댑터로 설치하는 것은 에서 지원되지 WebSphere Liberty z/OS 않습니다.

프로시저

1. 리소스 IBM MQ 어댑터를 동일한 시스템에 설치하십시오 WebSphere Liberty.
이 작업을 수행하는 방법에 대한 자세한 내용은 을 참조하십시오 398 페이지의 『리소스 어댑터 및 IBM MQ IBM MQ 리소스 어댑터 획득 Jakarta Messaging』.
2. WebSphere Liberty 서버를 중지하십시오.
3. 파일을 WebSphere Liberty server.xml 편집하고, <featureManager> 섹션에 다음 항목이 포함되어 있는지 확인하십시오:

```
<feature>jca-1.7</feature>
<feature>jms-2.0</feature>
<feature>jndi-1.0</feature>
<feature>mdb-3.2</feature>
```

4. server.xml 파일에 다음 항목을 추가하십시오.

```
<resourceAdapter id="mqJmsRa" location="<directory_path>/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

where <directory_path> 는 리소스 IBM MQ 어댑터가 포함된 디렉터리입니다. 이는 가 WebSphere Liberty 식별자 를 사용하여 리소스 IBM MQmqJmsRa 어댑터를 로드함을 의미합니다. 리소스 어댑터에 접근해야 하는 엔터프라이즈 애플리케이션은 이 식별자를 사용하여 접근할 수 있습니다.

5. 서버를 WebSphere Liberty 시작하십시오.

리소스 IBM MQ 어댑터가 올바르게 로드되고 있는지 확인하려면, 시작된 WebSphere Liberty 후 다음 \$ {server.config.dir}\mqjms.log.0 메시지가 파일에 있는지 확인하십시오:

```
-----
<timestamp>[기본 Executor-thread-6 ] ResourceAdapterImpl
WMQ 메시징 : '<버전 번호>'.

설명:
널
ACTION:
널
-----
<timestamp>[기본 Executor-thread-6 ] com.ibm.mq.connector.ResourceAdapterImpl
IBM MQ 리소스 어댑터가 WebSphere Liberty 에 일반 리소스 어댑터로 배포되었습니다.

설명:
IBM MQ 리소스 어댑터가 WebSphere Liberty 에 일반 리소스 어댑터로 배포된 경우, WebSphere Liberty 에서 제공하는 일부 기능(예: 연속적인 실패 후 메시징 엔드포인트를 일시 중지하는 기능)을 사용할 수 없습니다. 자세한 내용은 IBM MQ 문서의 " "WebSphere Liberty and IBM MQ 리소스 어댑터" 항목을 참조하십시오.

ACTION:
IBM MQ 문서의 " "WebSphere Liberty and IBM MQ 리소스 어댑터" 항목에 있는 정보를 검토하여 WebSphere Liberty 에서 제공하는 기능 중 필요한 것이 있는지 확인하십시오. 해당 작업을 수행한 경우, 애플리케이션 서버를 재구성하여 WebSphere Liberty 메시징 공급자( IBM MQ )를 사용하여 IBM MQ 리소스 어댑터를 로드하도록 설정하십시오.
-----
```

where <version_number> 는 리소스 IBM MQ 어댑터의 버전입니다.

JM 3.0 일반 리소스 어댑터로 Jakarta Messaging 에 WebSphere Liberty 대한 리소스 IBM MQ 어댑터 설치

의 IBM MQ 리소스 어댑터는 일반 리소스 Jakarta Messaging WebSphere Liberty 어댑터로 에 설치될 수 있습니다.

이 태스크 정보

이 구성에서는 의 IBM MQ 리소스 어댑터를 Jakarta Messaging WebSphere Liberty 표준 타사 어댑터로 취급하므로 여러 기능이 사용할 수 없습니다. 자세한 정보는 396 페이지의 『[리소스 어댑터 및 IBM MQ Jakarta Messaging 어댑터로 사용하기 위한 리소스 어댑터](#)』의 내용을 참조하십시오.

참고:  리소스 IBM MQ 어댑터를 일반 리소스 어댑터로 설치하는 것은 에서 지원되지 WebSphere Liberty z/OS 않습니다.

프로시저

1. 리소스 IBM MQ 어댑터를 에 Jakarta Messaging 설치하십시오 WebSphere Liberty.
이 작업을 수행하는 방법에 대한 자세한 내용은 을 참조하십시오 398 페이지의 『[리소스 어댑터 및 IBM MQ IBM MQ 리소스 어댑터 획득 Jakarta Messaging](#)』.
2. WebSphere Liberty 서버를 중지하십시오.
3. 파일을 WebSphere Libertyserver.xml 편집하고, 사용 중인 Jakarta EE 버전에 따라 해당 <featureManager> 섹션에 다음 항목이 포함되어 있는지 확인하십시오.

- 기능은 Jakarta EE 9.1 다음과 같습니다:

```
<feature>jndi-1.0</feature>
<feature>jakartaee-9.1</feature>
<feature>pages-3.0</feature>
<feature>mdb-4.0</feature>
```

- 기능은 Jakarta EE 10 다음과 같습니다:

```
<feature>jndi-1.0</feature>
<feature>jakartaee-10.0</feature>
<feature>pages-3.1</feature>
<feature>mdb-4.0</feature>
```

4. server.xml 파일에 다음 항목을 추가하십시오.

```
<resourceAdapter id="mqJakartaRa" location="<directory_path>/wmq.jakarta.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

where <directory_path> 는 에 대한 리소스 IBM MQ 어댑터가 Jakarta Messaging 포함된 디렉터리입니다. 이는 가 식별자를 사용하여 에 WebSphere LibertyJakarta Messaging 대한 mqJakartaRa 리소스 IBM MQ 어댑터를 로드함을 의미합니다. 리소스 어댑터에 접근해야 하는 엔터프라이즈 애플리케이션은 이 식별자를 사용하여 접근할 수 있습니다.

5. 서버를 WebSphere Liberty 시작하십시오.

리소스 IBM MQ 어댑터가 올바르게 Jakarta Messaging 로드되고 있는지 확인하려면, 가 시작된 WebSphere Liberty 후 `server.config.dir\mqjms.log.0` 파일에서 다음 메시지를 찾으십시오:

```
-----
<timestamp>[기본 Executor-thread-6 ] ResourceAdapterImpl
WMQ 메시징 : '<버전 번호>'.

설명:
널

ACTION:
널
-----
<timestamp>[기본 Executor-thread-6 ] com.ibm.mq.connector.ResourceAdapterImpl
IBM MQ 리소스 어댑터가 WebSphere Liberty 에 일반 리소스 어댑터로 배포되었습니다.

설명:
IBM MQ 리소스 어댑터가 일반 리소스 어댑터로 WebSphere Liberty 에 배포된 경우, WebSphere Liberty 에서 제공하는 일부 기능(예: 연속적인 오류가 일정 횟수 발생 후 메시징 엔드포인트를 일시 중지하는 기능 등)을 사용할 수 없습니다. 자세한 내용은 IBM MQ 문서의 “ "WebSphere Liberty 및 IBM MQ 리소스 어댑터” 항목을 참조하십시오.

ACTION:
IBM MQ 문서의 “ "WebSphere Liberty 및 IBM MQ 리소스 어댑터” 항목에 있는 정보를 검토하여, WebSphere Liberty 에서 제공하는 기능 중 사용해야 할 것이 있는지 확인하십시오. 이 경우, 애플리케이션 서버를 재구성하여 WebSphere Liberty IBM MQ 메시징 공급자를 사용하여 IBM MQ 리소스 어댑터를 로드하도록 설정하십시오.
-----
```

where <version_number> 는 에 대한 리소스 IBM MQ 어댑터의 Jakarta Messaging 버전입니다.

JMS 2.0 리소스 IBM MQ 어댑터를 설치합니다. WebSphere Application Server 9

WebSphere Application Server 9 리소스 IBM MQ 9.0 어댑터를 내장하며, 이는 픽스 WebSphere Application Server 팩으로 업데이트됩니다. 애플리케이션 서버에 포함된 버전 중 필요한 버전이 현재 제공되지 않는 경우, 특정 버전의 리소스 IBM MQ 9.0 어댑터를 설치할 수 있습니다.

이 태스크 정보

특정 버전의 리소스 IBM MQ 9.0 어댑터를 설치하는 방법에 대한 정보는 제품 WebSphere Application Server 설명서의 " [IBM MQ 리소스 어댑터의 특정 유지 관리 수준 설치](#) " 항목을 참조하십시오.

참고:

- WebSphere Application Server 9 와 Java EE 8Java EE 7 사양을 지원합니다. 해당 Jakarta EE 사양의 어떤 버전도 지원하지 않습니다. 이는 해당 애플리케이션 서버 버전에 Jakarta Messaging 리소스 IBM MQ 어댑터를 설치할 수 없음을 의미합니다.
- WebSphere Application Server 8.5.5 사양을 Java EE 6 지원합니다. 리소스 IBM MQ 9.0 어댑터가 및 Java EE 8 Java EE 7 사양과 호환되고, 용 리소스 IBM MQ 어댑터가 (Jakarta Messaging 및 이후 Jakarta EE 9.1 버전)과 호환되므로, 이러한 리소스 어댑터는 에 설치할 수 WebSphere Application Server 8.5.5 없습니다.

JMS 2.0 다른 애플리케이션 서버에 리소스 **IBM MQ** 어댑터 설치하기

리소스 IBM MQ 어댑터는 및 Java EE 8 Java EE 7 사양을 지원하는 모든 애플리케이션 서버에 설치할 수 있습니다. 이를 수행하는 방법에 대한 정보는 애플리케이션 서버 설명서를 참조하십시오.

이 태스크 정보

Windows Glassfish 애플리케이션 서버에 리소스 IBM MQ 어댑터를 설치하는 방법의 예는 에서 Windows 확인할 수 461 페이지의 『GlassFish 서버에서 리소스 IBM MQ 어댑터 설치 및 테스트』 있습니다.

WildFly 애플리케이션 서버에 리소스 IBM MQ 어댑터를 설치하는 방법의 예는 을 참조하십시오 467 페이지의 『WildFly 에서 리소스 IBM MQ 어댑터 설치 및 테스트』.

참고: 애플리케이션 서버가 로컬 큐 매니저와 동일한 시스템에 IBM MQ 설치된 경우, 리소스 IBM MQ 어댑터가 BINDINGS 전송을 사용하여 로컬 큐 매니저에 연결할 수 있습니다.

- **UNIX** 이를 위해 시스템에서 UNIX 네이티브 인터페이스(Java JNI) 라이브러리가 포함된 디렉터리가 시스템 경로에 있는지 확인하십시오. IBM MQ classes for JMS 라이브러리도 포함하는 이 디렉토리의 위치는 96 페이지의 『JNI(Java Native Interface) 라이브러리 구성』의 내용을 참조하십시오.
- **Windows** 설치 과정에서 Windows이 디렉터리가 IBM MQ classes for JMS시스템 경로에 자동으로 추가됩니다.

JM 3.0 다른 애플리케이션 서버에 **Jakarta Messaging** 리소스 **IBM MQ** 어댑터 설치

해당 IBM MQ 리소스 어댑터는 (또는 이후 Jakarta EE 9.1 버전) 사양을 Jakarta Messaging 지원하는 모든 애플리케이션 서버에 설치할 수 있습니다. 이를 수행하는 방법에 대한 정보는 애플리케이션 서버 설명서를 참조하십시오.

이 태스크 정보

Windows Glassfish 애플리케이션 서버에 리소스 IBM MQ 어댑터를 설치하는 방법의 예는 에서 Windows 확인할 수 464 페이지의 『GlassFish 서버용 Jakarta Messaging 리소스 어댑터 IBM MQ 설치 및 테스트』 있습니다.

WildFly 애플리케이션 서버에 리소스 IBM MQ 어댑터를 설치하는 방법의 예는 을 참조하십시오 468 페이지의 『WildFly 에서 Jakarta Messaging 리소스 IBM MQ 어댑터 설치 및 테스트』.

참고: 애플리케이션 서버가 로컬 큐 매니저와 동일한 시스템에 IBM MQ 설치된 경우, 로컬 큐 매니저용 리소스 IBM MQ 어댑터가 BINDINGS Jakarta Messaging 전송을 사용하여 로컬 큐 매니저에 연결할 수 있습니다.

- **UNIX** 이를 위해 시스템에서 UNIX 네이티브 인터페이스(Java JNI) 라이브러리가 포함된 디렉터리가 시스템 경로에 있는지 확인하십시오. IBM MQ classes for JMS 라이브러리도 포함하는 이 디렉토리의 위치는 96 페이지의 『JNI(Java Native Interface) 라이브러리 구성』의 내용을 참조하십시오.
- **Windows** 설치 과정에서 Windows이 디렉터리가 IBM MQ classes for JMS시스템 경로에 자동으로 추가됩니다.

IBM MQ 자원 어댑터 구성

리소스 IBM MQ 어댑터를 구성하려면 다양한 Java Platform, Enterprise Edition Connector Architecture (JCA) 리소스를 정의하고 선택적으로 시스템 속성을 정의합니다. 설치 확인 테스트(IVT) 프로그램을 실행하도록 자원 어댑터도 구성해야 합니다. 이는 IBM 서비스가 비IBM 애플리케이션 서버가 제대로 구성되었음을 표시하기 위해 이 프로그램을 실행해야 하기 때문에 중요합니다.

시작하기 전에

이 태스크에서는 사용자가 이미 JMS와 IBM MQ classes for JMS를 잘 안다고 가정합니다. IBM MQ 자원 어댑터를 구성하는 데 사용하는 많은 특성이 IBM MQ classes for JMS 오브젝트의 특성에 해당하며 기능이 동일합니다.

이 태스크 정보

모든 애플리케이션 서버에서 고유한 관리 인터페이스 세트를 제공합니다. 일부 애플리케이션 서버는 JCA 자원을 정의하는 그래픽 사용자 인터페이스를 제공하지만 다른 애플리케이션 서버에서는 관리자가 XML 배치 계획을 작성해야 합니다. 따라서, 각 애플리케이션 서버에 대한 리소스 IBM MQ 어댑터 구성 방법에 대한 정보 제공은 본 문서의 범위를 벗어납니다.

다음 단계는 구성해야 할 사항에만 집중합니다. JCA 자원 어댑터를 구성하는 방법에 관한 정보는 애플리케이션 서버와 함께 제공되는 문서를 참조하십시오.

프로시저

다음 범주의 JCA 자원을 정의하십시오.

- ResourceAdapter 오브젝트의 특성을 정의하십시오.
진단 추적 레벨과 같이 자원 어댑터의 글로벌 특성을 표시하는 이러한 특성은 [407 페이지의 『ResourceAdapter 오브젝트 특성의 구성』](#)에서 설명합니다.
- ActivationSpec 오브젝트의 특성을 정의하십시오.
이러한 특성은 MDB가 인바운드 통신에 활성화된 방식을 판별합니다. 자세한 정보는 [409 페이지의 『인바운드 통신용 자원 어댑터 구성』](#)의 내용을 참조하십시오.
- ConnectionFactory 오브젝트의 특성을 정의하십시오.
애플리케이션 서버에서 이러한 특성을 사용하여 아웃바운드 통신에 사용할 JMS ConnectionFactory 오브젝트를 작성합니다. 자세한 정보는 [425 페이지의 『아웃바운드 통신용 자원 어댑터 구성』](#)의 내용을 참조하십시오.
- 관리 대상 목적지 오브젝트의 특성을 정의하십시오.
애플리케이션 서버에서 이러한 특성을 사용하여 아웃바운드 통신에 사용할 JMS Queue 오브젝트 또는 JMS Topic 오브젝트를 작성합니다. 자세한 정보는 [425 페이지의 『아웃바운드 통신용 자원 어댑터 구성』](#)의 내용을 참조하십시오.
- 옵션: 자원 어댑터의 배치 계획을 정의하십시오.
IBM MQ 자원 어댑터 RAR 파일은 META-INF/ra.xml 파일을 포함하며 이 xml 파일은 자원 어댑터의 배치 디스크립터를 포함합니다. 이 배치 디스크립터는 https://xmlns.jcp.org/xml/ns/javaee/connector_1_7.xsd에서 XML 스키마로 정의하며 자원 어댑터와 이 어댑터에서 제공하는 서비스에 대한 정보를 포함합니다. 애플리케이션 서버에는 자원 어댑터의 배치 계획도 필요할 수 있습니다. 이 배치 계획은 애플리케이션 서버에 고유합니다.

다음과 같이 필요한 대로 JVM 시스템 특성을 지정하십시오.

- TLS(Transport Layer Security)를 사용하는 경우 다음 예에서와 같이 키 저장소 파일 및 신뢰 저장소 파일을 JVM 시스템 특성으로 지정하십시오.

```
java ... -Djavax.net.ssl.keyStore=  
key_store_location  
-Djavax.net.ssl.trustStore=trust_store_location  
-Djavax.net.ssl.keyStorePassword=key_store_password
```

이러한 특성은 ActivationSpec 또는 ConnectionFactory 오브젝트의 특성일 수 없으며 애플리케이션 서버의 키 저장소를 두 개 이상 지정할 수 없습니다. 이 특성은 전체 JVM에 적용되므로 애플리케이션 서버에서 실행 중인 다른 애플리케이션이 TLS 연결을 사용 중인 경우 애플리케이션 서버에 영향을 미칠 수 있습니다. 애플리케이션 서버는 이러한 특성을 다른 값으로도 설정할 수 있습니다. IBM MQ classes for JMS와 TLS를 사용하는 데 관한 자세한 정보는 [243 페이지의 『IBM MQ classes for JMS에서 TLS 사용』](#)의 내용을 참조하십시오.

- 옵션: 필요한 경우 애플리케이션 서버의 표준 출력 로그에 경로 메시지를 로깅하도록 자원 어댑터를 구성하십시오.

자원 어댑터 로그, 경로 및 오류 메시지에서 IBM MQ classes for JMS와 동일한 메커니즘을 사용합니다. 자세한 내용은 [IBM MQ classes for JMS의 오류 로깅을 참조하십시오](#). 즉, 메시지는 기본적으로 mqjms.log라는 파일로 이동합니다. 애플리케이션 서버의 표준 출력 로그에 경고 메시지를 추가로 로깅하도록 자원 어댑터를 구성하려면 애플리케이션 서버의 다음 JVM 시스템 특성을 설정하십시오.

```
-Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
```

IBM MQ classes for JMS의 추적을 제어하는 데 사용하는 특성과 동일합니다. IBM MQ classes for JMS에서와 같이, jms.config 파일을 가리키는 시스템 특성을 사용할 수 있습니다(98 페이지의 『[Jakarta Messaging 구성 IBM MQ classes for JMS 파일](#)』 참조). JVM 시스템 특성을 설정하는 방법에 대한 정보는 애플리케이션 서버 문서를 참조하십시오.

설치 확인 테스트를 실행하도록 자원 어댑터 구성

- IBM MQ 자원 어댑터와 함께 제공되는 설치 확인 테스트(IVT) 프로그램을 실행하도록 자원 어댑터를 구성하십시오.

IVT 프로그램을 실행하기 위해 구성해야 하는 항목에 대한 정보는 442 페이지의 『[리소스 어댑터 IBM MQ 설치 확인](#)』의 내용을 참조하십시오.

이는 IBM 서비스가 비IBM 애플리케이션 서버가 제대로 구성되었음을 표시하기 위해 이 프로그램을 실행해야 하기 때문에 중요합니다.

중요사항: 프로그램을 실행할 수 있으려면 자원 어댑터를 구성해야 합니다.

ResourceAdapter 오브젝트 특성의 구성

ResourceAdapter 오브젝트는 IBM MQ 자원 어댑터의 글로벌 특성(예: 진단 추적 레벨)을 캡슐화합니다. 이러한 특성을 정의하려면 애플리케이션 서버에서 제공하는 문서에 설명된 대로 자원 어댑터의 기능을 사용하십시오.

ResourceAdapter 오브젝트에는 두 세트의 특성이 있습니다.

- 진단 추적과 연관된 특성
- 자원 어댑터가 관리하는 연결 풀과 연관된 특성

이러한 특성을 정의하는 방식은 애플리케이션 서버가 제공하는 관리 인터페이스에 따라 다릅니다. WebSphere Application Server를 사용하는 경우에는 408 페이지의 『[WebSphere Application Server 구성](#)』의 내용을 참조하고, WebSphere Liberty를 사용하는 경우에는 409 페이지의 『[WebSphere Liberty 구성](#)』의 내용을 참조하십시오. 다른 애플리케이션 서버의 경우 애플리케이션 서버의 제품 문서를 참조하십시오.

진단 추적과 연관된 특성 정의에 대한 자세한 정보는 [IBM MQ 자원 어댑터 추적](#)을 참조하십시오.

자원 어댑터는 MDB에 메시지를 전달하는 데 사용하는 JMS 연결의 내부 연결 풀을 관리합니다. 407 페이지의 표 62에는 연결 풀과 연관된 ResourceAdapter 오브젝트의 특성이 나열되어 있습니다.

표 62. 연결 풀과 연관된 ResourceAdapter 오브젝트의 특성			
특성 이름	유형	기본값	설명
maxConnections	문자열	50	IBM MQ 큐 관리자의 최대 연결 수 및 배치된 최대 MDB 수입니다.
connectionConcurrency	문자열	1	JMS 연결을 공유할 MDB의 최대 수입니다. 연결 공유는 불가능하며 이 특성의 값은 항상 1입니다.
reconnectionRetryCount	문자열	5	연결에 실패하는 경우 자원 어댑터가 IBM MQ 큐 관리자에 다시 연결하는 최대 시도 수입니다.
reconnectionRetryInterval	문자열	300 000	IBM MQ 큐 관리자에 다시 연결하기 전에 자원 어댑터가 대기하는 시간(밀리초)입니다.
startupRetryCount	문자열	0	애플리케이션 서버가 시작될 때 큐 관리자가 실행 중이지 않은 경우 시동 시 MDB에 연결하려고 시도하는 기본 횟수입니다.

표 62. 연결 풀과 연관된 ResourceAdapter 오브젝트의 특성 (계속)

특성 이름	유형	기본값	설명
startupRetryInterval	문자열	30 000	시동 연결 시도 간의 기본 휴면 시간(밀리초)입니다.
supportMQExtensions	문자열	false	IBM MQ JMS 동작을 JMS 2.0 전 동작으로 되돌립니다. 자세한 정보는 302 페이지의 『SupportMQExtensions 특성』 의 내용을 참조하십시오.
nativeLibraryPath	문자열	<empty>	바인딩 모드 연결을 허용하도록 IBM MQ JNI 라이브러리를 로드하는 데 사용하는 경로입니다. Windows Windows에서 시스템 경로에는 일치하는 IBM MQ 설치의 위치도 포함되어야 합니다.

maxConnection 특성에 지정된 최대 연결 수가 초과되지 않는 경우, MDB가 애플리케이션에 배치되면 새로운 JMS 연결이 작성되고 큐 관리자와의 대화가 시작됩니다. 그러므로 MDB의 최대 수는 연결의 최대 수와 같습니다. 배치된 MDB의 수가 이 최대값에 도달하면 다른 MDB를 배치하려는 시도는 실패합니다. 한 MDB가 중지되면 해당 연결을 다른 MDB에 사용될 수 있습니다.

일반적으로 많은 MDB를 배치하려는 경우에는 maxConnections 특성의 값을 늘려야 합니다.

예를 들어 네트워크 실패로 인해 IBM MQ 큐 관리자에 대한 연결이 실패하면 reconnectionRetryCount와 reconnectionRetryInterval 특성을 통해 자원 어댑터의 동작을 제어합니다. 연결이 실패하면 자원 어댑터는 reconnectionRetryInterval 특성이 지정하는 간격 동안 해당 연결이 제공한 모든 MDB로의 메시지 전달을 일시 중단합니다. 그런 다음 자원 어댑터는 큐 관리자에 다시 연결하려고 시도합니다. 이 시도가 실패하면 자원 어댑터는 reconnectionRetryCount 특성으로 지정된 한계에 도달할 때까지 reconnectionRetryInterval 특성으로 지정된 간격으로 다시 연결하려고 추가로 시도합니다. 모든 시도가 실패하면 전달은 MDB를 수동으로 다시 시작할 때까지 영구적으로 중지됩니다.

일반적으로 ResourceAdapter 오브젝트는 관리할 필요가 없습니다. 그러나 예를 들어, AIX and Linux 시스템에서 진단 추적을 사용으로 설정하기 위해 다음 특성을 설정할 수 있습니다.

```
traceEnabled: true
traceLevel: 10
```

이러한 특성은 자원 어댑터가 시작되지 않은 경우, 즉 예를 들어 IBM MQ 자원을 사용하는 애플리케이션이 클라이언트 컨테이너에서만 실행 중인 경우 적용되지 않습니다. 이 경우 진단 추적의 특성을 JVM(Java Virtual Machine) 시스템 특성으로 설정할 수 있습니다. 다음 예와 같이 **java** 명령에 **-D** 플래그를 사용하여 특성을 설정할 수 있습니다.

```
java ... -DtraceEnabled=true -DtraceLevel=6
```

ResourceAdapter 오브젝트의 모든 특성을 정의할 필요는 없습니다. 지정되지 않고 남아 있는 모든 특성은 기본값이 사용됩니다. 관리되는 환경에서는 특성을 지정하는 두 가지 방법을 혼용하지 않는 것이 더 좋습니다. 혼용하는 경우 JVM 시스템 특성이 ResourceAdapter 오브젝트의 특성보다 우선순위가 높습니다.

WebSphere Application Server 구성

동일한 속성은 리소스 어댑터에서도 사용할 수 WebSphere Application Server 있지만, 해당 속성은 리소스 어댑터의 속성 패널 내에서 설정해야 합니다(제품 WebSphere Application Server 설명서의 [JMS 공급자 설정](#)을 참조하십시오). 추적은 WebSphere Application Server 구성의 진단 섹션에서 제어합니다. 자세한 내용은 제품 WebSphere Application Server 설명서의 ['진단 공급자 작업'](#)을 참조하십시오.

WebSphere Liberty 구성

자원 어댑터는 다음 예에 표시된 대로 `server.xml` 파일에서 XML 요소를 사용하여 구성됩니다.

```
JM 3.0
<featureManager>
...
  <feature>messaging-3.0</feature>
...
</featureManager>
  <variable name="wmqJmsClient.rar.location"
    value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jakarta.jmsra.rar"/>
...
  <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

```
JMS 2.0
<featureManager>
...
  <feature>wmqJmsClient-2.0</feature>
...
</featureManager>
  <variable name="wmqJmsClient.rar.location"
    value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jmsra.rar"/>
...
  <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

이 XML 요소를 추가하여 추적을 사용하도록 설정합니다.

```
<logging traceSpecification="JMSApi=all:WAS.j2c=all:"/>
```

인바운드 통신용 자원 어댑터 구성

인바운드 통신을 구성하려면 하나 이상의 `ActivationSpec` 오브젝트에 대한 특성을 정의하십시오.

`ActivationSpec` 객체의 속성은 메시지 주도 빈(MDB)이 메시지 IBM MQ 큐로부터 메시지를 JMS 수신하는 방식을 결정합니다. MDB의 이 트랜잭션 동작은 배치 디스크립터에 정의되어 있습니다.

`ActivationSpec` 오브젝트에는 다음 두 개의 특성 세트가 있습니다.

- 큐 IBM MQ 매니저에 대한 연결을 JMS 생성하는 데 사용되는 속성들
- 지정된 큐에 도착할 때 비동기식으로 메시지를 전달하는 JMS 연결 이용자를 작성하는 데 사용하는 특성

`ActivationSpec` 오브젝트의 특성을 정의하는 방식은 애플리케이션 서버에서 제공한 관리 인터페이스에 따라 달라집니다.

큐 IBM MQ 매니저에 연결을 JMS 생성하는 데 사용되는 속성

409 페이지의 표 63의 모든 특성은 선택사항입니다.

표 63. JMS 연결을 작성하는 데 사용하는 <i>ActivationSpec</i> 오브젝트의 특성			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
applicationName	문자열	• 사용 가능한 경우 28자 이하로 조정되는 호출 클래스 이름입니다. 사용할 수 없으면 WebSphere MQ Client for Java 문자열이 사용됩니다.	애플리케이션이 큐 관리자에 등록되는 이름입니다. 이 애플리케이션 이름은 DISPLAY CONN MQSC/PCF 명령어(해당 필드명: APPLTAG) 또는 애플리케이션 IBM MQ Explorer 연결 화면(해당 필드명: App name)에서 표시됩니다.

표 63. JMS 연결을 작성하는 데 사용하는 ActivationSpec 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
brokerCCDurSubQueue ¹	문자열	• SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE • 큐 이름	연결 이용자가 지속 가능한 구독 메시지를 수신하는 큐의 이름입니다.
brokerCCSubQueue ¹	문자열	• SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE • 큐 이름	연결 이용자가 지속 불가능 구독 메시지를 수신하는 큐의 이름입니다.
brokerControlQueue ¹	문자열	• SYSTEM.BROKER.CONTROL.QUEUE • 큐 이름	브로커 제어 큐의 이름입니다.
brokerQueueManager ¹	문자열	• ""(빈 문자열) • 큐 관리자 이름	브로커가 실행 중인 큐 관리자의 이름입니다.
brokerSubQueue ¹	문자열	• SYSTEM.JMS.ND.SUBSCRIBER.QUEUE • 큐 이름	지속 불가능한 메시지 이용자가 메시지를 수신하는 큐의 이름입니다.
brokerVersion ¹	문자열	• 지정되지 않음 - 브로커를 V6에서 V7로 마이그레이션한 후 RFH2 헤더를 더 이상 사용하지 않도록 이 특성을 설정하십시오. 마이그레이션하고 나면 이 특성은 더 이상 적절하지 않습니다. • V1 - IBM MQ 발행/구독 브로커를 사용합니다. TRANSPORT가 BIND 또는 CLIENT로 설정된 경우 이 값이 기본값입니다. • V2 - 고유 모드에서 IBM Integration Bus의 브로커를 사용합니다. TRANSPORT가 DIRECT 또는 DIRECTHTTP로 설정된 경우 이 값이 기본값입니다.	사용되는 브로커의 버전입니다.
ccdtURL	문자열	• 널 • URL(Uniform Resource Locator)	클라이언트 채널 정의 표 (CCDT)를 포함하는 파일의 이름과 위치를 식별하고 파일에 액세스할 수 있는 방법을 지정하는 URL입니다.
CCSID	문자열	• 819 • JVM(Java Virtual Machine)에서 지원되는 코드화된 문자 세트 ID	연결의 코드화된 문자 세트 ID입니다.
채널	문자열	• SYSTEM.DEF.SVRCONN • MQI 채널의 이름	사용할 MQI 채널의 이름입니다.
cleanupInterval ¹	int	• 3 600 000 • 양수	발행/구독 정리 유틸리티의 백그라운드 실행 간 간격(밀리초)입니다.

표 63. JMS 연결을 작성하는 데 사용하는 <i>ActivationSpec</i> 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
cleanupLevel ¹	문자열	• SAFE • NONE • STRONG • FORCE • NONDUR	브로커 기반 구독 저장소에 대한 정리 레벨입니다.
clientID	문자열	• 널 • 클라이언트 ID	연결에 사용하는 클라이언트 ID입니다.
cloneSupport	문자열	• DISABLED - 지속 가능한 토픽 구독자의 인스턴스는 한 번에 하나만 실행할 수 있습니다. • ENABLED - 지속 가능한 동일 토픽 구독자의 인스턴스를 동시에 두 개 이상 실행할 수 있지만 각 인스턴스는 개별 JVM(Java Virtual Machine)에서 실행해야 합니다.	동일한 지속 가능 토픽 구독자의 인스턴스를 동시에 두 개 이상 실행할 수 있는지 여부입니다.
connectionFactoryLookup	문자열	• 널 • ConnectionFactory 오브젝트의 JNDI 이름	이 속성이 설정되면 <i>ActivationSpec</i> 다음을 조회합니다. JMS 애플리케이션 서버의 JNDI 네임스페이스에 지정된 JNDI 이름을 가진 <i>ConnectionFactory</i> 객체를 생성한 다음, 해당 객체의 속성을 사용하여 생성합니다. JMS 연결 IBM MQ 큐 관리자, 단 한 가지 예외를 제외하고, JMS 연결을 작성할 때 사용될 <i>ActivationSpec</i> 의 특성은 <i>clientID</i> 뿐입니다. 자세한 정보는 421 페이지의 『ActivationSpec connectionFactoryLookup 및 destinationLookup 특성』 의 내용을 참조하십시오.

표 63. JMS 연결을 작성하는 데 사용하는 <i>ActivationSpec</i> 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
connectionNameList	문자열	• localhost(1414) • 쉼표로 분리되는 항목으로 구성된 문자열입니다. 각 항목의 형식은 다음과 같습니다. HOSTNAME(PORT) 여기서 <i>HOSTNAME</i>은 DNS 이름이거나 IP 주소입니다.	인바운드 통신에 사용된 TCP/IP 연결 이름의 목록입니다. 지정된 경우 connectionNameList는 hostname 및 port 특성을 대신합니다. 이 특성은 다중 인스턴스 큐 관리자에 다시 연결하는 데 사용됩니다. connectionNameList는 localAddress와 양식이 비슷하므로 혼동하지 않아야 합니다. localAddress는 로컬 통신의 특성을 지정하는 반면 connectionNameList는 리모트 큐 관리자에 도달하는 방식을 지정합니다.
dynamicallyBalanced ⁴	부울	• false • true	이 MDB에게 단일 양식 클러스터에서 애플리케이션 밸런싱의 일부로서 다른 큐 관리자로부터 메시지를 수신하도록 요청할 수 있는지 여부
failIfQuiesce	부울	• true • false	큐 관리자가 정지(quiescing) 상태에 있는 경우 특정 메소드에 대한 호출이 실패하는지 여부입니다.
headerCompression	문자열	• NONE • SYSTEM - RLE 메시지 헤더 압축이 수행됩니다.	연결에서 헤더 데이터를 압축하는 데 사용할 수 있는 기술 목록입니다.
hostName	문자열	• localhost • 호스트 이름 • IP 주소	큐 관리자가 상주하는 시스템의 호스트 이름 또는 IP 주소입니다. hostname 및 port 특성은 connectionNameList 특성이 지정된 경우 이 특성으로 대체됩니다.

표 63. JMS 연결을 작성하는 데 사용하는 ActivationSpec 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
localAddress	문자열	• 널 • 다음 형식의 문자열입니다. <pre>[host_name][(low_port [, high_port])]</pre> 여기서 <i>host_name</i>은 호스트 이름 또는 IP 주소이고, <i>low_port</i> 및 <i>high_port</i>는 TCP 포트 번호이며, 대괄호는 선택적 컴포넌트를 나타냅니다.	큐 관리자에 대한 연결에서 이 특성은 다음 중 하나 또는 둘 다를 지정합니다. • 사용할 로컬 네트워크 인터페이스 • 사용할 로컬 포트 또는 로컬 포트의 범위 localAddress는 connectionNameList와 양식이 비슷하므로 혼동하지 않아야 합니다. localAddress는 로컬 통신의 특성을 지정하는 반면 connectionNameList는 리모트 큐 관리자에 도달하는 방식을 지정합니다.
messageCompression	문자열	• NONE • 하나 이상의 다음 값이 공백 문자로 분리된 목록입니다. RLE ZLIBFAST ZLIBHIGH LZ4FAST LZ4HIGH	연결에서 메시지 데이터를 압축하는 데 사용할 수 있는 기술 목록입니다.
messageRetention ¹	부울	• true - 원하지 않는 메시지는 입력 큐에 남아 있습니다. • false - 원하지 않는 메시지는 해당 처리 옵션에 따라 처리됩니다.	연결 이용자가 입력 큐에 원하지 않는 메시지를 보존하는지 여부
messageSelection ¹	문자열	• CLIENT • BROKER	메시지 선택을 IBM MQ classes for JMS에서 수행하는지 아니면 브로커에서 수행하는지 판별합니다. brokerVersion의 값이 1이면 브로커에서 메시지를 선택하는 기능은 지원되지 않습니다.
비밀번호	문자열	• 널 • 비밀번호	큐 관리자에 대한 연결을 작성할 때 사용할 기본 비밀번호입니다.

표 63. JMS 연결을 작성하는 데 사용하는 <i>ActivationSpec</i> 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
pollingInterval ¹	int	• 5000 • 양수	세션 내에 있는 각 메시지 리스너의 큐에 적합한 메시지가 없는 경우, 이 값은 각 메시지 리스너가 해당 큐에서 메시지가져오기를 다시 시도하기 전에 경과하는 최대 간격(밀리초)입니다. 세션의 메시지 리스너에 사용 가능한 적합한 메시지가 없는 경우가 자주 발생하면 이 특성 값을 높여 보십시오. 이 특성은 TRANSPORT의 값이 BIND이거나 CLIENT인 경우에만 해당합니다.
포트	int	• 1414 • TCP 포트 번호	큐 관리자가 청취하는 포트입니다. hostname 및 port 특성은 connectionNameList 특성이 지정된 경우 이 특성으로 대체됩니다.
providerVersion	문자열	• 지정되지 않음 • 다음 형식 중 하나의 문자열 - V.R.M.F - V.R.M - V.R - V 여기서 V, R, M 및 F는 0 이상의 정수 값입니다.	MDB에서 연결하려는 큐 관리자의 버전, 릴리스, 수정 레벨 및 수정팩입니다.
queueManager	문자열	• ""(빈 문자열) • 큐 관리자 이름	연결할 큐 관리자의 이름입니다.
receiveExit ³	문자열	• 널 • 쉼표로 분리된 하나 이상의 항목으로 구성되는 문자열. 여기서 각 항목은 IBM MQ classes for Java 인터페이스인 MQReceiveExit를 구현하는 클래스의 완전한 이름입니다.	채널 수신 엑시트 프로그램이나 그 뒤를 이어 실행할 수신 엑시트 프로그램의 시퀀스를 식별합니다.
receiveExitInit	문자열	• 널 • 쉼표로 분리된 하나 이상의 사용자 데이터 항목으로 구성된 문자열	채널 수신 엑시트 프로그램 호출 시 이 프로그램에 전달되는 사용자 데이터입니다.

표 63. JMS 연결을 작성하는 데 사용하는 <i>ActivationSpec</i> 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
rescanInterval ¹	int	• 5000 • 양수	포인트 투 포인트 도메인 내 메시지 수신자가 메시지 선택기를 사용하여 수신할 메시지를 선택할 때, 해당 큐의 속성에 MsgDeliverySequence 의해 결정된 순서에 IBM MQ classes for JMS 따라 적합한 메시지를 큐에서 IBM MQ 검색합니다. IBM MQ classes for JMS에서 적당한 메시지를 찾아 이용자에게 전달할 때 IBM MQ classes for JMS가 큐의 현재 위치에서 적절한 다음 메시지의 검색을 재개합니다. IBM MQ classes for JMS가 큐의 끝에 도달하거나 이 특성 값으로 판별된 시간 간격(밀리초)이 만기될 때까지 이 방식으로 계속 큐를 검색합니다. 각각의 경우 IBM MQ classes for JMS가 검색을 계속하기 위해 큐의 시작으로 돌아가고 새로운 시간 간격이 시작됩니다.
securityExit ³	문자열	• 널 • IBM MQ classes for Java 인터페이스인 MQSecurityExit를 구현하는 클래스의 완전한 이름	채널 보안 엑시트 프로그램을 식별합니다.
securityExitInit	문자열	• 널 • 사용자 데이터 문자열	채널 보안 엑시트 프로그램 호출 시 이 프로그램에 전달되는 사용자 데이터입니다.
sendExit ³	문자열	• 널 • 심표로 분리된 하나 이상의 항목으로 구성되는 문자열. 여기서 각 항목은 IBM MQ classes for Java 인터페이스인 MQSendExit를 구현하는 클래스의 완전한 이름입니다.	채널 송신 엑시트 프로그램이나 그 뒤를 이어 실행할 송신 엑시트 프로그램의 시퀀스를 식별합니다.
sendExitInit	문자열	• 널 • 심표로 분리된 하나 이상의 사용자 데이터 항목으로 구성된 문자열	채널 송신 엑시트 프로그램 호출 시 이 프로그램에 전달되는 사용자 데이터입니다.
shareConvAllowed	부울	• 아니요 - 클라이언트 연결은 소켓을 공유할 수 없습니다. • 예 - 클라이언트 연결은 소켓을 공유할 수 있습니다.	채널 정의가 일치하는 경우 클라이언트 연결을 통해 동일한 프로세스부터 동일한 큐 관리자까지 기타 최상위 레벨 JMS 연결과 해당 소켓을 공유할 수 있는지 여부입니다.

표 63. JMS 연결을 작성하는 데 사용하는 ActivationSpec 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
sparseSubscriptions ¹	부울	• false - 구독에서 자주 일치하는 메시지를 수신합니다. • true - 구독에서 드물게 일치하는 메시지를 수신합니다. 이 값을 사용하려면 구독 큐를 읽기 전용으로 열 수 있어야 합니다.	TopicSubscriber 오브젝트의 메시지 검색 정책을 제어합니다.
sslCertStores	문자열	• 널 • 공백으로 분리된 하나 이상의 LDAP URL 문자열입니다. 각 LDAP URL의 형식은 다음과 같습니다. <pre>ldap://host_name [: port]</pre> 여기서 <i>host_name</i>은 호스트 이름 또는 IP 주소이고, <i>port</i>는 TCP 포트 번호이며 대괄호는 선택적 컴포넌트를 나타냅니다.	TLS 연결에서 사용할 인증서 폐기 목록(CRL)을 보유하는 LDAP(Lightweight Directory Access Protocol) 서버입니다.
sslCipherSuite	문자열	• 널 • CipherSuite의 이름	TLS 연결에 사용할 CipherSuite입니다.
sslFipsRequired ²	부울	• false • true	TLS 연결이 JSSE FIPSIBMJava 공급자 (IBMJSSEFIPS)에서 지원하는 CipherSuite를 사용해야 하는지 여부
sslPeerName	문자열	• 널 • 식별 이름의 템플릿	TLS 연결의 경우 큐 관리자가 제공한 디지털 인증서에서 식별 이름을 검사하는 데 사용하는 템플릿입니다.
sslResetCount	int	• 0 • 0 - 999 999 999 범위의 정수	TLS에서 사용하는 비밀 키를 재협상하기 전에 TLS 연결에서 보내고 받은 바이트 수의 합계입니다.
sslSocketFactory	문자열	javax.net.ssl.SSLSocketFactory 인터페이스의 구현을 제공하는 클래스의 완전한 클래스 이름을 나타내는 문자열입니다. 선택적으로 구성자 메소드에 전달될 괄호로 묶은 인수를 포함합니다.	관리 대상 오브젝트 범위에서 설정된 모든 연결이 이 SSLSocketFactory 인터페이스 구현에서 얻은 소켓을 사용합니다.
statusRefreshInterval ¹	int	• 60000 • 양수	구독자와 큐 관리자의 연결이 끊길 때 발견한 장기 실행 트랜잭션의 새로 고치기 간격 (밀리초)입니다. 이 특성은 subscriptionStore 에 QUEUE 값이 있는 경우에만 관련이 있습니다.
subscriptionStore ¹	문자열	• 브로커 • MIGRATE • 큐	IBM MQ classes for JMS가 활성 구독에 관한 지속적 데이터를 저장하는 위치를 판별합니다.

표 63. JMS 연결을 작성하는 데 사용하는 <i>ActivationSpec</i> 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
transportType	문자열	• CLIENT • BINDINGS • BINDINGS_THEN_CLIENT	큐 관리자에 연결할 때 사용할 모드. • BINDINGS : 로컬 연결이 속성으로 queueManager 지정된 큐 관리자에 생성됩니다. 그러나 클라이언트 모드 ccdtURL 속성도 지정하는 경우, 해당 ccdtURL 속성이 우선 적용됩니다(이전 버전과의 호환성을 IBM MQ 위해). 리소스 어댑터는 대신 CCDT 파일에 지정된 큐 관리자에 클라이언트 모드 연결을 수행합니다. • 클라이언트 : 큐 매니저와의 연결은, hostname 및 port, 또는 ccdtURL 속성에 의해 connectionNameList 제어됩니다. • BINDINGS_THEN_CLIENT : 리소스 어댑터가 바인딩 모드에서 연결을 시도합니다. 이 연결 시도가 실패하면 리소스 어댑터는 클라이언트 모드로 연결을 시도합니다. 해당 ccdtURL 속성은 이 모드에서 지원되지 않습니다.
transportType (계속)	문자열	• CLIENT • BINDINGS • BINDINGS_THEN_CLIENT	 WebSphere Application Server for z/OS 시스템에서 실행 중인 활성화 스펙이 BINDINGS_THEN_CLIENT 전송 모드를 사용하도록 구성되고 이전에 설정된 연결은 끊어진 경우, 활성화 스펙에 의한 재연결 시도에서는 먼저 BINDINGS 전송 모드를 사용하려 합니다. BINDINGS 전송 모드 연결 시도가 실패하면, 활성화 스펙은 연속적으로 CLIENT 전송 모드 연결을 시도합니다.
username	문자열	• 널 • 사용자 이름	큐 관리자에 대한 연결을 작성할 때 사용할 기본 사용자 이름입니다.

표 63. JMS 연결을 작성하는 데 사용하는 <i>ActivationSpec</i> 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
wildcardFormat	문자열	- CHAR- 브로커 버전 1에서 사용된 대로 문자 와일드카드만 인식 TOPIC - 브로커 버전 2에서 사용된 대로 토픽 레벨 와일드카드만 인식	사용할 와일드카드 구문 버전입니다.

참고:

1. 이 특성은 IBM MQ classes for JMS의 버전 70에서 사용할 수 있습니다.
2. sslFipsRequired 특성을 사용하는 데 관한 중요한 정보는 390 페이지의 『리소스 IBM MQ 어댑터의 한계 및 IBM MQ 리소스 어댑터에 대한 Jakarta Messaging』을 참조하십시오.
3. 엑시트를 찾을 수 있도록 자원 어댑터를 구성하는 방법에 대한 정보는 260 페이지의 『채널 엑시트를 사용하도록 IBM MQ classes for JMS 구성』의 내용을 참조하십시오.
4. dynamicallyBalanced 특성은 XA 트랜잭션 지원과 결합하여 지원되지 않습니다. dynamicallyBalanced가 "true"이면, XA 트랜잭션이 사용 안함으로 설정되도록 MDB를 구성해야 합니다.

JMS 연결 이용자를 작성하는 데 사용한 특성

참고: **destination** 및 **destinationType**은 명시적으로 정의해야 합니다. 418 페이지의 표 64의 다른 모든 특성은 선택적입니다.

표 64. JMS 연결 이용자를 작성하는 데 사용하는 <i>ActivationSpec</i> 오브젝트의 특성			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
destination	문자열	목적지 이름	메시지를 수신할 목적지입니다. useJNDI 특성은 이 특성값이 해석되는 방법을 판별합니다.
destinationLookup	문자열	널 - Destination 오브젝트의 JNDI 이름	이 특성이 설정되면 <i>ActivationSpec</i> 은 애플리케이션 서버의 JNDI 네임스페이스에서 지정된 JNDI 이름으로 JMS Destination 오브젝트를 검색한 후 <i>ActivationSpec</i> 에 지정된 다른 특성보다 우선하여 해당 오브젝트의 특성을 사용하여 JMS 연결 이용자를 작성합니다. 자세한 정보는 421 페이지의 『 <i>ActivationSpec</i> <i>connectionFactoryLookup</i> 및 <i>destinationLookup</i> 특성』의 내용을 참조하십시오.
destinationType	문자열	- jakarta.jms.Queue (Jakarta Messaging 3.0) - jakarta.jms.Topic (Jakarta Messaging 3.0) - javax.jms.Queue (JMS 2.0) - javax.jms.Topic (JMS 2.0)	목적지, 큐 또는 토픽의 유형입니다.
maxMessages	int	1 - 양수	동시에 서버 세션에 지정할 수 있는 최대 메시지 수입니다. 활성화 스펙이 XA 트랜잭션으로 MDB에 메시지를 전달하는 경우 이 특성의 설정에 상관없이 값 1을 사용합니다.

표 64. JMS 연결 이용자를 작성하는 데 사용하는 *ActivationSpec* 오브젝트의 특성 (계속)

특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
maxPoolDepth	int	• 10 • 양수	연결 이용자가 사용하는 서버 세션 풀의 최대 서버 세션 수입니다.
messageSelector	문자열	• 널 • SQL92 메시지 선택자 표현식	전달할 메시지를 지정하는 메시지 선택자 표현식입니다.
nonASFTimeout	int	• 0 • 양수	양수 값은 비ASF 전달이 사용됨을 나타냅니다. 이 값은 가져오기 요청에서 아직 도착하지 않은 메시지를 기다리는(대기 호출을 포함하는 가져오기) 시간(밀리초)입니다. 기본값인 0은 ASF 전달이 사용됨을 나타냅니다. 이 매개변수는 다음 조건에서 유효합니다: • 응용 프로그램은 이상에서 WebSphere Application Server 7.0 실행 중입니다. • 애플리케이션은 적절한 수준의 <code>wmqJmsClient</code> 기능을 사용하여 실행 WebSphere Liberty 중입니다. 자세한 내용은 을 참조하십시오 391 페이지의 『WebSphere Liberty 그리고 IBM MQ 리소스 어댑터』.
nonASFRollbackEnabled	부울	• false - MDB가 실패하는 경우에도 메시지를 이용합니다. • true - MDB 내 실패로 인해 메시지가 큐에 롤백됩니다.	MDB를 트랜잭션할 수 없는 경우 메시지 전달이 IBM MQ 동기점에 있는지 여부입니다. MDB가 트랜잭션되거나 nonASFTimeout이 0으로 설정되는 경우 무시됩니다.
poolTimeout	int	• 300000 • 양수	사용되지 않는 서버 세션을 비활성으로 인해 종료하기 전에 서버 세션 풀에서 열린 상태로 둘 시간(밀리초)입니다.
readAheadAllowed	int	• DESTINATION - 큐 또는 토픽 정의를 참조하여 미리 읽기가 허용되는지 판별합니다. • DISABLED - 미리 읽기가 허용되지 않습니다. • ENABLED - 미리 읽기가 허용됩니다. • QUEUE - 큐 정의를 참조하여 미리 읽기가 허용되는지 판별합니다. • TOPIC - 토픽 정의를 참조하여 미리 읽기가 허용되는지 판별합니다.	파괴적인 이용을 위해 서버 세션으로 전달하기 전에 활성화 스펙 찾아보기 스프레드가 대상에서 내부 버퍼로 다중 메시지를 찾아보기 위해 미리 읽기를 사용할 수 있는지 여부입니다. 참고: 미리 읽기를 사용하면 JMSCC0108 메시지가 증가하거나 성능이 저하됩니다. MDB 처리 속도가 대상에서 메시지를 찾아보는 속도를 따라가지 못할 경우 둘 다 발생할 수 있습니다.

표 64. JMS 연결 이용자를 작성하는 데 사용하는 *ActivationSpec* 오브젝트의 특성 (계속)

특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
readAheadClosePolicy	int	• ALL - MDB를 중지하기 전에 내부 미리 읽기 버퍼에 있는 모든 메시지를 MDB에 전달합니다. • CURRENT - 현재 MDB 호출만 완료되고, 잠재적으로 내부 미리 읽기 버퍼의 메시지는 그대로 두므로 해당 메시지가 나중에 제거됩니다.	관리자가 MDB를 중지할 때 내부 미리 읽기 버퍼의 메시지에 발생하는 사항입니다.
receiveCCSID	int	• 0 - JVM <code>Charset.defaultCharset</code> 사용 • 1208 - UTF-8 • 지원되는 코드화 문자 세트 ID	큐 관리자 메시지 변환용 대상 CCSID를 설정하는 목적지 특성입니다. receiveConversion 이 QMGR로 설정되는 경우가 아니면 값은 무시됩니다.
receiveConversion	문자열	• CLIENT_MSG • QMGR	큐 관리자가 데이터 변환을 수행할지 판별하는 목적지 특성입니다.
sharedSubscription	부울	• False - MDB는 공유 구독으로서 구독을 열지 않아야 합니다. • 사실 - MDB는 구독을 공유 구독으로 열어야 합니다(이에 JMS 2.0 따른 규칙 참조, . Javanet 사양 JMS 2.0 참조).	공유 구독에서 MDB가 구동되는 방식을 제어합니다. 이 특성을 사용하는 방법에 대한 자세한 정보는 423 페이지의 『sharedSubscription 특성을 정의하는 방법의 예』 의 내용을 참조하십시오.
startTimeout	int	• 10 000 • 양수	메시지를 전달하는 작업이 스케줄링된 후 MDB에 메시지를 전달하는 작업을 시작해야 하는 시간(밀리초)입니다. 이 시간이 경과하면 메시지가 큐에 롤백됩니다.
subscriptionDurability	문자열	• NonDurable - 비지속적 구독은 토픽을 구독하는 MDB에 메시지를 전달하는 데 사용합니다. • Durable - 지속적 구독은 토픽을 구독하는 MDB에 메시지를 전달하는 데 사용합니다.	주제를 구독하는 MDB에 메시지를 전달하는 데 사용하는 구독(지속적 구독 또는 비지속적 구독)입니다.
subscriptionName	문자열	• ""(빈 문자열) • 구독 이름	지속 가능한 구독의 이름입니다.

표 64. JMS 연결 이용자를 작성하는 데 사용하는 ActivationSpec 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
useJNDI	부울	• false - destination이라는 특성은 IBM MQ 큐 또는 토픽의 이름으로 해석됩니다. • true - destination이라는 속성은 애플리케이션 서버의 JNDI 네임스페이스에서 다음 객체 중 하나의 이름으로 해석됩니다: - jakarta.jms.Queue (Jakarta Messaging 3.0) - jakarta.jms.Topic (Jakarta Messaging 3.0) - javax.jms.Queue (JMS 2.0) - javax.jms.Topic (JMS 2.0)	Deprecated destination이라는 특성의 값을 해석하는 방법을 판별합니다. 참고: 이 특성은 IBM MQ 9.0에서는 더 이상 사용되지 않습니다. 대신 destinationLookup 특성을 사용해야 합니다.

특성 충돌과 종속 항목

ActivationSpec 오브젝트의 특성이 상충할 수 있습니다. 예를 들어 바인딩 모드로 연결하기 위해 TLS 특성을 지정할 수 있습니다. 이런 경우 작동은 전송 유형 및 메시지 도메인에 의해 판별되며, 이는 **destinationType** 특성에서 판별되는 포인트-투-포인트 또는 발행/구독입니다. 지정된 전송 유형 또는 메시징 도메인에 적용할 수 없는 특성은 무시됩니다.

다른 특성을 정의해야 하는 특성은 정의하지만 이 다른 특성은 정의하지 않는 경우 MDB를 배치하는 중에 validate() 메소드를 호출할 때 ActivationSpec 오브젝트에서 InvalidPropertyException 예외가 발생합니다. 이 예외는 애플리케이션 서버에 따라 달라지는 방식으로 애플리케이션 서버의 관리자에게 보고됩니다. 예를 들어 subscriptionDurability 특성이 Durable로 설정되어 지속 가능한 구독을 사용하려 함을 표시하는 경우에는 반드시 **subscriptionName** 특성을 정의해야 합니다.

ccdtURL 및 **channel** 특성 모두가 정의되면, InvalidPropertyException 예외가 발생합니다. 그러나 **ccdtURL** 특성만 정의하고 **channel** 특성은 기본값인 SYSTEM.DEF.SVRCONN으로 두는 경우, 예외가 발생하지 않고, **ccdtURL** 특성에서 식별되는 클라이언트 채널 정의 테이블을 사용하여 JMS 연결을 시작합니다.

ActivationSpec connectionFactoryLookup 및 destinationLookup 특성

JMS 2.0 스펙에서는 두 가지 새로운 ActivationSpec 특성을 소개했습니다. connectionFactoryLookup 및 destinationLookup 특성에는 다른 ActivationSpec 특성에 우선하여 사용할 관리 대상 오브젝트의 JNDI 이름이 제공될 수 있습니다.

예를 들어, 연결 팩토리가 에 정의되어 JNDI 있고 해당 객체의 이름이 JNDI 활성화 사양의 connectionFactoryLookup 속성에 지정된 경우, 에 정의된 연결 팩토리의 모든 속성이 JNDI의 속성보다 우선적으로 409 페이지의 표 63 사용됩니다.

목적지가 에 정의되고 이름이 JNDI ActivationSpec's의 JNDIDestinationLookup 속성에 설정된 경우, 해당 418 페이지의 표 64값이의 값보다 우선적으로 사용됩니다. 이러한 두 특성을 사용하는 방법에 대한 자세한 정보는 421 페이지의 『ActivationSpec connectionFactoryLookup 및 destinationLookup 특성』의 내용을 참조하십시오.

이 두 속성은 `ActivationSpec`의 속성보다 우선하여 사용되는 `ConnectionFactory` 및 `Destination` 객체의 이름을 JNDI 지정하는 데 사용할 수 있습니다. 이는 `` 및 409 페이지의 표 63 418 페이지의 표 64``에서 정의된 대로 적용됩니다.

이러한 특성이 작동하는 방식을 자세히 설명하는 다음과 같은 사항을 참고하는 것이 중요합니다.

connectionFactoryLookup

JNDI에서 검색하는 ConnectionFactory는 409 페이지의 표 63에 나열된 특성의 소스로 사용합니다. ConnectionFactory 오브젝트는 실제로 JMS 연결을 작성하는 데 사용하지 않으며 오브젝트의 특성만 조회합니다. ConnectionFactory의 해당 특성은 ActivationSpec에 정의된 특성을 대체합니다. 여기에는 예외가 하나 있습니다. ActivationSpec에서 **ClientID** 특성이 설정되어 있으면 이 특성 값은 ConnectionFactory에서 지정된 값을 대체합니다. 이는 공용 시나리오가 여러 ActivationSpec에 단일 ConnectionFactory를 사용하기 때문입니다. 이를 통해 관리가 간소화됩니다. 그러나 JMS 2.0 스펙에서는 ConnectionFactory에서 작성된 모든 JMS 연결에 고유한 **ClientID**가 있어야 함을 명시합니다. 이러한 이유로 ActivationSpec에는 ConnectionFactory에 설정된 값을 대체할 수 있는 기능이 있어야 합니다. ActivationSpec에서 **ClientID**가 설정되지 않으면 연결 팩토리 값이 사용됩니다.

destinationLookup

ActivationSpec에서 **Destination** 및 **UseJndi** 특성이 정의되어 있습니다. **UseJndi** 플래그가 true로 설정되면, 목적지 특성에서 지정되는 텍스트가 JNDI 이름인 것으로 간주되고, 해당 JNDI 이름을 가진 목적지 오브젝트는 JNDI에서 검색됩니다.

destinationLookup 특성도 정확하게 동일한 방식으로 작동합니다. 이 특성이 설정되어 있으면 이 특성을 통해 지정된 JNDI 이름이 있는 목적지 오브젝트가 JNDI에서 검색됩니다. 이 특성은 **useJNDI** 특성보다 우선합니다.

 **useJNDI** 속성은 에서 더 이상 IBM MQ 9.0 사용되지 않습니다. 동일한 기능을 수행하는 사양 JMS 2.0 또는 이후 버전의 동등한 속성인 **destinationLookup** 속성이 있기 때문입니다.

IBM MQ classes for JMS에 동등한 특성이 없는 ActivationSpec 특성

ActivationSpec 객체의 대부분의 속성은 또는 IBM MQ classes for Jakarta Messaging IBM MQ classes for JMS 객체의 속성, 또는 IBM MQ classes for JMSIBM MQ classes for Jakarta Messaging 메서드의 매개변수와 동등합니다. 그러나 세 가지 조정 속성과 하나의 사용성 속성은 또는 IBM MQ classes for JMS에 대응하는 IBM MQ classes for Jakarta Messaging항목이 없습니다:

startTimeout

자원 어댑터가 MDB에 메시지를 전달하도록 Work 오브젝트를 스케줄링한 다음 자원이 사용 가능해질 때까지 애플리케이션 서버의 작업 관리자가 대기하는 시간(밀리초)입니다. 메시지 전달을 시작하기 전에 이 시간이 경과되면 Work 오브젝트의 제한시간이 초과되고 메시지가 큐에 롤백된 다음 자원 어댑터가 메시지를 다시 전달하려고 시도할 수 있습니다. 사용되는 경우 경고가 진단 추적에 기록되지만 메시지 전달 프로세스에서는 영향을 미치지 않습니다. 애플리케이션 서버의 로드가 매우 높은 경우에만 가끔 이 상황이 발생할 수 있습니다. 이 상황이 정기적으로 발생하면 작업 관리자가 더 오래 메시지 전달을 스케줄링할 수 있도록 이 특성의 값을 늘려 보십시오.

maxPoolDepth

연결 이용자가 사용하는 서버 세션 풀의 최대 서버 세션 수입입니다. 서버 세션이 작성되면 큐 관리자와의 대화를 시작합니다. 연결 이용자가 서버 세션을 사용하여 MDB에 메시지를 전달합니다. 풀이 더 깊으면 볼륨이 많은 경우 더 많은 메시지를 동시에 전달할 수 있지만 애플리케이션 서버의 자원을 더 많이 사용합니다. 많은 MDB를 배치하는 경우 애플리케이션 서버의 로드를 관리 가능한 레벨로 유지보수하기 위해 풀 깊이를 낮게 만드십시오. 각 연결 이용자는 고유 서버 세션 풀을 사용하므로, 이 특성은 모든 연결 이용자가 사용할 수 있는 전체 서버 세션 수를 정의하지 않습니다.

poolTimeout

사용되지 않는 서버 세션을 비활성으로 인해 종료하기 전에 서버 세션 풀에서 열린 상태로 둘 시간(밀리초)입니다. 메시지 워크로드가 임시로 증가하면 로드를 분배하기 위해 추가 서버 세션이 작성되지만, 메시지 워크로드가 정상으로 돌아가면 추가 서버 세션이 풀에 그대로 남아 있으며 사용되지 않습니다.

서버 세션을 사용할 때마다 시간소인이 표시됩니다. 스캐빈저 스레드가 주기적으로 이 특성으로 지정된 기간 동안 각 서버 세션이 사용되었는지 확인합니다. 서버 세션을 사용하지 않으면 서버 세션이 닫히고 서버 세션 풀에서 제거됩니다. 지정된 기간이 경과된 후 즉시 서버 세션이 닫히지 않을 수 있습니다. 이 특성은 제거하기 전에 비활성화되는 최소 기간을 나타냅니다.

useJNDI

이 특성의 설명은 418 페이지의 표 64의 내용을 참조하십시오.

MDB 배치

MDB를 배치하려면 먼저 `ActivationSpec` 오브젝트의 특성을 정의하여 MDB에 필요한 특성을 지정하십시오. 다음 예는 명시적으로 정의할 수 있는 일반적인 특성 집합입니다.

JM 3.0

```
channel:          SYSTEM.DEF.SVRCONN
destination:      SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:  jakarta.jms.Queue
hostName:         192.168.0.42
messageSelector:  color='red'
port:            1414
queueManager:     ExampleQM
transportType:    CLIENT
```

JMS 2.0

```
channel:          SYSTEM.DEF.SVRCONN
destination:      SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:  javax.jms.Queue
hostName:         192.168.0.42
messageSelector:  color='red'
port:            1414
queueManager:     ExampleQM
transportType:    CLIENT
```

애플리케이션 서버에서 특성을 사용하여 `ActivationSpec` 오브젝트를 작성하면 이 오브젝트가 MDB와 연관됩니다. `ActivationSpec` 오브젝트의 특성을 통해 메시지를 MDB에 전달하는 방식을 판별합니다. MDB에 분배된 트랜잭션이 필요하지만 자원 어댑터에서 분배된 트랜잭션을 지원하지 않으면 MDB 배치에 실패합니다. 분배 트랜잭션이 지원되도록 자원 어댑터를 설치하는 방법에 대한 정보는 [398 페이지의 『리소스 IBM MQ 어댑터 설치』](#)의 내용을 참조하십시오.

두 개 이상의 MDB가 동일한 목적지에서 메시지를 받으면, 다른 MDB에서 메시지를 받을 수 있는 경우에도 하나의 MDB에서만 포인트-투-포인트 도메인에서 보낸 메시지를 받습니다. 특히 두 개의 MDB에서 서로 다른 메시지 선택자를 사용 중이며 수신되는 메시지가 두 메시지 선택자와 일치하는 경우 MDB 중 하나만 메시지를 받습니다. 메시지를 수신하도록 선택한 MDB가 정의되지 않았으므로 특정 MDB를 사용하여 메시지를 받을 수 없습니다. 발행/구독 도메인에서 송신된 메시지는 적합한 모든 MDB에서 수신합니다.

일부 환경에서 MDB에 전달된 메시지는 IBM MQ 큐에 롤백될 수 있습니다. 예를 들어 나중에 롤백되는 작업 단위에 메시지가 전달되는 경우 이와 같이 롤백이 발생할 수 있습니다. 롤백되는 메시지는 다시 전달되지만 메시지가 잘못 형식화되면 MDB가 반복적으로 실패하므로 메시지를 전달할 수 없습니다. 이러한 메시지는 변조 메시지라고 합니다. 설정을 통해 가 추가 조사를 위해 IBM MQ classes for JMS 독성 메시지를 다른 IBM MQ 큐로 자동 전송하거나 메시지를 폐기하도록 구성할 수 있습니다.

변조 메시지를 핸들링하는 방법에 대한 세부사항은 [224 페이지의 『IBM MQ classes for JMS에서 변조 메시지 핸들링』](#)의 내용을 참조하십시오.

관련 개념

[MQI 클라이언트에서 실행 시 인증된 CipherSpecs 만 FIPS 사용하도록 지정](#)

[Federal Information Processing Standards \(FIPS\)에 대해 AIX, Linux, and Windows](#)

관련 태스크

[WebSphere Application Server에서 JMS 자원 구성](#)

`sharedSubscription` 특성을 정의하는 방법의 예

WebSphere Liberty server.xml 파일 내에서 활성화 스펙의 `sharedSubscription` 특성을 정의할 수 있습니다. 또는 어노테이션을 사용하여 메시지 구동 Bean(MDB) 내에 특성을 정의할 수 있습니다.

예: Liberty server.xml 파일 내에서 정의

WebSphere Liberty server.xml 파일 내에서, 다음 예에 표시된 대로 활성화 스펙을 정의합니다. 이 예는 localhost/port 1490의 큐 관리자에 지속 가능한 공유 구독을 작성합니다.

```
<jmsActivationSpec id="SubApp/SubscribingEJB/SubscribingMDB" authDataRef="JMSConnectionAlias">
  <properties.wmqJms hostName="localhost" port="1490" maxPoolDepth="5"
    subscriptionName="MySubName">
```

```
subscriptionDurability="DURABLE" sharedSubscription="true"/>
</jmsActivationSpec>
```

예: MDB 내에 정의

다음 예에 표시된 대로 어노테이션을 사용하여 MDB 내에 sharedSubscription 특성을 정의할 수도 있습니다.

```
@ActioncationConfigProperty(propertyName = "sharedSubscription",
propertyValue = "true")
```

다음 예에서는 어노테이션 메소드를 사용하는 MDB 코드의 조각을 표시합니다.

```
JM 3.0
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
    activationConfig = {
        @ActivationConfigProperty(
            propertyName = "destinationType", propertyValue = "jakarta.jms.Topic"),
        @ActivationConfigProperty(
            propertyName = "sharedSubscription", propertyValue = "TRUE"),
        @ActivationConfigProperty(
            propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
    },
    mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

    // Default constructor.
    public SubscribingMDB() {
    }

    // @see MessageListener#onMessage(Message)
    public void onMessage(Message message) {
        // implement business logic here
    }
}
```

```
JMS 2.0
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
    activationConfig = {
        @ActivationConfigProperty(
            propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
        @ActivationConfigProperty(
            propertyName = "sharedSubscription", propertyValue = "TRUE"),
        @ActivationConfigProperty(
            propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
    },
    mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

    // Default constructor.
    public SubscribingMDB() {
    }

    // @see MessageListener#onMessage(Message)
    public void onMessage(Message message) {
        // implement business logic here
    }
}
```

관련 개념

[구독자 및 구독](#)

[구독 지속성](#)

[301 페이지의 『복제 및 공유된 구독』](#)

여러 소비자가 동일한 구독에 접근할 수 있도록 하는 방법은 두 가지가 있습니다. 이러한 두 개의 메소드는 복제된 구독을 사용하거나 공유된 구독을 사용하여 실행됩니다.

아웃바운드 통신용 자원 어댑터 구성

아웃바운드 통신을 구성하려면 ConnectionFactory 오브젝트와 관리 대상 목적지 오브젝트의 특성을 정의하십시오.

아웃바운드 통신을 사용하는 예

아웃바운드 통신을 사용할 때 애플리케이션 서버에서 실행 중인 애플리케이션은 큐 관리자에 연결을 시작한 다음 동기적인 방식으로 큐에 메시지를 송신하고 큐에서 메시지를 수신합니다. 예를 들어 다음 서블릿 메소드, doGet()에서는 아웃바운드 통신을 사용합니다.

JM 3.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    ...

    // Look up ConnectionFactory and Queue objects from the JNDI namespace
    InitialContext ic = new InitialContext();
    ConnectionFactory cf = (jakarta.jms.ConnectionFactory) ic.lookup("myCF");
    Queue q = (jakarta.jms.Queue) ic.lookup("myQueue");

    // Create and start a connection
    Connection c = cf.createConnection();
    c.start();

    // Create a session and message producer
    Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
    MessageProducer pr = s.createProducer(q);

    // Create and send a message
    Message m = s.createTextMessage("Hello, World!");
    pr.send(m);

    // Create a message consumer and receive the message just sent
    MessageConsumer co = s.createConsumer(q);
    Message mr = co.receive(5000);

    // Close the connection
    c.close();
}
```

JMS 2.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    ...

    // Look up ConnectionFactory and Queue objects from the JNDI namespace
    InitialContext ic = new InitialContext();
    ConnectionFactory cf = (javax.jms.ConnectionFactory) ic.lookup("myCF");
    Queue q = (javax.jms.Queue) ic.lookup("myQueue");

    // Create and start a connection
    Connection c = cf.createConnection();
    c.start();

    // Create a session and message producer
    Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
    MessageProducer pr = s.createProducer(q);

    // Create and send a message
    Message m = s.createTextMessage("Hello, World!");
    pr.send(m);

    // Create a message consumer and receive the message just sent
```

```

    MessageConsumer co = s.createConsumer(q);
    Message mr = co.receive(5000);

    // Close the connection

    c.close();
}

```

서블릿이 HTTP GET 요청을 수신하면 JNDI 네임스페이스에서 `ConnectionFactory` 오브젝트와 `Queue` 오브젝트를 수신하고 이 오브젝트를 사용하여 IBM MQ 큐에 메시지를 송신합니다. 그러면 서블릿이 송신된 메시지를 수신합니다.

아웃바운드 통신에 필요한 자원

아웃바운드 통신을 구성하려면 다음 범주의 Java EE Connector Architecture(JCA) 자원을 정의하십시오.

- 객체의 `ConnectionFactory` 속성 애플리케이션 서버가 객체를 `JMSConnectionFactory` 생성하는 데 사용하는
- 애플리케이션 서버가 큐 객체 또는 `JMSJMS` 토픽 객체를 생성하는 데 사용하는, 지정된 대상 객체의 속성.

이러한 특성을 정의하는 방식은 애플리케이션 서버가 제공하는 관리 인터페이스에 따라 다릅니다. 애플리케이션 서버에서 작성한 `ConnectionFactory`, `Queue` 및 `Topic` 오브젝트는 애플리케이션에서 검색할 수 있는 JNDI 네임스페이스에 바인딩됩니다.

일반적으로 애플리케이션에서 연결해야 하는 각 큐 관리자의 `ConnectionFactory` 오브젝트를 하나 정의합니다. 애플리케이션에서 포인트-투-포인트 도메인에서 액세스해야 하는 각 큐의 `Queue` 오브젝트를 하나 정의합니다. 애플리케이션에서 발행 또는 구독할 수 있는 각 토픽의 `Topic` 오브젝트를 하나 정의합니다. `ConnectionFactory` 오브젝트는 도메인에 독립적일 수 있습니다. 또는 도메인별로 다르거나 포인트-투-포인트 도메인의 경우 `QueueConnectionFactory` 오브젝트이거나 발행/구독 도메인의 경우 `TopicConnectionFactory` 오브젝트일 수 있습니다.

팁: JMS 2.0에서 연결 팩토리를 사용하여 연결과 컨텍스트를 모두 작성할 수 있습니다. 결과적으로 연결과 컨텍스트가 둘 다 혼합되어 있는 연결 팩토리와 연결 풀이 연관될 수 있습니다. 연결 팩토리는 연결을 작성하거나 컨텍스트를 작성하는 데만 사용하는 것이 좋습니다. 그러면 해당 연결 팩토리의 연결 풀에 한 가지 유형의 오브젝트만 포함되므로 풀의 효율성이 향상됩니다.

ConnectionFactory 오브젝트의 특성

426 페이지의 표 65에서는 `ConnectionFactory` 오브젝트의 특성을 나열합니다. 애플리케이션 서버에서는 이러한 특성을 사용하여 JMS `ConnectionFactory` 오브젝트를 작성합니다.

표 65. <code>ConnectionFactory</code> 오브젝트의 특성			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
<code>applicationName</code>	문자열	• 사용 가능한 경우 28자 이하로 조정되는 호출 클래스 이름입니다. 사용할 수 없으면 WebSphere MQ Client for Java 문자열이 사용됩니다.	애플리케이션이 큐 관리자에 등록되는 이름입니다. 이 애플리케이션 이름은 DISPLAY CONN MQSC/PCF 명령(여기서 필드는 APPLTAG 라고 함)을 사용하여 표시하거나 IBM MQ 탐색기 애플리케이션 연결 표시(여기서 필드는 App name 이라고 함)에 표시됩니다.
<code>brokerCCSubQueue</code>	문자열	• SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE • 큐 이름	연결 이용자가 지속 불가능 구독 메시지를 수신하는 큐의 이름입니다.
<code>brokerControlQueue</code>	문자열	• SYSTEM.BROKER.CONTROL.QUEUE • 큐 이름	브로커 제어 큐의 이름입니다.

표 65. ConnectionFactory 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
brokerPubQueue	문자열	• SYSTEM.BROKER.DEFAULT.STREAM • 큐 이름	발행된 메시지가 송신되는 큐(스트림 큐)의 이름입니다.
brokerQueueManager	문자열	• ""(빈 문자열) • 큐 관리자 이름	브로커를 실행 중인 큐 관리자의 이름입니다.
brokerSubQueue	문자열	• SYSTEM.JMS.ND.SUBSCRIBER.QUEUE • 큐 이름	지속 불가능 메시지 이용자가 메시지를 수신하는 큐의 이름입니다. 자세한 정보는 BROKERSUBQ 특성을 참조하십시오.
brokerVersion	문자열	• 지정되지 않음 - 브로커를 V6에서 V7로 마이그레이션한 후 RFH2 헤더를 더 이상 사용하지 않도록 이 특성을 설정하십시오. 마이그레이션하고 나면 이 특성은 더 이상 적절하지 않습니다. • V1 - IBM MQ 발행/구독 브로커를 사용합니다. TRANSPORT가 BIND 또는 CLIENT로 설정된 경우 이 값이 기본값입니다. • V2 - 고유 모드에서 IBM Integration Bus의 브로커를 사용합니다. TRANSPORT가 DIRECT 또는 DIRECTHTTP로 설정된 경우 이 값이 기본값입니다.	사용 중인 브로커의 버전입니다.
ccdtURL	문자열	• 널 • URL(Uniform Resource Locator)	클라이언트 채널 정의 표(CCDT)를 포함하여 파일의 이름과 위치를 식별하고, 파일 액세스 방법을 지정하는 URL입니다.
CCSID	문자열	• 819 • JVM(Java Virtual Machine)에서 지원되는 코드화된 문자 세트 ID	연결의 코드화된 문자 세트 ID입니다.
채널	문자열	• SYSTEM.DEF.SVRCONN • MQI 채널의 이름	사용할 MQI 채널의 이름입니다.
cleanupInterval	int	• 3 600 000 • 양수	발행/구독 정리 유틸리티의 백그라운드 실행 간 간격(밀리초)입니다.
cleanupLevel	문자열	• SAFE • NONE • STRONG • FORCE • NONDUR	브로커 기반 구독 저장소에 대한 정리 레벨입니다.
clientID	문자열	• 널 • 클라이언트 ID	연결에 대한 클라이언트 ID입니다.

표 65. <i>ConnectionFactory</i> 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
cloneSupport	문자열	• DISABLED - 지속 가능한 토픽 구독자의 인스턴스는 한 번에 하나만 실행할 수 있습니다. • ENABLED - 지속 가능한 동일 토픽 구독자의 인스턴스를 동시에 두 개 이상 실행할 수 있지만 각 인스턴스는 개별 JVM(Java Virtual Machine)에서 실행해야 합니다.	동일한 지속 가능 토픽 구독자의 인스턴스를 동시에 두 개 이상 실행할 수 있는지 여부입니다.
connectionNameList	문자열	• localhost(1414) • 쉼표로 분리되는 항목으로 구성된 문자열입니다. 각 항목의 형식은 다음과 같습니다. HOSTNAME (PORT) 여기서 HOSTNAME은 DNS 이름이거나 IP 주소입니다.	아웃바운드 통신에 사용된 TCP/IP 연결 이름의 목록입니다. connectionNameList는 hostname 및 port 특성을 대신합니다. 이 특성은 다중 인스턴스 큐 관리자에 다시 연결하는 데 사용됩니다. connectionNameList는 localAddress와 양식이 비슷하므로 혼동하지 않아야 합니다. localAddress는 로컬 통신의 특성을 지정하는 반면 connectionNameList는 리모트 큐 관리자에 도달하는 방식을 지정합니다.
failIfQuiesce	부울	• true • false	큐 관리자가 정지(quiescing) 상태에 있는 경우 특정 메소드에 대한 호출이 실패하는지 여부입니다.
headerCompression	문자열	• NONE • SYSTEM - RLE 메시지 헤더 압축이 수행됩니다.	연결에서 헤더 데이터를 압축하는 데 사용할 수 있는 기술 목록입니다.
hostName	문자열	• localhost • 호스트 이름 • IP 주소	큐 관리자가 상주하는 시스템의 호스트 이름 또는 IP 주소입니다. hostname 및 port 특성은 connectionNameList 특성이 지정된 경우 이 특성으로 대체됩니다.

표 65. ConnectionFactory 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
localAddress	문자열	• 널 • 다음 형식의 문자열입니다. <pre>[host_name][(low_port [, high_port])]</pre> 여기서 <i>host_name</i>은 호스트 이름 또는 IP 주소이고, <i>low_port</i> 및 <i>high_port</i>는 TCP 포트 번호이며, 대괄호는 선택적 컴포넌트를 나타냅니다.	큐 관리자에 대한 연결에서 이 특성은 다음 중 하나 또는 둘 다를 지정합니다. • 사용할 로컬 네트워크 인터페이스 • 사용할 로컬 포트 또는 로컬 포트의 범위 localAddress는 connectionNameList와 양식이 비슷하므로 혼동하지 않아야 합니다. localAddress는 로컬 통신의 특성을 지정하는 반면 connectionNameList는 리모트 큐 관리자에 도달하는 방식을 지정합니다.
messageCompression	문자열	• NONE • 하나 이상의 다음 값이 공백 문자로 분리된 목록입니다. RLE ZLIBFAST ZLIBHIGH LZ4FAST LZ4HIGH	연결에서 메시지 데이터를 압축하는데 사용할 수 있는 기술 목록입니다.
messageSelection	문자열	• CLIENT • BROKER	메시지 선택을 IBM MQ classes for JMS에서 수행하는지 아니면 브로커에서 수행하는지 판별합니다. brokerVersion 값이 1인 경우 브로커에 의한 메시지 선택은 지원되지 않습니다.
비밀번호	문자열	• 널 • 비밀번호	큐 관리자에 대한 연결을 작성할 때 사용할 기본 비밀번호입니다.
pollingInterval	int	• 5000 • 양수	세션 내에 있는 각 메시지 리스너의 큐에 적합한 메시지가 없는 경우, 이 값은 각 메시지 리스너가 해당 큐에서 메시지 가져오기를 다시 시도하기 전에 경과하는 최대 간격(밀리초)입니다. 세션의 메시지 리스너에 사용 가능한 적합한 메시지가 없는 경우가 자주 발생하면 이 특성 값을 높여 보십시오. 이 특성은 TRANSPORT 값이 BIND 또는 CLIENT인 경우에만 관련이 있습니다.
포트	int	• 1414 • TCP 포트 번호	큐 관리자가 청취하는 포트입니다. hostname 및 port 특성은 connectionNameList 특성이 지정된 경우 이 특성으로 대체됩니다.

표 65. ConnectionFactory 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
providerVersion	문자열	- 지정되지 않음 - 다음 형식 중 하나의 문자열 - V.R.M.F - V.R.M - V.R - V 여기서 V, R, M 및 F는 0 이상의 정수 값입니다.	애플리케이션이 연결하려는 큐 관리자의 버전, 릴리스, 수정 레벨 및 수정 팩입니다.
pubAckInterval	int	25 - 양수	IBM MQ classes for JMS에서 브로커의 수신확인을 요청하기 전에 발행자가 발행한 메시지 수입니다.
queueManager	문자열	""(빈 문자열) - 큐 관리자 이름	연결할 큐 관리자의 이름입니다.
receiveExit ³	문자열	- 널 - 쉼표로 분리된 하나 이상의 항목으로 구성되는 문자열. 여기서 각 항목은 IBM MQ classes for Java 인터페이스인 MQReceiveExit를 구현하는 클래스의 완전한 이름입니다.	채널 수신 엑시트 프로그램이나 그 뒤를 이어 실행할 수신 엑시트 프로그램의 시퀀스를 식별합니다.
receiveExitInit	문자열	- 널 - 쉼표로 분리된 하나 이상의 사용자 데이터 항목으로 구성된 문자열	채널 수신 엑시트 프로그램 호출 시 이 프로그램에 전달되는 사용자 데이터입니다.
rescanInterval	int	5000 - 양수	포인트-투-포인트 도메인의 메시지 이용자가 메시지 선택자를 사용하여 수신할 메시지를 선택하면 IBM MQ classes for JMS가 IBM MQ에서 큐의 MsgDeliverySequence 속성으로 판별되는 시퀀스의 적당한 메시지를 검색합니다. IBM MQ classes for JMS에서 적당한 메시지를 찾아 이용자에게 전달할 때 IBM MQ classes for JMS가 큐의 현재 위치에서 적절한 다음 메시지의 검색을 재개합니다. IBM MQ classes for JMS가 큐의 끝에 도달하거나 이 특성 값으로 판별된 시간 간격(밀리초)이 만기될 때까지 이 방식으로 계속 큐를 검색합니다. 각각의 경우 IBM MQ classes for JMS가 검색을 계속하기 위해 큐의 시작으로 돌아가고 새로운 시간 간격이 시작됩니다.

표 65. <i>ConnectionFactory</i> 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
<code>securityExit</code> ³	문자열	- 널 - IBM MQ classes for Java 인터페이스인 <code>MQSecurityExit</code>를 구현하는 클래스의 완전한 이름	채널 보안 엑시트 프로그램을 식별합니다.
<code>securityExitInit</code>	문자열	- 널 - 사용자 데이터 문자열	채널 보안 엑시트 프로그램 호출 시 이 프로그램에 전달되는 사용자 데이터입니다.
<code>sendCheckCount</code>	int	0 - 양수	변환되지 않은 하나의 JMS 세션에서 비동기 put 오류를 검사하는 사이에 허용할 수 있는 송신 호출 수입니다.
<code>sendExit</code> ³	문자열	- 널 - 선택으로 분리된 하나 이상의 항목으로 구성되는 문자열. 여기서 각 항목은 IBM MQ classes for Java 인터페이스인 <code>MQSendExit</code>를 구현하는 클래스의 완전한 이름입니다.	채널 송신 엑시트 프로그램이나 그 뒤를 이어 실행할 송신 엑시트 프로그램의 시퀀스를 식별합니다.
<code>sendExitInit</code>	문자열	- 널 - 선택으로 분리된 하나 이상의 사용자 데이터 항목으로 구성된 문자열	채널 송신 엑시트 프로그램 호출 시 이 프로그램에 전달되는 사용자 데이터입니다.
<code>shareConvAllowed</code>	부울	- 아니요 - 클라이언트 연결은 소켓을 공유할 수 없습니다. 예 - 클라이언트 연결은 소켓을 공유할 수 있습니다.	채널 정의가 일치하는 경우 클라이언트 연결을 통해 동일한 프로세스부터 동일한 큐 관리자까지 기타 최상위 레벨 JMS 연결과 해당 소켓을 공유할 수 있는지 여부입니다.
<code>sparseSubscriptions</code>	부울	false - 구독에서 자주 일치하는 메시지를 수신합니다. true - 구독에서 드물게 일치하는 메시지를 수신합니다. 이 값을 사용하려면 구독 큐를 읽기 전용으로 열 수 있어야 합니다.	<code>TopicSubscriber</code> 오브젝트의 메시지 검색 정책을 제어합니다.
<code>sslCertStores</code>	문자열	- 널 - 공백으로 분리된 하나 이상의 LDAP URL 문자열입니다. 각 LDAP URL의 형식은 다음과 같습니다. <pre>ldap://host_name [: port]</pre> 여기서 <i>host_name</i>은 호스트 이름 또는 IP 주소이고, <i>port</i>는 TCP 포트 번호이며 대괄호는 선택적 컴포넌트를 나타냅니다.	TLS 연결에서 사용할 인증서 폐기 목록(CRL)을 보유하는 LDAP(Lightweight Directory Access Protocol) 서버입니다.
<code>sslCipherSuite</code>	문자열	- 널 <code>CipherSuite</code>의 이름	TLS 연결에 사용할 <code>CipherSuite</code> 입니다.

표 65. ConnectionFactory 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
sslFipsRequired ²	부울	• false • true	TLS 연결이 JSSE FIPSIBMJava 공급자(IBMJSSEFIPS)에서 지원하는 CipherSuite 를 사용해야 하는지 여부.
sslPeerName	문자열	• 널 • 식별 이름의 템플릿	TLS 연결의 경우 큐 관리자가 제공한 디지털 인증서에서 식별 이름을 검사하는 데 사용하는 템플릿입니다.
sslResetCount	int	• 0 • 0 - 999 999 999 범위의 정수	TLS에서 사용하는 비밀 키를 재협상하기 전에 TLS 연결에서 보내고 받은 바이트 수의 합계입니다.
sslSocketFactory	문자열	javax.net.ssl.SSLSocketFactory 인터페이스의 구현을 제공하는 클래스의 완전한 클래스 이름을 나타내는 문자열로서, 선택적으로 구성자 메소드에 전달될 괄호로 묶은 인수를 포함합니다.	관리 대상 목적지 오브젝트 범위에서 설정된 모든 연결이 이 SSLSocketFactory 인터페이스 구현에서 얻은 소켓을 사용합니다.
statusRefreshInterval	int	• 60000 • 양수	구독자와 큐 관리자의 연결이 끊길 때 발견한 장기 실행 트랜잭션의 새로 고치기 간격(밀리초)입니다. 이 특성은 SUBSTORE 에 QUEUE 값이 있는 경우에만 관련이 있습니다.
subscriptionStore	문자열	• 브로커 • MIGRATE • 큐	IBM MQ classes for JMS가 활성화 구독에 관한 지속적 데이터를 저장하는 위치를 판별합니다.
targetClientMatching	부울	• true • false	수신되는 메시지에 MQRFH2 헤더가 있는 경우에만, 수신되는 메시지의 JMSReplyTo 헤더 필드를 통해 식별된 큐에 송신한 응답 메시지에 MQRFH2 헤더가 있는지 여부입니다. 활성화 스펙에 대해 이 특성을 구성할 수도 있습니다. 자세한 정보는 440 페이지의 『활성화 스펙에 대해 targetClientMatching 특성 구성』의 내용을 참조하십시오.

표 65. *ConnectionFactory* 오브젝트의 특성 (계속)

특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
temporaryModel	문자열	• SYSTEM.DEFAULT.MODEL.QUEUE • SYSTEM.JMS.TEMPQ.MODEL • 임의의 문자열	JMS 임시 큐가 작성되는 모델 큐의 이름입니다. 다음 두 명령문이 모두 true이면 SYSTEM.DEFAULT.MODEL.QUEUE를 사용하십시오. • 애플리케이션에서 비지속 메시지를 승인할 임시 큐를 사용합니다. • 한 번에 하나의 애플리케이션에서만 ConnectionFactory가 가리키는 큐 관리자에 임시 큐를 작성합니다. SYSTEM.DEFAULT.MODEL.QUEUE는 한 번에 하나의 애플리케이션에서만 열릴 수 있음을 참조하십시오. 다음 상황에서는 SYSTEM.JMS.TEMPQ.MODEL을 사용하십시오. • 애플리케이션에서 지속 메시지를 승인할 임시 큐를 사용하는 경우. • 다수의 애플리케이션에서 ConnectionFactory가 가리키는 큐 관리자에 연결할 수 있어야 하며 해당 애플리케이션이 동시에 임시 큐를 작성해야 하는 경우. 다음 상황에서는 DEFPSIST 속성을 YES로 설정하고 DEFSOPT 속성을 SHARED로 설정하여 새 모델 큐를 정의하십시오. • 애플리케이션에서 비지속 메시지를 승인할 임시 큐를 사용하고 다수의 애플리케이션에서 ConnectionFactory가 가리키는 큐 관리자에 연결하며 해당 애플리케이션이 동시에 임시 큐를 작성해야 하는 경우. 새 모델 큐가 작성되면, temporaryModel 특성을 새 모델 큐의 이름으로 설정하십시오.
tempQPrefix	문자열	• ""(빈 문자열) • IBM MQ 동적 큐의 이름을 형성하는 데 사용할 수 있는 접두부. 접두부 형식 규칙은 객체 IBM MQ 설명자 구조체 MQOD의 필드 DynamicQName 내용을 형성하는 규칙과 동일하나, 마지막 비공백 문자는 반드시 별표(*)여야 합니다. 속성 값이 빈 문자열인 경우, AMQ.* IBM MQ classes for JMS 값을 사용합니다 동적 큐를 생성할 때.	IBM MQ 동적 큐의 이름을 형성하는 데 사용하는 접두부.

표 65. <i>ConnectionFactory</i> 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
tempTopicPrefix	문자열	IBM MQ 토픽 문자열의 올바른 문자로만 구성된 널이 아닌 문자열	임시 토픽을 작성할 때 JMS에서 "TEMP/TEMPTOPICPREFIX/unique_id" 양식의 토픽 문자열을 생성합니다. 이 특성의 기본값을 그대로 두는 경우에는 "TEMP/unique_id"만 생성합니다. 비어 있지 않은 TEMPTOPICPREFIX 를 지정하면 이 연결에서 작성된 임시 토픽에 대한 구독자의 관리 큐를 작성하기 위해 특정 모델 큐를 정의할 수 있습니다.
transportType	문자열	• CLIENT • BINDINGS • BINDINGS_THEN_CLIENT	큐 관리자에 연결할 때 사용할 모드. • BINDINGS : 로컬 연결이 속성으로 queueManager 지정된 큐 관리자에 생성됩니다. 그러나 클라이언트 모드 ccdtURL 속성도 지정하는 경우, 해당 ccdtURL 속성이 우선 적용됩니다(이전 버전과의 호환성을 IBM MQ 위해). 리소스 어댑터는 대신 CCDT 파일에 지정된 큐 관리자에 클라이언트 모드 연결을 수행합니다. • 클라이언트 : 큐 매니저와의 연결은, hostname 및 port, 또는 ccdtURL 속성에 의해 connectionNameList 제어됩니다. • BINDINGS_THEN_CLIENT : 리소스 어댑터가 바인딩 모드에서 연결을 시도합니다. 이 연결 시도가 실패하면 리소스 어댑터는 클라이언트 모드로 연결을 시도합니다. 해당 ccdtURL 속성은 이 모드에서 지원되지 않습니다.  z/OS WebSphere Application Server for z/OS 시스템에서 실행 중인 활성화 스펙이 BINDINGS_THEN_CLIENT 전송 모드를 사용하도록 구성되고 이전에 설정된 연결은 끊어진 경우, 활성화 스펙에 의한 재연결 시도에서는 먼저 BINDINGS 전송 모드를 사용하려 합니다. BINDINGS 전송 모드 연결 시도가 실패하면, 활성화 스펙은 연속적으로 CLIENT 전송 모드 연결을 시도합니다.
username	문자열	• 널 • 사용자 이름	큐 관리자에 대한 연결을 작성할 때 사용할 기본 사용자 이름입니다.

표 65. <i>ConnectionFactory</i> 오브젝트의 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
wildcardFormat	int	- CHAR- 브로커 버전 1에서 사용된 대로 문자 와일드카드만 인식 - TOPIC - 브로커 버전 2에서 사용된 대로 토픽 레벨 와일드카드만 인식	사용할 와일드카드 구문 버전입니다.

참고:

1. sslFipsRequired 특성을 사용하는 데 관한 중요한 정보는 390 페이지의 『리소스 IBM MQ 어댑터의 한계 및 IBM MQ 리소스 어댑터에 대한 Jakarta Messaging』을 참조하십시오.
2. 엑시트를 찾을 수 있도록 자원 어댑터를 구성하는 방법에 대한 정보는 260 페이지의 『채널 엑시트를 사용하여 IBM MQ classes for JMS 구성』의 내용을 참조하십시오.

다음 예에서는 *ConnectionFactory* 오브젝트의 일반 특성 세트를 표시합니다.

```
channel:      SYSTEM.DEF.SVRCONN
hostName:    192.168.0.42
port:        1414
queueManager: ExampleQM
transportType: CLIENT
```

관리 대상 목적지 오브젝트의 특성

애플리케이션 서버에서 관리 대상 목적지 오브젝트의 특성을 사용하여 JMS Queue 오브젝트 또는 JMS Topic 오브젝트를 작성합니다.

435 페이지의 표 66에서는 Queue 오브젝트 및 Topic 오브젝트에 공통된 특성을 나열합니다.

표 66. <i>Queue</i> 오브젝트와 <i>Topic</i> 오브젝트에 공통된 특성			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
CCSID	문자열	1208 - JVM(Java Virtual Machine)에서 지원되는 코딩된 문자 세트 ID	목적지의 코딩된 문자 세트 ID입니다.

표 66. Queue 오브젝트와 Topic 오브젝트에 공통된 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
encoding	문자열	• NATIVE • 다음 세 개의 문자로 된 문자열입니다. - 첫 번째 문자는 2진 정수의 표현을 지정합니다. - <i>N</i>은 일반 인코딩을 나타냅니다. - <i>R</i>은 역방향 인코딩을 나타냅니다. - 두 번째 문자는 팩형 10진수의 표현을 지정합니다. - <i>N</i>은 일반 인코딩을 나타냅니다. - <i>R</i>은 역방향 인코딩을 나타냅니다. - 세 번째 문자는 부동 소수점 수의 표현을 지정합니다. - <i>N</i>은 표준 IEEE 인코딩을 나타냅니다. - <i>R</i>은 역방향 IEEE 인코딩을 나타냅니다. - 3은 zSeries 인코딩을 나타냅니다. NATIVE는 NNN 문자열과 동일합니다.	목적지의 2진 정수, 팩형 10진수 및 부동 소수점 수 표현입니다.
expiry	문자열	• APP - 메시지 만기 시간은 메시지 생성자를 통해 판별합니다. • UNLIM - 메시지가 만기되지 않습니다. • 0 - 메시지가 만기되지 않습니다. • 메시지의 만기 시간을 나타내는 양수(밀리초)입니다.	목적지에 송신된 메시지의 만기 시간입니다.
failIfQuiesce	문자열	• true • false	큐 관리자가 정지 상태인 경우 목록지에 액세스하기 위한 시도에 실패하는지 여부입니다.

표 66. Queue 오브젝트와 Topic 오브젝트에 공통된 특성 (계속)

특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
messageBodyStyle	문자열	• UNSPECIFIED • JMS • MQ	큐와 JMS 토픽에 대해 다음 messageBodyStyle 속성을 설정할 수 있습니다: UNSPECIFIED(기본값) • 송신 시 WMQ_TARGET_CLIENT의 값에 따라 IBM MQ classes for JMS에서 MQRFH2 헤더를 생성하여 포함합니다. • 수신 시 MQRFH2의 값에 따라(있는 경우) IBM MQ classes for JMS에서 JMS 메시지 특성을 설정합니다. MQRFH2는 JMS 메시지 본문의 일부로 표시되지 않습니다. JMS • 송신 시 IBM MQ classes for JMS에서 MQRFH2 헤더를 자동으로 생성하여 IBM MQ 메시지에 헤더를 포함시킵니다. • 수신 시 MQRFH2의 값에 따라(있는 경우) IBM MQ classes for JMS에서 JMS 메시지 특성을 설정합니다. MQRFH2는 JMS 메시지 본문의 일부로 표시되지 않습니다. MQ • 송신 시 IBM MQ classes for JMS에서 MQRFH2를 생성하지 않습니다. • 수신 시 IBM MQ classes for JMS에서 MQRFH2를 JMS 메시지 본문의 일부로 표시합니다.
persistence	문자열	• APP - 메시지 지속은 메시지 생성자를 통해 판별합니다. • QDEF - 메시지의 지속성은 큐의 IBM MQ 속성에 DefPersistence 의해 결정됩니다. • PERS - 메시지가 지속적입니다. • NON - 메시지가 지속적이지 않습니다. • HIGH - 메시지의 지속성은 예 설명된 대로 큐의 IBM MQ 242 페이지의 『JMS 지속 메시지』 속성에 NonPersistentMessageClass 의해 결정됩니다.	목적지에 보낸 메시지의 지속성입니다.

표 66. Queue 오브젝트와 Topic 오브젝트에 공통된 특성 (계속)			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
priority	문자열	• APP - 메시지 우선순위는 메시지 생성자를 통해 판별합니다. • QDEF - 메시지의 우선순위는 큐의 IBM MQ QDEF DefPriority 속성에 의해 결정됩니다. • 가장 낮은 우선순위인 0부터 가장 높은 우선순위인 9까지 범위의 정수입니다.	목적지에 보낸 메시지의 우선순위입니다.
putAsyncAllowed	문자열	• QUEUE - 큐 정의를 참조하여 비동기 put이 허용되는지 판별합니다. queue 속성의 이름은, DefPResp (MQLONG)입니다. • TOPIC - 토픽 정의를 참조하여 비동기 put이 허용되는지 판별합니다. • DESTINATION - 큐 또는 토픽 정의를 참조하여 비동기 put이 허용되는지 판별합니다. • DISABLED - 비동기 put이 허용되지 않습니다. • ENABLED - 비동기 put이 허용됩니다.	메시지 생성자가 비동기 put을 사용하여 이 목적지에 메시지를 보낼 수 있는지 여부입니다.
readAheadAllowed	int	• DESTINATION - 큐 또는 토픽 정의를 참조하여 미리 읽기가 허용되는지 판별합니다. • DISABLED - 미리 읽기가 허용되지 않습니다. • ENABLED - 미리 읽기가 허용됩니다. • QUEUE - 큐 정의를 참조하여 미리 읽기가 허용되는지 판별합니다. • TOPIC - 토픽 정의를 참조하여 미리 읽기가 허용되는지 판별합니다.	메시지 이용자와 큐 브라우저에서 비지속 메시지를 수신하기 전에 목적지에서 내부 버퍼로 비지속 메시지를 가져오기 위해 미리 읽기를 사용할 수 있는지 여부입니다.
receiveCCSID	int	• 0 - JVM Charset.defaultCharset 사용 • 1208 - UTF-8 • 지원되는 코드화 문자 세트 ID	큐 관리자 메시지 변환용 대상 CCSID를 설정하는 목적지 특성입니다. receiveConversion 이 QMGR로 설정되는 경우가 아니면 값은 무시됩니다.
receiveConversion	문자열	• CLIENT_MSG • QMGR	큐 관리자가 데이터 변환을 수행할지 판별하는 목적지 특성입니다.
targetClient	문자열	• JMS - 메시지의 대상은 JMS 애플리케이션입니다. • MQ - 메시지의 대상은 비JMS IBM MQ 애플리케이션입니다.	목적지에 송신된 메시지의 대상이 JMS 애플리케이션인지 여부입니다. 대상이 JMS 애플리케이션인 메시지에 MQRFH2 헤더가 포함되어 있습니다.

439 페이지의 표 67에서는 Queue 오브젝트에 특정한 특성을 나열합니다.

표 67. Queue 오브젝트에 특정한 특성			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
baseQueueManagerName	문자열	""(빈 문자열) - 큐 관리자 이름	근본적인 IBM MQ 큐를 소유하는 큐 관리자의 이름입니다.
baseQueueName	문자열	""(빈 문자열) - 큐 이름	근본적인 IBM MQ 큐의 이름입니다.

439 페이지의 표 68에서는 Topic 오브젝트에 특정한 특성을 나열합니다.

표 68. Topic 오브젝트에 특정한 특성			
특성 이름	유형	올바른 값(기본값은 굵은체로 표시됨)	설명
baseTopicName	문자열	""(빈 문자열) - 토픽 이름	근본적인 토픽의 이름입니다.
brokerCCDurSubQueue >	문자열	SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE - 큐 이름	연결 이용자가 지속 가능한 구독 메시지를 수신하는 큐의 이름입니다.
brokerDurSubQueue	문자열	SYSTEM.JMS.D.SUBSCRIBER.QUEUE - 큐 이름	지속 가능한 토픽 구독자가 메시지를 수신하는 큐의 이름입니다. 자세한 내용은 문서의 IBM MQ Explorer 해당 BROKEDURRSUBQ 속성을 참조하십시오.
brokerPubQueue	문자열	- 설정되지 않음 - 큐 이름	발행된 메시지가 송신되는 큐 (스트림 큐)의 이름입니다. 이 특성 값은 ConnectionFactory 오브젝트의 brokerPubQueue 특성 값을 대체합니다. 그러나 이 특성 값을 설정하지 않은 경우, ConnectionFactory 오브젝트의 brokerPubQueue 특성 값이 대신 사용됩니다.
brokerPubQueueManager	문자열	""(빈 문자열) - 큐 관리자 이름	토픽에서 발행된 메시지가 송신되는 큐를 소유하는 큐 관리자의 이름입니다.
brokerVersion	문자열	- 설정되지 않음 1 2	사용 중인 브로커의 버전입니다. 이 특성 값은 ConnectionFactory 오브젝트의 brokerVersion 특성 값을 대체합니다. 그러나 이 특성 값을 설정하지 않은 경우, ConnectionFactory 오브젝트의 brokerVersion 특성 값이 대신 사용됩니다.

다음 예에서는 Queue 오브젝트의 특성 세트를 표시합니다.

```
expiry:          UNLIM
persistence:    QDEF
```

```
baseQueueManagerName: ExampleQM
baseQueueName:        SYSTEM.JMS.TEMPQ.MODEL
```

다음 예에서는 Topic 오브젝트의 특성 세트를 표시합니다.

```
expiry:           UNLIM
persistence:      NON
baseTopicName:    myTestTopic
```

관련 태스크

[MQI 클라이언트에서 실행 시 인증된 CipherSpecsFIPS 만 사용하도록 지정](#)

[WebSphere Application Server에서 JMS 자원 구성](#)

관련 참조

[Federal Information Processing Standards \(FIPS\)를 위한 AIX, Linux, and Windows](#)

활성화 스펙에 대해 **targetClientMatching** 특성 구성

요청 메시지에 MQRFH2 헤더가 포함되지 않을 때 MQRFH2 헤더가 응답 메시지에서 포함되도록 활성화 스펙에 대해 **targetClientMatching** 특성을 구성할 수 있습니다. 이는 애플리케이션이 응답 메시지에서 정의하는 메시지 특성이 메시지 전송 시에 포함됨을 의미합니다.

이 태스크 정보

MDB(Message-Driven Bean) 애플리케이션이 IBM MQ JCA 자원 어댑터 활성화 스펙을 통해 MQRFH2 헤더가 포함되지 않은 메시지를 이용하며 나중에 요청 메시지의 JMSReplyTo 필드에서 작성된 JMS 대상으로 응답 메시지를 전송하는 경우, 요청 메시지에는 포함되지 않아도 응답 메시지에는 MQRFH2 헤더가 포함되어야 합니다. 그렇지 않으면, 애플리케이션이 응답 메시지에서 정의한 메시지 특성이 유실됩니다.

targetClientMatching 특성은 수신 메시지에 MQRFH2 헤더가 있는 경우에만 수신 메시지의 JMSReply 헤더 필드로 식별된 큐에 전송된 응답 메시지에 MQRFH2 헤더가 있는지 여부를 정의합니다. WebSphere Application Server 및 WebSphere Liberty 모두에서 활성화 스펙에 대해 이 특성을 구성할 수 있습니다.

targetClientMatching 특성의 값을 false로 설정하는 경우, MQRFH2 헤더는 MQRFH2가 포함되지 않은 수신 요청 메시지의 JMSReplyTo 헤더에서 작성된 JMS 대상으로 전송된 응답 메시지에 포함될 수 있습니다. 이는 JMS 대상의 **targetClient** 특성이 메시지에 MQRFH2 헤더가 포함되어 있음을 의미하는 0 값으로 설정되었기 때문입니다. 아웃바운드 메시지에 MQRFH2 헤더가 존재하면 IBM MQ 큐로 전송될 때 메시지에서 사용자 정의된 메시지 특성의 저장에 허용됩니다.

targetClientMatching 특성이 true로 설정되어 있으며 요청 메시지에 MQRFH2 헤더가 포함되지 않은 경우, MQRFH2 헤더는 응답 메시지에 포함되지 않습니다.

프로시저

- WebSphere Application Server에서 관리 콘솔을 사용하여 **targetClientMatching** 특성을 IBM MQ 활성화 스펙의 사용자 정의 특성으로 정의하십시오.
 - 도움말 탐색창에서 **자원 -> JMS -> 활성화 스펙**을 클릭하십시오.
 - 보거나 변경할 활성화 스펙의 이름을 선택하십시오.
 - 사용자 정의 특성 -> 새로 작성**을 클릭한 후 새 사용자 정의 특성의 세부사항을 입력하십시오.
특성의 이름을 **targetClientMatching**으로, 유형을 **java.lang.Boolean**으로, 그리고 값을 **false**로 설정하십시오.
- WebSphere Liberty에서 **server.xml**내의 활성화 스펙 정의에 **targetClientMatching** 특성을 지정하십시오.

예를 들면, 다음과 같습니다.

```
<jmsActivationSpec id="SimpleMDBApplication/SimpleEchoMDB/SimpleEchoMDB">
  <properties.wmqJms destinationRef="MDBRequestQ"
    queueManager="MY_QMGR" transportType="BINDINGS" targetClientMatching="false"/>
  <authData password="*****" user="tom"/>
</jmsActivationSpec>
```

관련 개념

213 페이지의 『JMS 애플리케이션에서 목적지 작성』

JNDI(Java Naming and Directory Interface)에서 관리 대상 오브젝트로서 목적지를 검색하는 대신, JMS 애플리케이션은 세션을 사용하여 런타임에 동적으로 목적지를 작성할 수 있습니다. 애플리케이션은 URI(Uniform Resource Identifier)를 사용하여 IBM MQ 큐 또는 토픽을 식별하고, 선택적으로 큐 또는 토픽 오브젝트의 하나 이상의 특성을 지정할 수 있습니다.

425 페이지의 『아웃바운드 통신용 자원 어댑터 구성』

아웃바운드 통신을 구성하려면 ConnectionFactory 오브젝트와 관리 대상 목적지 오브젝트의 특성을 정의하십시오.

IBM MQ 메시지에 의한 WebSphere Liberty의 Bean 일시정지

활성화 스펙의 **maxSequentialDeliveryFailures** 특성은 MDB를 일시정지하기 전에 자원 어댑터가 허용하는 MDB(Message-Driven Bean) 인스턴스에 대한 최대 순차적 메시지 전달 수를 정의합니다.

시작하기 전에

WebSphere Liberty에서 MDB를 일시정지시킬 수 있는 이벤트 세트를 인식해야 합니다. 자원 어댑터는 다음 중 하나를 메시지 전송 실패로 간주합니다.

- MDB의 **onMessage** 방법으로 부터 예외 처리되는 미확인 예외.
- 메시지를 MDB에게 전달하기 전에 자원 어댑터의 처리에서 발생하는 **JMSEException**.
- 메시지를 MDB로 전달한 후에 자원 어댑터 처리에서 발생하는 **JMSEException**.
- 롤백 중인 메시지를 이용하는 데 사용되는 XA 트랜잭션 또는 로컬 트랜잭션.
- 메시지를 MDB에 전달하기 위해 애플리케이션 서버에서 사용할 수 있는 스레드가 없음.

이 태스크 정보

maxSequentialDeliveryFailures 특성의 기본값은 -1이며, 이는 MDB가 일시정지되지 않음을 의미합니다. 기타 음수 값은 -1과 동일하게 처리됩니다. 값은 다음과 같습니다.

- 0은 MDB가 첫 번째 오류에 일시정지함을 의미함
- 1은 MDB가 두 개의 순차 오류에 대해 일시정지함을 의미함
- 2는 MDB가 세 개의 순차 오류 등에 대해 일시정지함을 의미함

WebSphere Liberty에서만 활성화 스펙에 대해 이 특성을 구성할 수 있으며 Liberty 레벨이 18.0.0.4 이상일 때 이 특성을 구성할 수 있습니다.



주의: 속성을 Liberty 이외의 애플리케이션 서버 환경에서 기본값이 아닌 값으로 설정하면 값은 무시되고 로그에 경고 메시지가 작성됩니다.

또한 IBM MQ 자원 어댑터를 일반 자원 어댑터로 WebSphere Liberty에 설치할 수 있습니다. 이렇게 하면 모든 IBM MQ 및 WebSphere Application Server 통합 기능이 비활성화되고 자원 어댑터가 Liberty에서 실행 중인지 발견할 수 없게 됩니다. 따라서 **maxSequentialDeliveryFailures**를 0보다 크거나 같은 값으로 설정할 수 없으므로 로그에 경고 메시지가 표시됩니다.

프로시저

- 에서 WebSphere Liberty, 활성화 사양 정의 내부의 에 대한 속성을 **maxSequentialDeliveryFailures**server.xml 지정하십시오.
예를 들면, 다음과 같습니다.

```
<jmsActivationSpec>
  <properties.wmqJms destinationRef="jndi/MDBQ"
                    transportType="BINDINGS"
                    queueManager="MQ21"
                    maxSequentialDeliveryFailures="1"/>
</jmsActivationSpec>
```

관련 개념

425 페이지의 『아웃바운드 통신용 자원 어댑터 구성』

아웃바운드 통신을 구성하려면 `ConnectionFactory` 오브젝트와 관리 대상 목적지 오브젝트의 특성을 정의하십시오.

리소스 어댑터 IBM MQ 설치 확인

리소스 IBM MQ 어댑터 및 IBM MQ 리소스 어댑터용 설치 검증 테스트(IVT) Jakarta Messaging 프로그램은 EAR 파일로 제공됩니다. 프로그램을 사용하려면 해당 프로그램을 배포하고 일부 객체를 JCA 리소스로 정의해야 합니다. 이를 수행하기 위해 취해야 할 단계는 사용 중인 애플리케이션 서버에 따라 다릅니다.

JMS 2.0 WebSphere Liberty IBM MQ 메시징 공급자를 사용하여 배포된 IBM MQ 리소스 어댑터 설치 확인하기

IBM MQ 리소스 어댑터 설치 확인 테스트(IVT) 프로그램은 EAR(엔터프라이즈 아카이브) 파일 `wmq.jmsra.ivt.ear`에 제공됩니다. 이 파일은 IBM MQ 리소스 어댑터 RAR 파일 `wmq.jmsra.rar`과 같은 디렉토리에 IBM MQ classes for JMS로 설치됩니다.

이 태스크 정보

`wmq.jmsra.ivt.ear` 및 `wmq.jmsra.rar` 파일이 설치되는 위치에 대한 자세한 내용은 398 페이지의 『리소스 IBM MQ 어댑터 설치』를 참조하세요.

IVT 프로그램에는 IBM MQ 대기열로 메시지를 보내고 받을 수 있는지 테스트하는 서블릿과 메시지 기반 빈(MDB)이 포함되어 있습니다. IVT 프로그램을 사용하여 IBM MQ 리소스 어댑터가 WebSphere Liberty IBM MQ 메시징 공급자를 사용하여 WebSphere Liberty에 올바르게 배포되었는지, 분산 트랜잭션을 지원하도록 구성되어 있는지 확인할 수 있습니다.

IVT 프로그램을 실행하려면 먼저 JMS 관리 객체 3개를 정의해야 합니다:

- 대기열 관리자의 대기열에 매핑되는 대기열 대상입니다.
- 대기열 관리자에 연결하는 방법에 대한 세부 정보가 포함된 연결 팩토리 개체입니다. 이는 IVT의 서블릿 부분에서 사용됩니다.
- 활성화 사양입니다. 여기에는 대기열 관리자에 연결하는 방법에 대한 정보가 포함되어 있으며 메시지를 모니터링할 대기열을 알 수 있도록 대기열 대상에 대한 참조가 포함되어야 합니다. 이는 IVT의 MDB 부분에서 사용됩니다.

이러한 개체의 속성은 대기열 관리자에 따라 다릅니다. 다음은 간단한 예제입니다.

```
Queue destination:
  baseQueueName:      TEST.QUEUE

Connection factory:
  channel:             SYSTEM.DEF.SVRCONN
  hostName:            localhost
  port:                1414
  queueManager:        QM1
  transportType:       CLIENT

Activation specification:
  channel:             SYSTEM.DEF.SVRCONN
  hostName:            localhost
  port:                1414
  queueManager:        QM1
  transportType:       CLIENT
```

참고:

1. 다음 단계를 수행하기 전에 WebSphere Liberty IBM MQ 메시징 공급자를 사용하여 IBM MQ 리소스 어댑터를 WebSphere Liberty에 설치해야 합니다. 자세한 정보는 398 페이지의 『리소스 IBM MQ 어댑터 설치』 및 392 페이지의 『메시징 WebSphere Liberty IBM MQ 공급자와 함께 리소스 IBM MQ 어댑터 사용』의 내용을 참조하십시오.

2. 대기열 관리자의 보안 구성에 따라 IVT 애플리케이션이 실행될 때 이유 코드 2035(MQRC_NOT_AUTHORIZED)가 포함된 오류를 보고할 수 있습니다. 자세한 내용은 [2035\(07F3\)](#) (RC2035): MQRC_NOT_AUTHORIZED를 참조하세요.

프로시저

1. WebSphere Liberty 서버를 중지하십시오.
2. IBM MQ 리소스 어댑터 IVT 프로그램 `wmq.jmsra.ivt.ear` 을 `<server.config.dir>/dropins` 디렉토리에 복사합니다. 여기서 `<server.config.dir>` 은 WebSphere Liberty 서버의 구성 디렉터리입니다.
3. WebSphere Liberty `server.xml` 파일을 편집하고 다음 예시와 같이 항목을 추가합니다:

```
<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
  <properties.wmqJms channel="SYSTEM.DEF.SVRCONN"
    hostname="localhost"
    port="1414"
    queueManager="QM1"
    transportType="CLIENT"/>
</jmsQueueConnectionFactory>

<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queue -->
<jmsQueue id="IVT-Queue" jndiName="IVTQueue">
  <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
  <properties.wmqJms channel="SYSTEM.DEF.SVRCONN"
    destinationRef="IVT-Queue"
    hostName="localhost"
    maxPoolDepth="1"
    port="1414"
    queueManager="QM1"
    transportType="CLIENT"/>
</jmsActivationSpec>
```

참고: 이러한 정의는 앞서 표시된 [예제 값](#)을 사용합니다. 대기열 관리자와 대기열의 값이 다른 경우 정의에서 해당 값을 사용해야 합니다.

4. `server.xml` 파일의 `<featureManager>` 섹션에 다음 항목이 포함되어 있는지 확인합니다:

```
<feature>wmqJmsClient-2.0</feature>
<feature>jndi-1.0</feature>
<feature>jsp-2.3</feature>
<feature>mdb-3.2</feature>
```

5. WebSphere Liberty을(를) 시작하십시오.
6. 다음 형식의 URL을 웹 브라우저에 입력하여 IVT 프로그램을 시작하십시오.

```
http://<app_server_host>:<port>/WMQ_IVT/
```

여기서 `app_server_host` 는 애플리케이션 서버가 실행 중인 시스템의 IP 주소 또는 호스트 이름이며, `port` 는 애플리케이션 서버가 리스닝 중인 TCP 포트의 번호입니다. 다음은 예제입니다.

```
http://localhost:9080/WMQ_IVT/
```

이제 IVT 프로그램의 초기 페이지가 표시됩니다.

IBM MQ JavaEE 7 Connector Architecture IVT

Installation Verification Test

Check to ensure that the IBM MQ J2EE Connector Architecture resource adapter is correctly installed.

Connection Factory:

Destination:

그림 46. IVT 프로그램의 초기 페이지

7. 테스트를 실행하려면 **IVT 실행**을 클릭하십시오.

다음은 IVT가 성공하면 표시되는 페이지의 예입니다.

IBM MQ JavaEE 7 Connector Architecture IVT

Running Installation Verification Test:

Using Connection Factory: *IVTCF*
Using Destination: *IVTQueue*

Creating initial context...	☑
Looking up MQ Connection Factory...	☑
Looking up Destination...	☑
Creating connection...	☑
Starting connection...	☑
Creating session...	☑
Creating a temporary reply queue...	☑
Creating message consumer...	☑
Creating message producer...	☑
Creating message...	☑
Sending message to the MDB...	☑
Receiving response message from the MDB...	☑
Closing connection...	☑

Installation Verification Test completed successfully!

[Re-run Installation Verification Test](#)

그림 47. 성공한 IVT 결과를 보여주는 페이지

다음은 IVT가 실패할 경우 표시되는 페이지의 예입니다. 실패 원인에 대한 추가 정보를 얻으려면 **스택 추적 보기**를 클릭하십시오.

IBM MQ JavaEE 7 Connector Architecture IVT

Running Installation Verification Test:

Using Connection Factory:IVTCF
Using Destination:IVTQueue

Creating initial context...	☑
Looking up MQ Connection Factory...	☑
Looking up Destination...	☑
Creating connection...	☑
Starting connection...	☑
Creating session...	☑
Creating a temporary reply queue...	☑
Creating message consumer...	☑
Creating message producer... failed to create message producer!	☒

Installation Verification Test failed!

Error received - JMS Exception:

com.ibm.msg.client.jms.DetailedJMSException: JMSWMQ2008: Failed to open MQ queue 'TEST.QUEUE'.
JMS attempted to perform an MQOPEN, but IBM MQ reported an error.
Use the linked exception to determine the cause of this error. Check that the specified queue and queue manager are defined correctly.

[View Stack Trace](#)

Installation Verification Test failed!

[Retry Installation Verification Test](#)
[Change IVT parameters](#)

그림 48. 실패한 IVT 결과를 보여주는 페이지

JMS 2.0 에서 일반 리소스 어댑터로 배포된 **IBM MQ** 리소스 어댑터 설치를 확인합니다

WebSphere Liberty

IBM MQ 리소스 어댑터 설치 확인 테스트(IVT) 프로그램은 EAR(엔터프라이즈 아카이브) 파일 `wmq.jmsra.ivt.ear` 에 제공됩니다. 이 파일은 IBM MQ 리소스 어댑터 RAR 파일 `wmq.jmsra.rar` 과 같은 디렉토리에 IBM MQ classes for JMS 로 설치됩니다.

이 태스크 정보

`wmq.jmsra.ivt.ear` 및 `wmq.jmsra.rar` 파일이 설치되는 위치에 대한 자세한 내용은 [398 페이지의 『리소스 IBM MQ 어댑터 설치』](#) 을 참조하세요.

IVT 프로그램에는 IBM MQ 대기열로 메시지를 보내고 받을 수 있는지 테스트하는 서블릿과 메시지 기반 빈(MDB)이 포함되어 있습니다. IVT 프로그램을 사용하여 IBM MQ 리소스 어댑터가 일반 리소스 어댑터로 WebSphere Liberty 에 올바르게 배포되었는지, 분산 트랜잭션을 지원하도록 구성되었는지 확인할 수 있습니다.

IVT 프로그램을 실행하려면 먼저 JMS 관리 객체 3개를 정의해야 합니다:

- 대기열 관리자의 대기열에 매핑되는 대기열 대상입니다.
- 대기열 관리자에 연결하는 방법에 대한 세부 정보가 포함된 연결 팩토리 개체입니다. 이는 IVT의 서블릿 부분에서 사용됩니다.
- 활성화 사양입니다. 여기에는 대기열 관리자에 연결하는 방법에 대한 정보가 포함되어 있으며 메시지를 모니터링할 대기열을 알 수 있도록 대기열 대상에 대한 참조가 포함되어야 합니다. 이는 IVT의 MDB 부분에서 사용됩니다.

이러한 개체의 속성은 대기열 관리자에 따라 다릅니다. 다음은 간단한 예제입니다.

```
Queue destination:
  baseQueueName:    TEST.QUEUE

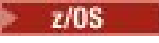
Connection factory:
  channel:          SYSTEM.DEF.SVRCONN
  hostName:         localhost
  port:             1414
  queueManager:     QM1
```

```

transportType:      CLIENT
Activation specification:
channel:            SYSTEM.DEF.SVRCONN
hostName:           localhost
port:               1414
queueManager:       QM1
transportType:      CLIENT

```

참고:

1. 다음 단계를 수행하기 전에 IBM MQ 리소스 어댑터를 일반 리소스 어댑터로 WebSphere Liberty 에 설치해야 합니다. 자세한 정보는 398 페이지의 『리소스 IBM MQ 어댑터 설치』 및 394 페이지의 『리소스 IBM MQ 어댑터를 일반 리소스 어댑터로 사용하기』의 내용을 참조하십시오.
2. 대기열 관리자의 보안 구성에 따라 IVT 애플리케이션이 실행될 때 이유 코드 2035(MQRC_NOT_AUTHORIZED)가 포함된 오류를 보고할 수 있습니다. 자세한 내용은 2035(07F3) (RC2035): MQRC_NOT_AUTHORIZED를 참조하세요.
3.  IBM MQ 리소스 어댑터를 일반 리소스 어댑터로 WebSphere Liberty 에 설치하는 것은 z/OS 에서 지원되지 않습니다.

프로시저

1. WebSphere Liberty 서버를 중지하십시오.
2. IBM MQ 리소스 어댑터 IVT 프로그램 wmq.jmsra.ivt.ear 을 <server.config.dir>/dropins 디렉토리에 복사합니다. 여기서 <server.config.dir> 은 WebSphere Liberty 서버의 구성 디렉터리입니다.
3. WebSphere Liberty server.xml 파일을 편집하고 다음 예시와 같이 항목을 추가합니다:

```

<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
  <properties.<resource_adapter_id>
channel="SYSTEM.DEF.SVRCONN"
                                hostname="localhost"
                                port="1414"
                                queueManager="QM1"
                                transportType="CLIENT"/>
  </jmsQueueConnectionFactory>

  <connectionManager id="ConMgrIVT" maxPoolSize="10"/>

  <!-- IVT Queue -->
  <jmsQueue id="IVT-Queue" jndiName="IVTQueue">
    <properties.<resource_adapter_id> baseQueueName="TEST.QUEUE"/>
  </jmsQueue>

  <!-- IVT Activation Spec -->
  <jmsActivationSpec id="wmq.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
    <properties.<resource_adapter_id> channel="SYSTEM.DEF.SVRCONN"
                                destinationRef="IVT-Queue"
                                destinationType="javax.jms.Queue"
                                hostName="localhost"
                                maxPoolDepth="1"
                                port="1414"
                                queueManager="QM1"
                                transportType="CLIENT"/>
  </jmsActivationSpec>

```

여기서 <resource_adapter_id> 은 server.xml 파일에서 IBM MQ 리소스 어댑터를 정의하는 resourceadapter 요소의 id 속성 값입니다.

참고: 이러한 정의는 앞서 표시된 [예제 값](#)을 사용합니다. 대기열 관리자와 대기열의 값이 다른 경우 정의에서 해당 값을 사용해야 합니다.

4. server.xml 파일의 <featureManager> 섹션에 다음 항목이 포함되어 있는지 확인합니다:

```

<feature>jca-1.7</feature>
<feature>jms-2.0</feature>
<feature>jndi-1.0</feature>

```

```
<feature>jsp-2.3</feature>
<feature>mdb-3.2</feature>
```

5. WebSphere Liberty을(를) 시작하십시오.
6. 다음 형식의 URL을 웹 브라우저에 입력하여 IVT 프로그램을 시작하십시오.

```
http://<app_server_host>:<port>/WMQ_IVT/
```

여기서 *app_server_host* 는 애플리케이션 서버가 실행 중인 시스템의 IP 주소 또는 호스트 이름이며, *port* 는 애플리케이션 서버가 리스닝 중인 TCP 포트의 번호입니다. 다음은 예제입니다.

```
http://localhost:9080/WMQ_IVT/
```

이제 IVT 프로그램의 초기 페이지가 표시됩니다.



IBM MQ JavaEE 7 Connector Architecture IVT

Installation Verification Test
Check to ensure that the IBM MQ J2EE Connector Architecture resource adapter is correctly installed.

Connection Factory:

Destination:

그림 49. IVT 프로그램의 초기 페이지

7. 테스트를 실행하려면 **IVT 실행**을 클릭하십시오.
- 다음은 IVT가 성공하면 표시되는 페이지의 예입니다.

IBM MQ JavaEE 7 Connector Architecture IVT

Running Installation Verification Test:

Using Connection Factory: *IVTCF*
Using Destination: *IVTQueue*

Creating initial context...	☑
Looking up MQ Connection Factory...	☑
Looking up Destination...	☑
Creating connection...	☑
Starting connection...	☑
Creating session...	☑
Creating a temporary reply queue...	☑
Creating message consumer...	☑
Creating message producer...	☑
Creating message...	☑
Sending message to the MDB...	☑
Receiving response message from the MDB...	☑
Closing connection...	☑

Installation Verification Test completed successfully!

[View Message Contents](#)

[Re-run Installation Verification Test](#)

그림 50. 성공한 IVT 결과를 보여주는 페이지

다음은 IVT가 실패할 경우 표시되는 페이지의 예입니다. 실패 원인에 대한 추가 정보를 얻으려면 **스택 추적 보기**를 클릭하십시오.

IBM MQ JavaEE 7 Connector Architecture IVT

Running Installation Verification Test:

Using Connection Factory: *IVTCF*
Using Destination: *IVTQueue*

Creating initial context...	☑
Looking up MQ Connection Factory...	☑
Looking up Destination...	☑
Creating connection...	☑
Starting connection...	☑
Creating session...	☑
Creating a temporary reply queue...	☑
Creating message consumer...	☑
Creating message producer... failed to create message producer!	☒

Installation Verification Test failed!

Error received - JMS Exception:

com.ibm.msg.client.jms.DetailedJMSSecurityException: JMSWMQ2008: Failed to open MQ queue 'TEST.QUEUE'.
JMS attempted to perform an MQOPEN, but IBM MQ reported an error.
Use the linked exception to determine the cause of this error. Check that the specified queue and queue manager are defined correctly.

[View Stack Trace](#)

Installation Verification Test failed!

[Retry Installation Verification Test](#)
[Change IVT parameters](#)

그림 51. 실패한 IVT 결과를 보여주는 페이지

▶ JMS 2.0 다른 Java EE 애플리케이션 서버에 IBM MQ 리소스 어댑터 설치 확인

IBM MQ 리소스 어댑터 설치 확인 테스트(IVT) 프로그램은 EAR(엔터프라이즈 아카이브) 파일 `wmq.jmsra.ivt.ear`에 제공됩니다. 이 파일은 IBM MQ 리소스 어댑터 RAR 파일 `wmq.jmsra.rar`과 같은 디렉토리에 IBM MQ classes for JMS로 설치됩니다.

이 태스크 정보

`wmq.jmsra.ivt.ear` 및 `wmq.jmsra.rar` 파일이 설치되는 위치에 대한 자세한 내용은 398 페이지의 『리소스 IBM MQ 어댑터 설치』을 참조하세요.

IVT 프로그램에는 IBM MQ 대기열로 메시지를 보내고 받을 수 있는지 테스트하는 서블릿과 메시지 기반 빈(MDB)이 포함되어 있습니다. IVT 프로그램을 사용하여 IBM MQ 리소스 어댑터가 애플리케이션 서버에 올바르게 배포되었는지, 분산 트랜잭션을 지원하도록 구성되었는지 확인할 수 있습니다. 또한 IBM 지원팀에서 애플리케이션 서버가 올바르게 구성되었는지 확인하기 위해 IVT 프로그램을 실행하도록 요청할 수도 있습니다.

IVT 프로그램을 실행하려면 먼저 JMS 관리 객체 3개를 정의해야 합니다:

- 대기열 관리자의 대기열에 매핑되는 대기열 대상입니다.
- 대기열 관리자에 연결하는 방법에 대한 세부 정보가 포함된 연결 팩토리 개체입니다. 이는 IVT의 서블릿 부분에서 사용됩니다.
- 활성화 사양입니다. 여기에는 대기열 관리자에 연결하는 방법에 대한 정보가 포함되어 있으며 메시지를 모니터링할 대기열을 알 수 있도록 대기열 대상에 대한 참조가 포함되어야 합니다. 이는 IVT의 MDB 부분에서 사용됩니다.

이러한 개체의 속성은 대기열 관리자에 따라 다릅니다. 다음은 간단한 예제입니다.

```
Queue destination:
  baseQueueName:      TEST.QUEUE

Connection factory:
  channel:             SYSTEM.DEF.SVRCONN
  hostName:            localhost
  port:                1414
  queueManager:        QM1
  transportType:       CLIENT

Activation specification:
  channel:             SYSTEM.DEF.SVRCONN
  hostName:            localhost
  port:                1414
  queueManager:        QM1
  transportType:       CLIENT
```

이러한 객체를 정의하는 데 사용되는 메커니즘은 애플리케이션 서버에 따라 다릅니다. 이 작업을 수행하는 방법은 애플리케이션 서버 설명서를 참조하세요.

IVT 프로그램은 연결 팩토리 개체가 애플리케이션 서버의 JNDI 네임스페이스에 `jms/ivt/IVTCF`이라는 이름으로 바인딩되고 큐 대상 개체는 `jms/ivt/IVTQueue`이라는 이름으로 바인딩될 것으로 예상합니다. 다른 이름을 사용할 수 있지만 그런 경우 IVT 프로그램의 초기 페이지에서 오브젝트 이름을 입력하고 EAR 파일을 적절하게 수정해야 합니다.

IVT 프로그램을 배포하고 애플리케이션 서버가 JMS 개체를 생성하여 JNDI 네임스페이스에 바인딩한 후에는 다음 단계를 완료하여 IVT 프로그램을 시작할 수 있습니다.

참고:

1. 이 단계에서는 IBM MQ 리소스 어댑터가 애플리케이션 서버에 이미 설치되어 있다고 가정합니다. 자세한 정보는 398 페이지의 『리소스 IBM MQ 어댑터 설치』의 내용을 참조하십시오.
2. 대기열 관리자의 보안 구성에 따라 IVT 애플리케이션이 실행될 때 이유 코드 2035(MQRC_NOT_AUTHORIZED)가 포함된 오류를 보고할 수 있습니다. 자세한 내용은 2035(07F3)(RC2035): MQRC_NOT_AUTHORIZED를 참조하세요.

프로시저

1. 다음 형식의 URL을 웹 브라우저에 입력하여 IVT 프로그램을 시작하십시오.

```
http://<app_server_host>:<port>/WMQ_IVT/
```

여기서 *app_server_host* 는 애플리케이션 서버가 실행 중인 시스템의 IP 주소 또는 호스트 이름이며, *port* 는 애플리케이션 서버가 리스닝 중인 TCP 포트의 번호입니다. 다음은 예제입니다.

```
http://localhost:9080/WMQ_IVT/
```

이제 IVT 프로그램의 초기 페이지가 표시됩니다.



그림 52. IVT 프로그램의 초기 페이지

2. 테스트를 실행하려면 **IVT 실행**을 클릭하십시오.

다음은 IVT가 성공하면 표시되는 페이지의 예입니다.

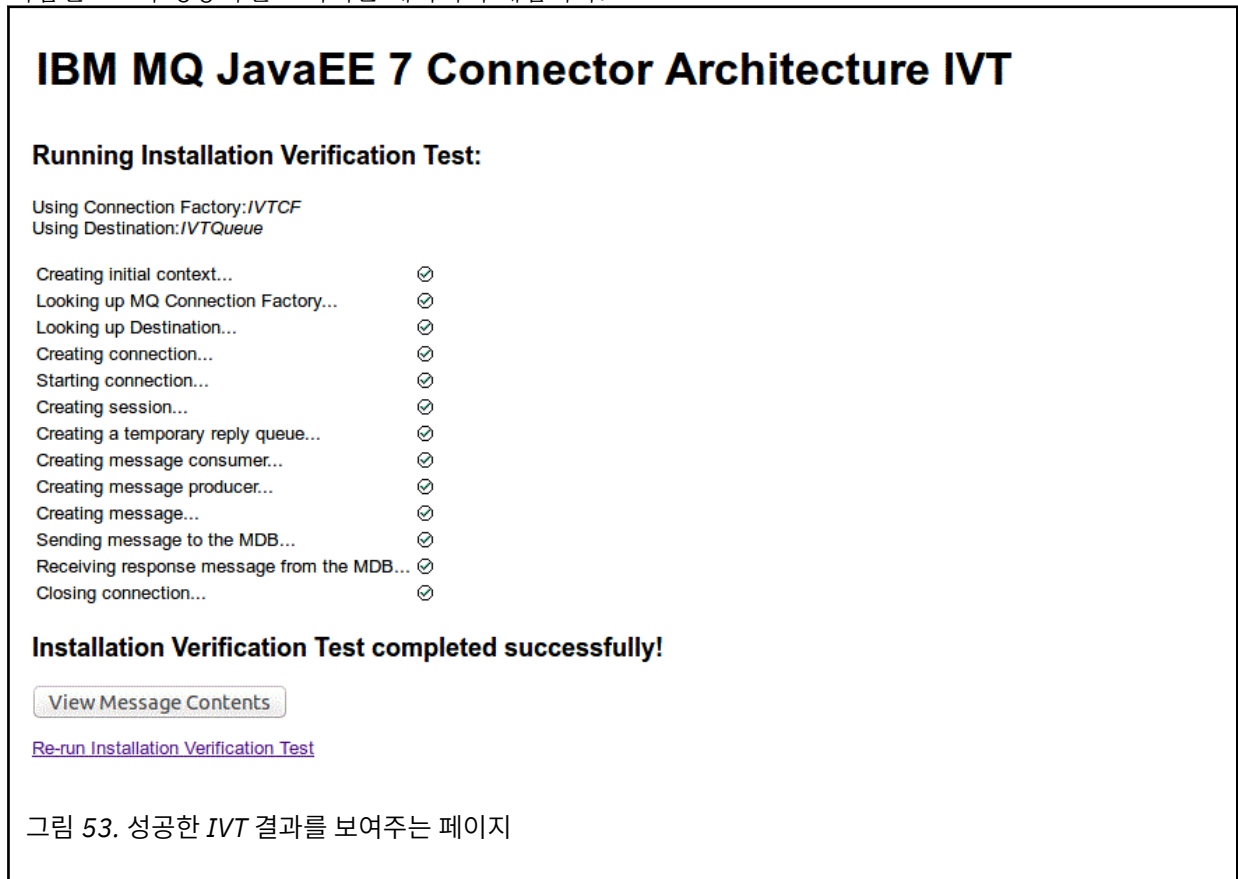


그림 53. 성공한 IVT 결과를 보여주는 페이지

다음은 IVT가 실패할 경우 표시되는 페이지의 예입니다. 실패 원인에 대한 추가 정보를 얻으려면 **스택 추적 보기**를 클릭하십시오.

IBM MQ JavaEE 7 Connector Architecture IVT

Running Installation Verification Test:

Using Connection Factory:IVTCF
Using Destination:IVTQueue

Creating initial context...	☑
Looking up MQ Connection Factory...	☑
Looking up Destination...	☑
Creating connection...	☑
Starting connection...	☑
Creating session...	☑
Creating a temporary reply queue...	☑
Creating message consumer...	☑
Creating message producer... failed to create message producer!	❌

Installation Verification Test failed!

Error received - JMS Exception:

com.ibm.msg.client.jms.DetailedJMSSecurityException: JMSWMQ2008: Failed to open MQ queue 'TEST.QUEUE'.
JMS attempted to perform an MQOPEN, but IBM MQ reported an error.
Use the linked exception to determine the cause of this error. Check that the specified queue and queue manager are defined correctly.

[View Stack Trace](#)

Installation Verification Test failed!

[Retry Installation Verification Test](#)
[Change IVT parameters](#)

그림 54. 실패한 IVT 결과를 보여주는 페이지

JM 3.0

WebSphere Liberty IBM MQ 메시징 공급자를 사용하여 배포된 Jakarta Messaging 설치에 대한 IBM MQ 리소스 어댑터 확인

Jakarta Messaging 설치 검증 테스트(IVT) 프로그램용 IBM MQ 리소스 어댑터는 EAR(엔터프라이즈 아카이브) 파일 `wmq.jakarta.jmsra.ivt.ear`에 제공됩니다. 이 파일은 Jakarta Messaging 리소스 어댑터 RAR 파일 `wmq.jakarta.jmsra.rar`에 대한 IBM MQ 과 같은 디렉토리에 IBM MQ classes for Jakarta Messaging 와 함께 설치됩니다.

이 태스크 정보

`wmq.jakarta.jmsra.ivt.ear` 및 `wmq.jakarta.jmsra.rar` 파일이 설치되는 위치에 대한 자세한 내용은 398 페이지의 『리소스 IBM MQ 어댑터 설치』을 참조하세요.

IVT 프로그램에는 IBM MQ 대기열로 메시지를 보내고 받을 수 있는지 테스트하는 서블릿과 메시지 기반 빈(MDB)이 포함되어 있습니다. IVT 프로그램을 사용하여 Jakarta Messaging 용 IBM MQ 리소스 어댑터가 WebSphere Liberty IBM MQ 메시징 공급자를 사용하여 WebSphere Liberty 에 올바르게 배포되었는지, 분산 트랜잭션을 지원하도록 구성되었는지 확인할 수 있습니다.

IVT 프로그램을 실행하려면 먼저 Jakarta Messaging 관리 객체 3개를 정의해야 합니다:

- 대기열 관리자의 대기열에 매핑되는 대기열 대상입니다.
- 대기열 관리자에 연결하는 방법에 대한 세부 정보가 포함된 연결 팩토리 개체입니다. 이는 IVT의 서블릿 부분에서 사용됩니다.
- 활성화 사양입니다. 여기에는 대기열 관리자에 연결하는 방법에 대한 정보가 포함되어 있으며 메시지를 모니터링 대기열을 알 수 있도록 대기열 대상에 대한 참조가 포함되어야 합니다. 이는 IVT의 MDB 부분에서 사용됩니다.

이러한 개체의 속성은 대기열 관리자에 따라 다릅니다. 다음은 간단한 예제입니다.

Queue destination:	
baseQueueName:	TEST.QUEUE
Connection factory:	
channel:	SYSTEM.DEF.SVRCONN

```

hostName:      localhost
port:          1414
queueManager:  QM1
transportType: CLIENT

Activation specification:
channel:       SYSTEM.DEF.SVRCONN
hostName:     localhost
port:         1414
queueManager: QM1
transportType: CLIENT

```

참고:

1. 다음 단계를 수행하기 전에 WebSphere Liberty IBM MQ 메시징 공급자를 사용하여 Jakarta Messaging 용 IBM MQ 리소스 어댑터를 WebSphere Liberty 에 설치해야 합니다. 자세한 정보는 398 페이지의 『리소스 IBM MQ 어댑터 설치』 및 393 페이지의 『메시징 WebSphere Liberty IBM MQ 공급자와 함께 Jakarta Messaging 리소스 IBM MQ 어댑터 사용』의 내용을 참조하십시오.
2. 대기열 관리자의 보안 구성에 따라 IVT 애플리케이션이 실행될 때 이유 코드 2035(MQRC_NOT_AUTHORIZED)가 포함된 오류를 보고할 수 있습니다. 자세한 내용은 2035(07F3) (RC2035): MQRC_NOT_AUTHORIZED를 참조하세요.

프로시저

1. WebSphere Liberty 서버를 중지하십시오.
2. Jakarta Messaging IVT 프로그램용 IBM MQ 리소스 어댑터 `wmq.jakarta.jmsra.ivt.ear` 를 `<server.config.dir>/dropins` 디렉터리에 복사합니다. 여기서 `<server.config.dir>` 는 WebSphere Liberty 서버의 구성 디렉터리입니다.
3. WebSphere Liberty `server.xml` 파일을 편집하고 다음 예시와 같이 항목을 추가합니다:

```

<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
  <properties.wmqJms channel="SYSTEM.DEF.SVRCONN"
                    hostname="localhost"
                    port="1414"
                    queueManager="QM1"
                    transportType="CLIENT"/>
</jmsQueueConnectionFactory>

<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queue -->
<jmsQueue id="IVT-Queue" jndiName="IVTQueue">
  <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jakarta.jmsra.ivt/WMQ_JAKARTA_IVT_MDB/WMQ_JAKARTA_IVT_MDB">
  <properties.wmqJms channel="SYSTEM.DEF.SVRCONN"
                    destinationRef="IVT-Queue"
                    hostName="localhost"
                    maxPoolDepth="1"
                    port="1414"
                    queueManager="QM1"
                    transportType="CLIENT"/>
</jmsActivationSpec>

```

참고: 이러한 정의는 앞서 표시된 [예제 값](#)을 사용합니다. 대기열 관리자와 대기열의 값이 다른 경우 정의에서 해당 값을 사용해야 합니다.

4. `server.xml` 파일의 `<featureManager>` 섹션에 IVT 프로그램에 필요한 기능에 대한 항목이 포함되어 있는지 확인합니다. 기능은 사용 중인 Jakarta EE 버전에 따라 다릅니다:

표 69. IVT 프로그램에 필요한 기능	
Jakarta EE 버전	server.xml 파일의 <featureManager> 섹션에 추가해야 하는 항목은 다음과 같습니다
Jakarta EE 9	<pre><feature>jakartaee-9.1</feature> <feature>jndi-1.0</feature> <feature>mdb-4.0</feature> <feature>pages-3.0</feature> <feature>wmqMessagingClient-3.0</feature></pre>
Jakarta EE 10	<pre><feature>jakartaee-10.0</feature> <feature>jndi-1.0</feature> <feature>mdb-4.0</feature> <feature>pages-3.1</feature> <feature>wmqMessagingClient-3.0</feature></pre>

5. WebSphere Liberty을(를) 시작하십시오.
6. 다음 형식의 URL을 웹 브라우저에 입력하여 IVT 프로그램을 시작하십시오.

```
http://<app_server_host>:<port>/WMQ_JAKARTA_IVT/
```

여기서 *app_server_host* 는 애플리케이션 서버가 실행 중인 시스템의 IP 주소 또는 호스트 이름이며, *port* 는 애플리케이션 서버가 리스닝 중인 TCP 포트의 번호입니다. 다음은 예제입니다.

```
http://localhost:9080/WMQ_JAKARTA_IVT/
```

이제 IVT 프로그램의 초기 페이지가 표시됩니다.

IBM MQ JakartaEE 9.1 Connector Architecture IVT

Tue Feb 11 10:26:38 GMT 2025

9.4.0.5-p940-005-240823

Installation Verification Test

Check to ensure that the IBM MQ J2EE Connector Architecture resource adapter is correctly installed.

Connection Factory:

Destination:

그림 55. IVT 프로그램의 초기 페이지

7. 테스트를 실행하려면 **IVT 실행**을 클릭하십시오.
- 다음은 IVT가 성공하면 표시되는 페이지의 예입니다.

IBM MQ Jakarta EE 9 Connector Architecture IVT

Tue Feb 11 10:27:30 GMT 2025

Running Installation Verification Test:

Using Connection Factory: *IVTCF*
Using Destination: *IVTQueue*

Creating initial context...	☑
Looking up MQ Connection Factory...	☑
Looking up Destination...	☑
Creating connection...	☑
Starting connection...	☑
Creating session...	☑
Creating a temporary reply queue...	☑
Creating message consumer...	☑
Creating message producer...	☑
Creating message...	☑
Sending message to the MDB...	☑
Receiving response message from the MDB...	☑
Closing connection...	☑

Installation Verification Test completed successfully!

[View Message Contents](#)

[Re-run Installation Verification Test](#)

그림 56. 성공한 IVT 결과를 보여주는 페이지

다음은 IVT가 실패할 경우 표시되는 페이지의 예입니다. 실패 원인에 대한 추가 정보를 얻으려면 [스택 추적 보기](#)를 클릭하십시오.

IBM MQ Jakarta EE 9 Connector Architecture IVT

Tue Feb 11 13:02:33 GMT 2025

Running Installation Verification Test:

Using Connection Factory: *IVTCF*
Using Destination: *IVTQueue*

Creating initial context...	☑
Looking up MQ Connection Factory...	☑
Looking up Destination...	☑
Creating connection...	☑
Starting connection...	☑
Creating session...	☑
Creating a temporary reply queue...	☑
Creating message consumer...	☑
Creating message producer... failed to create message producer!	☒

Installation Verification Test failed!

Error received - JMS Exception:

com.ibm.msg.client.jakarta.jms.DetailedInvalidDestinationException: JMSWMQ2008: Failed to open MQ queue 'TEST.QUEUE'.

JMS attempted to perform an MQOPEN, but IBM MQ reported an error.

Use the linked exception to determine the cause of this error. Check that the specified queue and queue manager are defined correctly.

[View Stack Trace](#)

Installation Verification Test failed!

[Retry Installation Verification Test](#)
[Change IVT parameters](#)

그림 57. 실패한 IVT 결과를 보여주는 페이지

▶ JM 3.0 에 일반 리소스 어댑터로 배포된 Jakarta Messaging 설치에 대한 IBM MQ 리소스 어댑터 확인 WebSphere Liberty

Jakarta Messaging 설치 검증 테스트(IVT) 프로그램용 IBM MQ 리소스 어댑터는 EAR(엔터프라이즈 아카이브) 파일 `wmq.jakarta.jmsra.ivt.ear`에 제공됩니다. 이 파일은 Jakarta Messaging 리소스 어댑터 RAR 파일 `wmq.jakarta.jmsra.rar`에 대한 IBM MQ 과 동일한 디렉토리에 IBM MQ classes for Jakarta Messaging 와 함께 설치됩니다.

이 태스크 정보

`wmq.jakarta.jmsra.ivt.ear` 및 `wmq.jakarta.jmsra.rar` 파일이 설치되는 위치에 대한 자세한 내용은 398 페이지의 『리소스 IBM MQ 어댑터 설치』을 참조하세요.

IVT 프로그램에는 IBM MQ 대기열로 메시지를 보내고 받을 수 있는지 테스트하는 서블릿과 메시지 기반 빈(MDB)이 포함되어 있습니다. IVT 프로그램을 사용하여 Jakarta Messaging 용 IBM MQ 리소스 어댑터가 일반 리소스 어댑터로 WebSphere Liberty 에 올바르게 배포되었으며 분산 트랜잭션을 지원하도록 구성되었는지 확인할 수 있습니다.

IVT 프로그램을 실행하려면 먼저 Jakarta Messaging 관리 객체 3개를 정의해야 합니다:

- 대기열 관리자의 대기열에 매핑되는 대기열 대상입니다.
- 대기열 관리자에 연결하는 방법에 대한 세부 정보가 포함된 연결 팩토리 개체입니다. 이는 IVT의 서블릿 부분에서 사용됩니다.
- 활성화 사양입니다. 여기에는 대기열 관리자에 연결하는 방법에 대한 정보가 포함되어 있으며 메시지를 모니터링할 대기열을 알 수 있도록 대기열 대상에 대한 참조가 포함되어야 합니다. 이는 IVT의 MDB 부분에서 사용됩니다.

이러한 개체의 속성은 대기열 관리자에 따라 다릅니다. 다음은 간단한 예제입니다.

```
Queue destination:
  baseQueueName:      TEST.QUEUE

Connection factory:
  channel:             SYSTEM.DEF.SVRCONN
  hostName:            localhost
  port:                1414
  queueManager:        QM1
  transportType:       CLIENT

Activation specification:
  channel:             SYSTEM.DEF.SVRCONN
  hostName:            localhost
  port:                1414
  queueManager:        QM1
  transportType:       CLIENT
```

참고:

1. 다음 단계를 수행하기 전에 Jakarta Messaging 용 IBM MQ 리소스 어댑터를 일반 리소스 어댑터로 WebSphere Liberty 에 설치해야 합니다. 자세한 정보는 398 페이지의 『리소스 IBM MQ 어댑터 설치』 및 396 페이지의 『를 일반 리소스 IBM MQ Jakarta Messaging 어댑터로 사용하기 위한 리소스 어댑터』의 내용을 참조하십시오.
2. 대기열 관리자의 보안 구성에 따라 IVT 애플리케이션이 실행될 때 이유 코드 2035(MQRC_NOT_AUTHORIZED)가 포함된 오류를 보고할 수 있습니다. 자세한 내용은 2035(07F3) (RC2035): MQRC_NOT_AUTHORIZED를 참조하세요.
3. ▶ **z/OS** Jakarta Messaging 에 대한 IBM MQ 리소스 어댑터를 일반 리소스 어댑터로 WebSphere Liberty 에 설치하는 것은 z/OS 에서 지원되지 않습니다.

프로시저

1. WebSphere Liberty 서버를 중지하십시오.

2. Jakarta Messaging IVT 프로그램용 IBM MQ 리소스 어댑터 `wmq.jakarta.jmsra.ivt.ear` 를 `<server.config.dir>/dropins` 디렉터리에 복사합니다. 여기서 `<server.config.dir>` 는 WebSphere Liberty 서버의 구성 디렉터리입니다.
3. WebSphere Liberty `server.xml` 파일을 편집하고 다음 예시와 같이 항목을 추가합니다:

```
<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
  <properties.<resource_adapter_id>
    channel="SYSTEM.DEF.SVRCONN"
                                hostname="localhost"
                                port="1414"
                                queueManager="QM1"
                                transportType="CLIENT"/>
  </properties>
</jmsQueueConnectionFactory>

<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queue -->
<jmsQueue id="IVT-Queue" jndiName="IVTQueue">
  <properties.<resource_adapter_id> baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jakarta.jmsra.ivt/WMQ_JAKARTA_IVT_MDB/WMQ_JAKARTA_IVT_MDB">
  <properties.<resource_adapter_id> channel="SYSTEM.DEF.SVRCONN"
                                destinationRef="IVT-Queue"
                                destinationType="jakarta.jms.queue"
                                hostName="localhost"
                                maxPoolDepth="1"
                                port="1414"
                                queueManager="QM1"
                                transportType="CLIENT"/>
  </properties>
</jmsActivationSpec>
```

여기서 `<resource_adapter_id>` 은 `server.xml` 파일에서 Jakarta Messaging 에 대한 IBM MQ 리소스 어댑터를 정의하는 `resourceadapter` 요소의 `id` 속성 값입니다.

참고: 이러한 정의는 앞서 표시된 [예제 값](#)을 사용합니다. 대기열 관리자와 대기열의 값이 다른 경우 정의에서 해당 값을 사용해야 합니다.

4. `server.xml` 파일의 `<featureManager>` 섹션에 IVT 프로그램에 필요한 기능에 대한 항목이 포함되어 있는지 확인합니다. 기능은 사용 중인 Jakarta EE 버전에 따라 다릅니다:

표 70. IVT 프로그램에 필요한 기능	
Jakarta EE 버전	<code>server.xml</code> 파일의 <code><featureManager></code> 섹션에 추가해야 하는 항목은 다음과 같습니다
Jakarta EE 9	<pre><feature>jndi-1.0</feature> <feature>jakartaee-9.1</feature> <feature>pages-3.0</feature> <feature>mdb-4.0</feature></pre>
Jakarta EE 10	<pre><feature>jndi-1.0</feature> <feature>jakartaee-10.0</feature> <feature>pages-3.1</feature> <feature>mdb-4.0</feature></pre>

5. WebSphere Liberty을(를) 시작하십시오.
6. 다음 형식의 URL을 웹 브라우저에 입력하여 IVT 프로그램을 시작하십시오.

```
http://<app_server_host>:<port>/WMQ_JAKARTA_IVT/
```

여기서 `app_server_host` 는 애플리케이션 서버가 실행 중인 시스템의 IP 주소 또는 호스트 이름이며, `port` 는 애플리케이션 서버가 리스닝 중인 TCP 포트의 번호입니다. 다음은 예제입니다.

```
http://localhost:9080/WMQ_JAKARTA_IVT/
```

이제 IVT 프로그램의 초기 페이지가 표시됩니다.

IBM MQ JakartaEE 9.1 Connector Architecture IVT

Tue Feb 11 10:26:38 GMT 2025

9.4.0.5-p940-005-240823

Installation Verification Test

Check to ensure that the IBM MQ J2EE Connector Architecture resource adapter is correctly installed.

Connection Factory:

Destination:

그림 58. IVT 프로그램의 초기 페이지

7. 테스트를 실행하려면 **IVT 실행**을 클릭하십시오.

다음은 IVT가 성공하면 표시되는 페이지의 예입니다.

IBM MQ Jakarta EE 9 Connector Architecture IVT

Tue Feb 11 10:27:30 GMT 2025

Running Installation Verification Test:

Using Connection Factory: *IVTCF*
Using Destination: *IVTQueue*

Creating initial context...	☑
Looking up MQ Connection Factory...	☑
Looking up Destination...	☑
Creating connection...	☑
Starting connection...	☑
Creating session...	☑
Creating a temporary reply queue...	☑
Creating message consumer...	☑
Creating message producer...	☑
Creating message...	☑
Sending message to the MDB...	☑
Receiving response message from the MDB...	☑
Closing connection...	☑

Installation Verification Test completed successfully!

[Re-run Installation Verification Test](#)

그림 59. 성공한 IVT 결과를 보여주는 페이지

다음은 IVT가 실패할 경우 표시되는 페이지의 예입니다. 실패 원인에 대한 추가 정보를 얻으려면 **스택 추적 보기**를 클릭하십시오.

IBM MQ Jakarta EE 9 Connector Architecture IVT

Tue Feb 11 13:02:33 GMT 2025

Running Installation Verification Test:

Using Connection Factory: IVTCF
Using Destination: IVTQueue

```
Creating initial context... ☺  
Looking up MQ Connection Factory... ☺  
Looking up Destination... ☺  
Creating connection... ☺  
Starting connection... ☺  
Creating session... ☺  
Creating a temporary reply queue... ☺  
Creating message consumer... ☺  
Creating message producer... failed to create message producer! ☹
```

Installation Verification Test failed!

Error received - JMS Exception:

com.ibm.msg.client.jakarta.jms.DetailedInvalidDestinationException: JMSMQ2008: Failed to open MQ queue 'TEST.QUEUE'.

JMS attempted to perform an MQOPEN, but IBM MQ reported an error.

Use the linked exception to determine the cause of this error. Check that the specified queue and queue manager are defined correctly.

[View Stack Trace](#)

Installation Verification Test failed!

[Retry Installation Verification Test](#)
[Change IVT parameters](#)

그림 60. 실패한 IVT 결과를 보여주는 페이지

▶ JM 3.0 다른 Java EE 애플리케이션 서버에 Jakarta Messaging 설치를 위한 IBM MQ 리소스 어댑터 확인

Jakarta Messaging 설치 검증 테스트(IVT) 프로그램용 IBM MQ 리소스 어댑터는 EAR(엔터프라이즈 아카이브) 파일 `wmq.jakarta.jmsra.ivt.ear`에 제공됩니다. 이 파일은 Jakarta Messaging 리소스 어댑터 RAR 파일 `wmq.jakarta.jmsra.rar`에 대한 IBM MQ과 같은 디렉토리에 IBM MQ classes for Jakarta Messaging와 함께 설치됩니다.

이 태스크 정보

`wmq.jakarta.jmsra.ivt.ear` 및 `wmq.jakarta.jmsra.rar` 파일이 설치되는 위치에 대한 자세한 내용은 398 페이지의 『리소스 IBM MQ 어댑터 설치』를 참조하세요.

IVT 프로그램에는 IBM MQ 대기열로 메시지를 보내고 받을 수 있는지 테스트하는 서블릿과 메시지 기반 빈(MDB)이 포함되어 있습니다. IVT 프로그램을 사용하여 IBM MQ 리소스 어댑터(Jakarta Messaging)가 애플리케이션 서버에 올바르게 배포되었고 분산 트랜잭션을 지원하도록 구성되었는지 확인할 수 있습니다. 또한 IBM 지원팀에서 애플리케이션 서버가 올바르게 구성되었는지 확인하기 위해 IVT 프로그램을 실행하도록 요청할 수도 있습니다.

IVT 프로그램을 실행하려면 먼저 Jakarta Messaging 관리 객체 3개를 정의해야 합니다:

- 대기열 관리자의 대기열에 매핑되는 대기열 대상입니다.
- 대기열 관리자에 연결하는 방법에 대한 세부 정보가 포함된 연결 팩토리 개체입니다. 이는 IVT의 서블릿 부분에서 사용됩니다.
- 활성화 사양입니다. 여기에는 대기열 관리자에 연결하는 방법에 대한 정보가 포함되어 있으며 메시지를 모니터링할 대기열을 알 수 있도록 대기열 대상에 대한 참조가 포함되어야 합니다. 이는 IVT의 MDB 부분에서 사용됩니다.

이러한 개체의 속성은 대기열 관리자에 따라 다릅니다. 다음은 간단한 예제입니다.

```
Queue destination:  
baseQueueName: TEST.QUEUE
```

```

Connection factory:
channel:          SYSTEM.DEF.SVRCONN
hostName:         localhost
port:            1414
queueManager:    QM1
transportType:    CLIENT

Activation specification:
channel:          SYSTEM.DEF.SVRCONN
hostName:         localhost
port:            1414
queueManager:    QM1
transportType:    CLIENT

```

이러한 객체를 정의하는 데 사용되는 메커니즘은 애플리케이션 서버에 따라 다릅니다. 이 작업을 수행하는 방법은 애플리케이션 서버 설명서를 참조하세요.

IVT 프로그램은 연결 팩토리 개체가 애플리케이션 서버의 JNDI 네임스페이스에 `jms/ivt/IVTCF` 이라는 이름으로 바인딩되고 큐 대상 개체는 `jms/ivt/IVTQueue` 이라는 이름으로 바인딩될 것으로 예상합니다. 다른 이름을 사용할 수 있지만 그런 경우 IVT 프로그램의 초기 페이지에서 오브젝트 이름을 입력하고 EAR 파일을 적절하게 수정해야 합니다.

IVT 프로그램을 배치하고 애플리케이션 서버에서 Jakarta Messaging 오브젝트를 작성한 다음 JNDI 네임스페이스에 바인딩하고 나면 다음 단계를 완료하여 IVT 프로그램을 시작할 수 있습니다.

참고:

1. 이 단계에서는 Jakarta Messaging 용 IBM MQ 리소스 어댑터가 이미 애플리케이션 서버에 설치되어 있다고 가정합니다. 자세한 정보는 [398 페이지의 『리소스 IBM MQ 어댑터 설치』](#)의 내용을 참조하십시오.
2. 대기열 관리자의 보안 구성에 따라 IVT 애플리케이션이 실행될 때 이유 코드 `2035(MQRC_NOT_AUTHORIZED)`가 포함된 오류를 보고할 수 있습니다. 자세한 내용은 [2035\(07F3\) \(RC2035\): MQRC_NOT_AUTHORIZED](#)를 참조하세요.

프로시저

1. 다음 형식의 URL을 웹 브라우저에 입력하여 IVT 프로그램을 시작하십시오.

```
http://<app_server_host>:<port>/WMQ_JAKARTA_IVT/
```

여기서 `app_server_host` 는 애플리케이션 서버가 실행 중인 시스템의 IP 주소 또는 호스트 이름이며, `port` 는 애플리케이션 서버가 리스닝 중인 TCP 포트의 번호입니다. 다음은 예제입니다.

```
http://localhost:9080/WMQ_JAKARTA_IVT/
```

이제 IVT 프로그램의 초기 페이지가 표시됩니다.

IBM MQ JakartaEE 9.1 Connector Architecture IVT

Tue Feb 11 10:26:38 GMT 2025

9.4.0.5-p940-005-240823

Installation Verification Test

Check to ensure that the IBM MQ J2EE Connector Architecture resource adapter is correctly installed.

Connection Factory:	IVTCF
Destination:	IVTQueue
Run IVT Reset Values	

그림 61. IVT 프로그램의 초기 페이지

2. 테스트를 실행하려면 **IVT 실행**을 클릭하십시오.

다음은 IVT가 성공하면 표시되는 페이지의 예입니다.

IBM MQ Jakarta EE 9 Connector Architecture IVT

Tue Feb 11 10:27:30 GMT 2025

Running Installation Verification Test:

Using Connection Factory: *IVTCF*

Using Destination: *IVTQueue*

Creating initial context...	☑
Looking up MQ Connection Factory...	☑
Looking up Destination...	☑
Creating connection...	☑
Starting connection...	☑
Creating session...	☑
Creating a temporary reply queue...	☑
Creating message consumer...	☑
Creating message producer...	☑
Creating message...	☑
Sending message to the MDB...	☑
Receiving response message from the MDB...	☑
Closing connection...	☑

Installation Verification Test completed successfully!

[View Message Contents](#)

[Re-run Installation Verification Test](#)

그림 62. 성공한 IVT 결과를 보여주는 페이지

다음은 IVT가 실패할 경우 표시되는 페이지의 예입니다. 실패 원인에 대한 추가 정보를 얻으려면 **스택 추적 보기**를 클릭하십시오.

IBM MQ Jakarta EE 9 Connector Architecture IVT

Tue Feb 11 13:02:33 GMT 2025

Running Installation Verification Test:

Using Connection Factory:IVTCF
Using Destination:IVTQueue

Creating initial context...	☺
Looking up MQ Connection Factory...	☺
Looking up Destination...	☺
Creating connection...	☺
Starting connection...	☺
Creating session...	☺
Creating a temporary reply queue...	☺
Creating message consumer...	☺
Creating message producer... failed to create message producer!	☹

Installation Verification Test failed!

Error received - JMS Exception:

com.ibm.msg.client.jakarta.jms.DetailedInvalidDestinationException: JMSWMQ2008: Failed to open MQ queue 'TEST.QUEUE'.

JMS attempted to perform an MQOPEN, but IBM MQ reported an error.

Use the linked exception to determine the cause of this error. Check that the specified queue and queue manager are defined correctly.

[View Stack Trace](#)

Installation Verification Test failed!

[Retry Installation Verification Test](#)
[Change IVT parameters](#)

그림 63. 실패한 IVT 결과를 보여주는 페이지

Windows GlassFish 서버 및 IBM MQ 리소스 어댑터

리소스 IBM MQ 어댑터는 운영 Windows 체제 상의 GlassFish 서버에 설치할 수 있습니다. 설치 검증 테스트 (IVT) 애플리케이션을 설치하고 실행하여 설치를 테스트할 수 있습니다

Windows JMS 2.0 GlassFish 서버에서 리소스 IBM MQ 어댑터 설치 및 테스트

GlassFish Server를 IBM MQ 운영 Windows 체제에 설치하려면 먼저 도메인을 생성하고 시작해야 합니다. 그런 다음 자원 어댑터를 배치하고 구성할 수 있으며 설치 확인 테스트(IVT) 애플리케이션을 배치하고 실행할 수 있습니다.

시작하기 전에

- 이 지시사항은 GlassFish Server 버전 4용입니다.
- **JM 3.0** 이 버전의 GlassFish 서버는,을 Jakarta EE지원하지 않습니다. **4.0.0**에 대한 지원에 관한 자세한 Jakarta EE 내용은 을 참조하십시오 464 페이지의 『GlassFish 서버용 Jakarta Messaging 리소스 어댑터 IBM MQ 설치 및 테스트』.

이 태스크 정보

이 태스크는 GlassFish Server 애플리케이션 서버가 실행 중이며 사용자가 이에 대한 표준 관리 태스크를 잘 알고 있다고 가정합니다. 이 태스크는 또한 로컬 시스템에 IBM MQ가 설치되어 있으며 사용자가 표준 관리 태스크를 잘 알고 있다고 가정합니다.

참고: 다음 태스크 단계를 완료하려면 다음 오브젝트가 구성된 IBM MQ 설치가 작동 중이어야 합니다.

- 1414 포트에서 시작되고 SYSTEM.DEF.SVRCONN 채널을 사용하며 클라이언트 전송을 사용하여 연결되는 QM이라는 큐 관리자
- Q1이라는 큐

프로시저

1. GlassFish Server **asadmin** 셸 프로그램을 시작하십시오.

- a) Windows 명령행을 열고 *GlassFish/bin* 디렉토리로 이동하십시오. 여기서 *GlassFish*는 GlassFish Server 버전 4가 설치된 디렉토리입니다.
- b) 명령행에서 **asadmin** 명령을 입력하십시오.

asadmin 명령은 새 도메인을 작성할 수 있는 셸 프로그램을 명령행에서 엽니다.

GlassFish Server 버전 4가 시스템에서 시작됩니다.

2. 도메인을 작성한 후에 이를 시작하십시오.

- a) 포트 및 도메인 이름을 지정하여 **create-domain** 명령을 사용하여 새 도메인을 작성하십시오. 명령행에서 다음 명령을 입력하십시오.

```
create-domain --adminport <port> <domain_name>
```

여기서 *port*는 포트 번호이고 *domain_name*은 도메인에서 사용할 이름입니다.

참고: **create-domain** 명령에는 이와 연관된 다수의 선택적 매개변수가 있습니다. 그러나 이 태스크에는 **--adminport** 매개변수만 필요합니다. 자세한 정보는 GlassFish Server 버전 4의 제품 문서를 참조하십시오.

지정한 포트가 사용 중인 경우 다음 메시지가 표시됩니다.

< 도메인_이름 >의 < 포트 >가 사용 중입니다

지정한 도메인 이름이 사용 중이면 현재 사용 불가능한 모든 도메인 이름의 목록과 함께 지정된 이름이 이미 사용 중임을 알리는 메시지를 수신합니다.

- b) 사용자 이름 및 비밀번호를 입력하라는 프롬프트가 나타나면 웹 브라우저를 통해 애플리케이션 서버에 로그인하기 위해 사용되는 신임 정보를 입력하십시오.

명령이 성공적으로 완료되면 도메인 생성 요약 메시지가 명령줄에 표시되며, 다음 메시지를 포함합니다
명령어 create-domain이 성공적으로 실행되었습니다

도메인 작성이 완료되었습니다.

- c) 명령행에 다음 명령을 입력하여 도메인을 시작하십시오.

```
start-domain <domain_name>
```

여기서 *domain_name*은 사용자가 이전에 지정했던 도메인 이름입니다.

3. 웹 브라우저를 사용하여 GlassFish 애플리케이션 서버에 액세스하십시오.

- a) 웹 브라우저의 주소 표시줄에서 다음 명령을 입력하십시오.

```
localhost:<port>
```

여기서 *port*는 도메인을 작성할 때 이전에 지정한 포트입니다.

GlassFish 콘솔이 표시됩니다.

- b) GlassFish 콘솔이 로드되고 사용자 이름과 암호를 입력하라는 메시지가 표시되면, 단계에서 지정한 자격 증명을 462 페이지의 『2.b』 입력하십시오.

4. 자원 어댑터를 GlassFish Server 4에 업로드하십시오.

- a) 공통 작업 도구 모음에서 **애플리케이션** 메뉴 항목을 선택하여 **애플리케이션** 페이지를 표시합니다.
- b) 배치 단추를 클릭하여 **애플리케이션 또는 모듈 배치** 페이지를 여십시오.
- c) **찾아보기** 버튼을 클릭한 후 파일의 *wmq.jmsra.rar* 위치로 이동하십시오. 파일을 선택한 후 **확인**을 클릭하십시오.

5. 연결 풀을 작성하십시오.

- a) 도구 모음의 **자원** 아래에서 **커넥터** 메뉴 항목을 선택하십시오.
- b) 그런 다음 **커넥터 연결 풀** 메뉴 항목을 선택하여 **커넥터 연결 풀** 페이지를 엽니다.
- c) **새로 작성**을 클릭하여 **새 커넥터 연결 풀(1/2단계)** 페이지를 여십시오.

- d) 새 커넥터 연결 풀(2단계 중 1단계) 페이지에서 풀 이름 필드에 풀 jms/ivt/IVTCF-Connection-Pool 이름을 입력하십시오.
 - e) 리소스 어댑터 필드에서 를 선택하십시오 wmq.jmsra.
 - f) 연결 정의 필드에 을 입력하십시오 javax.jms.ConnectionFactory.
 - g) 다음을 선택한 후에 완료를 선택하십시오.
6. 커넥터 자원을 작성하십시오.
- a) 도구 모음의 커넥터 메뉴에서 커넥터 리소스 옵션을 선택하여 커넥터 리소스 페이지를 엽니다.
 - b) 새로 만들기를 선택하여 새 커넥터 리소스 페이지를 엽니다.
 - c) JNDI 이름 필드에서 IVTCF를 입력하십시오.
 - d) 풀 이름 필드에서 jms/ivt/IVTCF-Connection-Pool을 입력하십시오.
 - e) 다른 모든 필드는 비워 두십시오.
 - f) 다음 속성/값 쌍 각각에 대해 '속성 추가'를 클릭하고 다음 예시와 같이 속성 이름과 값을 입력하십시오:

```
name: host; value: localhost
name: port; value: 1414
name: channel; value: SYSTEM.DEF.SVRCONN
name: queueManager; value: QM
name: transportType; value: CLIENT
```

참고: 자체 구성 설정에 대해 올바른 값을 사용 중인지 확인하십시오. 이는 이 예제에서 표시된 것과 다를 수 있습니다.

- g) 도구 모음의 '커넥터' 아래에서 '관리자 객체 리소스' 메뉴 항목을 선택하여 관리자 객체 리소스 페이지를 엽니다.
- h) 관리 오브젝트 자원 페이지에서 새로 작성을 클릭하여 새 관리 오브젝트 자원 페이지를 여십시오.
- i) JNDI 이름 필드에서 IVTQueue를 입력하십시오.
- j) 자원 어댑터 필드에서 wmq.jmsra를 입력하십시오.
- k) 자원 유형 필드에서 javax.jms.Queue를 입력하십시오.
- l) 클래스 이름 필드는 그대로 두십시오.
- m) 다음의 특성/값 쌍 각각에 대해 특성 추가를 클릭하고 다음 예에 표시된 대로 특성 이름 및 값을 입력하십시오.

```
name: name; value: IVTQueue
name: baseQueueManagerName; value QM
name: baseQueueName; value: Q1
```

참고: 자체 구성 설정에 대해 올바른 값을 사용 중인지 확인하십시오. 이는 이 예제에서 표시된 것과 다를 수 있습니다.

- n) 확인을 클릭하십시오.
 - o) 활성화 확인란을 선택한 다음 활성화를 클릭하십시오.
7. EAR 파일 wmq.jmsra.ivt.ear을(를) GlassFish 서버에 배치하십시오.
- a) 도구 모음에서 애플리케이션 옵션을 클릭하여 애플리케이션 페이지를 표시하십시오.
 - b) 배치를 클릭하여 IVT 애플리케이션을 추가하십시오.
 - c) 위치 필드에서 wmq.jmsra.ivt.ear을(를) 탐색하고 선택하십시오.
 - d) 가상 서버 필드에서 서버를 선택한 후에 확인을 클릭하십시오.
8. IVT 프로그램을 실행하십시오.
- a) 도구 모음에서 애플리케이션 옵션을 클릭하여 애플리케이션 페이지를 표시하십시오.
 - b) 배포된 애플리케이션 테이블에서 wmq.jmsra.ivt 클릭하세요.
 - c) 모듈 및 컴포넌트 테이블에서 시작 단추를 클릭하십시오.
 - d) http: 링크를 선택하십시오.

e) **IVT 실행**을 클릭하십시오.

IVT 프로그램이 시작되었으며, 성공하면 다음 출력이 표시됩니다.

Running Installation Verification Test:

Using Connection Factory: *IVTCF*
Using Destination: *IVTQueue*

Creating initial context...	☑
Looking up MQ Connection Factory...	☑
Looking up Destination...	☑
Creating connection...	☑
Starting connection...	☑
Creating session...	☑
Creating a temporary reply queue...	☑
Creating message consumer...	☑
Creating message producer...	☑
Creating message...	☑
Sending message to the MDB...	☑
Receiving response message from the MDB...	☑
Closing connection...	☑

Installation Verification Test completed successfully!

[View Message Contents](#)

[Re-run Installation Verification Test](#)

그림 64. 성공적인 IVT 출력

Windows JM 3.0 GlassFish 서버용 Jakarta Messaging 리소스 어댑터 IBM MQ 설치 및 테스트

GlassFish 서버 7.0 은 Jakarta EE 기반 리소스 어댑터만 지원합니다. GlassFish Server(7.0) Jakarta Messaging 용 리소스 IBM MQ 어댑터를 운영 Windows 체제에 설치하려면 먼저 도메인을 생성하고 시작해야 합니다. 그런 다음 자원 어댑터를 배치하고 구성할 수 있으며 설치 확인 테스트(IVT) 애플리케이션을 배치하고 실행할 수 있습니다.

시작하기 전에

- 이 지침은 GlassFish 서버 7.0 정식 버전을 위한 것입니다.
- 이 지침은 GlassFish 서버 7.0 웹 버전에서는 유효하지 않습니다.

이 태스크 정보

이 작업은 실행 중인 GlassFish 서버(7.0) 애플리케이션 서버가 있으며, 해당 서버의 표준 관리 작업에 익숙하다는 것을 전제로 합니다. 이 태스크는 또한 로컬 시스템에 IBM MQ가 설치되어 있으며 사용자가 표준 관리 태스크를 잘 알고 있다고 가정합니다.

참고: 다음 태스크 단계를 완료하려면 다음 오브젝트가 구성된 IBM MQ 설치가 작동 중이어야 합니다.

- 포트 1414에서 시작되는 QM이라는 큐 관리자이며, 채널 MQJMSRA.SVRCONN 을 사용하고, 클라이언트 전송을 통해 연결합니다.
- Q1이라는 큐

프로시저

1. GlassFish Server **asadmin** 셸 프로그램을 시작하십시오.

- a) 명령줄을 Windows 열고 *GlassFish/bin* 디렉터리로 이동하십시오. 여기서 *GlassFish* 는 GlassFish 서버 7.0 가 설치된 디렉터리입니다.
- b) 명령행에서 **asadmin** 명령을 입력하십시오.

asadmin 명령은 새 도메인을 작성할 수 있는 셸 프로그램을 명령행에서 엽니다.

GlassFish 서버 7.0 이(가) 시스템에서 시작되었습니다.

2. 도메인을 작성한 후에 이를 시작하십시오.

- a) 포트 및 도메인 이름을 지정하여 **create-domain** 명령을 사용하여 새 도메인을 작성하십시오.
명령행에서 다음 명령을 입력하십시오.

```
create-domain --adminport <port> <domain_name>
```

여기서 *port*는 포트 번호이고 *domain_name*은 도메인에서 사용할 이름입니다. 예를 들면, 다음과 같습니다.

```
create-domain -adminport 3232 mqjmsrdom
```

참고: **create-domain** 명령에는 이와 연관된 다수의 선택적 매개변수가 있습니다. 그러나 이 작업에는 **--adminport** 매개변수만 필요합니다. 자세한 내용은 GlassFish Server 7.0 정식 버전의 제품 설명서를 참조하십시오.

지정한 포트가 사용 중인 경우 다음 메시지가 표시됩니다.

< 도메인_이름 >의 < 포트 >가 사용 중입니다

지정한 도메인 이름이 사용 중이면 현재 사용 불가능한 모든 도메인 이름의 목록과 함께 지정된 이름이 이미 사용 중임을 알리는 메시지를 수신합니다.

- b) 사용자 이름 및 비밀번호를 입력하라는 프롬프트가 나타나면 웹 브라우저를 통해 애플리케이션 서버에 로그인하기 위해 사용되는 신임 정보를 입력하십시오.

명령이 성공적으로 완료되면 도메인 생성 요약 메시지가 명령줄에 표시되며, 다음 메시지를 포함합니다

명령어 **create-domain**이 성공적으로 실행되었습니다

도메인 작성이 완료되었습니다.

- c) 명령행에 다음 명령을 입력하여 도메인을 시작하십시오.

```
start-domain <domain_name>
```

여기서 *domain_name*은 사용자가 이전에 지정했던 도메인 이름입니다.

3. 웹 브라우저를 사용하여 GlassFish 애플리케이션 서버에 액세스하십시오.

- a) 웹 브라우저의 주소 표시줄에서 다음 명령을 입력하십시오.

```
localhost:<port>
```

여기서 *port*는 도메인을 작성할 때 이전에 지정한 포트입니다.

GlassFish 콘솔이 표시됩니다.

- b) GlassFish 콘솔이 로드되고 사용자 이름과 암호를 입력하라는 메시지가 표시되면, 단계에서 지정한 자격 증명을 465 페이지의 『2.b』 입력하십시오.

4. 리소스 어댑터를 GlassFish 서버(7.0)에 업로드하십시오.

- a) 공통 작업 도구 모음에서 **애플리케이션** 메뉴 항목을 선택하여 **애플리케이션** 페이지를 표시합니다.
- b) 배치 단추를 클릭하여 **애플리케이션 또는 모듈 배치** 페이지를 여십시오.
- c) **찾아보기** 버튼을 클릭한 후 파일의 **wmq.jakarta.jmsra.rar** 위치로 이동하십시오. 파일을 선택한 후 **확인**을 클릭하십시오.

5. 연결 풀을 작성하십시오.

- a) 도구 모음의 **자원** 아래에서 **커넥터** 메뉴 항목을 선택하십시오.
 - b) 그런 다음 **커넥터 연결 풀** 메뉴 항목을 선택하여 **커넥터 연결 풀** 페이지를 엽니다.
 - c) **새로 작성**을 클릭하여 **새 커넥터 연결 풀(1/2단계)** 페이지를 여십시오.
 - d) **새 커넥터 연결 풀(2단계 중 1단계)** 페이지에서 **풀 이름** 필드에 풀 `jms/ivt/IVTCF-Connection-Pool` 이름을 입력하십시오.
 - e) **리소스 어댑터** 필드에서 `를` 선택하십시오 `wmq.jakarta.jmsra`.
 - f) **연결 정의** 필드에 `를` 입력하십시오 `jakarta.jms.ConnectionFactory`.
 - g) 다음을 선택한 후에 **완료**를 선택하십시오.
6. 커넥터 자원을 작성하십시오.
- a) 도구 모음의 **커넥터** 메뉴에서 **커넥터 리소스** 옵션을 선택하여 **커넥터 리소스** 페이지를 엽니다.
 - b) **새로 만들기**를 선택하여 **새 커넥터 리소스** 페이지를 엽니다.
 - c) **JNDI 이름** 필드에서 `IVTCF`를 입력하십시오.
 - d) **풀 이름** 필드에서 `jms/ivt/IVTCF-Connection-Pool`을 입력하십시오.
 - e) 다른 모든 필드는 비워 두십시오.
 - f) 다음 속성/값 쌍 각각에 대해 **'속성 추가'**를 클릭하고 다음 예시와 같이 속성 이름과 값을 입력하십시오:

```
name: host; value: localhost
name: port; value: 1414
name: channel; value: MQJMSRA.SVRCONN
name: queueManager; value: QM
name: transportType; value: CLIENT
name: username; value: <user id for connecting to queue manager>
name: password; value: <password for connecting to queue manager>
```

참고: 자체 구성 설정에 대해 올바른 값을 사용 중인지 확인하십시오. 이는 이 예제에서 표시된 것과 다를 수 있습니다.

- g) 도구 모음의 **'커넥터'** 아래에서 **'관리자 객체 리소스'** 메뉴 항목을 선택하여 **관리자 객체 리소스** 페이지를 엽니다.
- h) **관리 오브젝트 자원** 페이지에서 **새로 작성**을 클릭하여 **새 관리 오브젝트 자원** 페이지를 여십시오.
- i) **JNDI 이름** 필드에서 `IVTQueue`를 입력하십시오.
- j) **리소스 어댑터** 필드에 `를` 입력하십시오 `wmq.jakarta.jmsra`.
- k) **리소스 유형** 필드에 `를` 입력하십시오 `jakarta.jms.Queue`.
- l) **클래스 이름** 필드는 그대로 두십시오.
- m) 다음의 특성/값 쌍 각각에 대해 **특성 추가**를 클릭하고 다음 예에 표시된 대로 특성 이름 및 값을 입력하십시오.

```
name: name; value: IVTQueue
name: baseQueueManagerName; value: QM
name: baseQueueName; value: Q1
```

참고: 자체 구성 설정에 대해 올바른 값을 사용 중인지 확인하십시오. 이는 이 예제에서 표시된 것과 다를 수 있습니다.

- n) **확인**을 클릭하십시오.
7. EAR 파일 `wmq.jakarta.jmsra.ivt.ear`(를) GlassFish 서버에 배치하십시오.
- a) 도구 모음에서 **애플리케이션** 옵션을 클릭하여 **애플리케이션** 페이지를 표시하십시오.
 - b) **배치**를 클릭하여 `IVT` 애플리케이션을 추가하십시오.
 - c) **위치** 필드에서 `wmq.jakarta.jmsra.ivt.ear`(를) 탐색하고 선택하십시오.
 - d) **가상 서버** 필드에서 **서버**를 선택한 후에 **확인**을 클릭하십시오.
8. `IVT` 애플리케이션을 실행하십시오.
- 자세한 정보는 [442 페이지의 『리소스 어댑터 IBM MQ 설치 확인』](#)의 내용을 참조하십시오.

WildFly 그리고 IBM MQ 리소스 어댑터

리소스 IBM MQ 어댑터는 Wildfly에 설치할 수 있으며, 설치 검증 테스트(IVT) 애플리케이션을 설치하고 실행하여 설치를 테스트할 수 있습니다.

JMS 2.0 WildFly 에서 리소스 IBM MQ 어댑터 설치 및 테스트

IBM MQ 자원 어댑터를 Wildfly V10에 설치하는 경우 먼저 IBM MQ 자원 어댑터에 대한 서브시스템 정의를 추가하도록 구성 파일을 변경해야 합니다. 그런 다음 자원 어댑터를 배치하고 설치 확인 테스트(IVT) 애플리케이션을 설치하고 실행하여 자원 어댑터를 테스트할 수 있습니다.

시작하기 전에

- 이 지시사항은 Wildfly V10에 사용됩니다.
- 이 버전의 WildFly 은, Jakarta EE을 지원하지 않습니다.

이 태스크 정보

이 태스크는 WildFly 애플리케이션 서버가 실행 중이며 사용자가 이에 대한 표준 관리 태스크를 잘 알고 있다고 가정합니다. 이 태스크는 또한 IBM MQ가 설치되어 있으며 사용자가 표준 관리 태스크를 잘 알고 있다고 가정합니다.

프로시저

1. ExampleQM이라는 IBM MQ 큐 관리자를 작성하고 979 페이지의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』에서 설명하는 대로 이를 설정하십시오.
큐 관리자를 설정할 때 다음과 같은 점에 유의하십시오.
 - 리스너는 포트 1414에서 시작되어야 합니다.
 - 사용할 채널을 SYSTEM.DEF.SVRCONN이라고 합니다.
 - IVT 애플리케이션에서 사용하는 큐는 TEST.QUEUE로 이름 지정됩니다.또한 이 애플리케이션이 임시 응답 큐를 작성할 수 있도록 모델 큐 SYSTEM.DEFAULT.MODEL.QUEUE에 DSP 및 PUT 권한을 부여해야 합니다.
2. 구성 파일 *WildFly_Home/standalone/configuration/standalone-full.xml*을(를) 편집하고 다음 서브시스템을 추가하십시오.

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:4.0">
  <resource-adapters>
    <resource-adapter id="wmq.jmsra">
      <archive>
        wmq.jmsra.rar
      </archive>
      <transaction-support>NoTransaction</transaction-support>
      <connection-definitions>
        <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
jndi-name="java:jboss/jms/ivt/IVTCF" enabled="true"
use-java-context="true"
pool-name="IVTCF">
          <config-property name="channel">SYSTEM.DEF.SVRCONN
        </config-property>
          <config-property
name="hostName">localhost
        </config-property>
          <config-property name="transportType">
CLIENT
        </config-property>
          <config-property name="queueManager">
ExampleQM
        </config-property>
          <config-property name="port">
1414
        </config-property>
        </connection-definition>
        <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl">
```

```

jndi-name="java:jboss/jms/ivt/JMS2CF" enabled="true"
use-java-context="true"
    pool-name="JMS2CF">
    <config-property name="channel">
        SYSTEM.DEF.SVRCONN
    </config-property>
    <config-property name="hostName">
        localhost
    </config-property>
    <config-property name="transportType">
        CLIENT
    </config-property>
    <config-property name="queueManager">
        ExampleQM
    </config-property>
    <config-property name="port">
        1414
    </config-property>
    </connection-definition>
</connection-definitions>
<admin-objects>
    <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
        jndi-name="java:jboss/jms/ivt/IVTQueue" pool-name="IVTQueue">
        <config-property name="baseQueueName">
            TEST.QUEUE
        </config-property>
    </admin-object>
</admin-objects>
</resource-adapter>
</resource-adapters>
</subsystem>

```

3. `wmq.jmsra.rar` 파일을 `WildFly_Home/standalone/deployments` 디렉토리에 복사하여 서버에 자원 어댑터를 배치하십시오.
4. `wmq.jmsra.ivt.ear` 파일을 `WildFly_Home/standalone/deployments` 디렉토리로 복사해서 IVT 애플리케이션을 배치하십시오.
5. 명령 프롬프트를 표시하고 `WildFly_Home/bin` 디렉토리로 이동한 후 다음 명령을 실행하여 애플리케이션 서버를 시작하십시오.

```
standalone.bat -c standalone-full.xml
```

6. IVT 애플리케이션을 실행하십시오.

자세한 정보는 442 페이지의 『리소스 어댑터 IBM MQ 설치 확인』의 내용을 참조하십시오. Wildfly의 경우 기본 URL은 `http://localhost:8080/WMQ_IVT/`입니다.

WildFly에서 Jakarta Messaging 리소스 IBM MQ 어댑터 설치 및 테스트

WildFly28.0.0 또는 이후 버전에 Jakarta Messaging 리소스 IBM MQ 어댑터를 설치하는 경우, 먼저 구성 파일을 수정하여 리소스 어댑터에 대한 하위 시스템 정의를 추가해야 합니다. 그런 다음 자원 어댑터를 배치하고 설치 확인 테스트(IVT) 애플리케이션을 설치하고 실행하여 자원 어댑터를 테스트할 수 있습니다.

이 태스크 정보

중요사항: 이 지침은 WildFly28.0.0에 대한 것입니다.

이 태스크는 WildFly 애플리케이션 서버가 실행 중이며 사용자가 이에 대한 표준 관리 태스크를 잘 알고 있다고 가정합니다. 이 태스크는 또한 IBM MQ가 설치되어 있으며 사용자가 표준 관리 태스크를 잘 알고 있다고 가정합니다.

프로시저

1. 라는 이름의 큐 IBM MQExampleQM 관리자를 생성하고, [다중 플랫폼에서 클라이언트 연결을 수락하도록 큐 관리자](#) 구성에 설명된 대로 설정하십시오.
큐 관리자를 설정할 때 다음과 같은 점에 유의하십시오.
 - 리스너는 포트 1414에서 시작되어야 합니다.
 - 사용할 채널을 SYSTEM.DEF.SVRCONN이라고 합니다.

- IVT 애플리케이션에서 사용하는 큐는 TEST.QUEUE로 이름 지정됩니다.

또한 이 애플리케이션이 임시 응답 큐를 작성할 수 있도록 모델 큐 SYSTEM.DEFAULT.MODEL.QUEUE에 DSP 및 PUT 권한을 부여해야 합니다.

2. 구성 파일 *WildFly_Home/standalone/configuration/standalone-full.xml*을(를) 편집하고 다음 서브시스템을 추가하십시오.

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:6.1">
  <resource-adapters>
    <resource-adapter id="wmq.jakarta.jmsra.rar">
      <archive>
        wmq.jakarta.jmsra.rar
      </archive>
      <transaction-support>NoTransaction</transaction-support>
      <connection-definitions>
        <connection-definition class-
name="com.ibm.mq.jakarta.connector.outbound.ManagedConnectionFactoryImpl"
jndi-name="java:jboss/jms/ivt/IVTCF" enabled="true"
use-java-context="true"
pool-name="IVTCF">
          <config-property name="channel">
            SYSTEM.DEF.SVRCONN
          </config-property>
          <config-property name="hostName">
            localhost
          </config-property>
          <config-property name="transportType">
            CLIENT
          </config-property>
          <config-property name="queueManager">
            ExampleQM
          </config-property>
          <config-property name="port">
            1414
          </config-property>
        </connection-definition>
        <connection-definition class-
name="com.ibm.mq.jakarta.connector.outbound.ManagedConnectionFactoryImpl"
jndi-name="java:jboss/jms/ivt/JMS2CF" enabled="true"
use-java-context="true"
pool-name="JMS2CF">
          <config-property name="channel">
            SYSTEM.DEF.SVRCONN
          </config-property>
          <config-property name="hostName">
            localhost
          </config-property>
          <config-property name="transportType">
            CLIENT
          </config-property>
          <config-property name="queueManager">
            ExampleQM
          </config-property>
          <config-property name="port">
            1414
          </config-property>
        </connection-definition>
      </connection-definitions>
      <admin-objects>
        <admin-object class-
name="com.ibm.mq.jakarta.connector.outbound.MQQueueProxy"
jndi-name="java:jboss/jms/ivt/IVTQueue" pool-name="IVTQueue">
          <config-property name="baseQueueName">
            TEST.QUEUE
          </config-property>
        </admin-object>
      </admin-objects>
    </resource-adapter>
  </resource-adapters>
</subsystem>
```

3. wmq.jakarta.jmsra.rar 파일을 *WildFly_Home/standalone/deployments* 디렉토리에 복사하여 서버에 자원 어댑터를 배치하십시오.
4. IVT 애플리케이션을 배포하려면 파일을 wmq.jakarta.jmsra.ivt.ear 디렉터리로 *WildFly_Home/standalone/deployments* 복사하십시오.

5. 명령 프롬프트를 표시하고 `WildFly_Home/bin` 디렉토리로 이동한 후 다음 명령을 실행하여 애플리케이션 서버를 시작하십시오.

```
standalone.bat -c standalone-full.xml
```

6. IVT 애플리케이션을 실행하십시오.

자세한 정보는 442 페이지의 『리소스 어댑터 IBM MQ 설치 확인』의 내용을 참조하십시오. WildFly, 의 경우 기본 URL 은 http://localhost:8080/WMQ_JAKARTA_IVT/

IBM MQ 및 WebSphere Application Server 함께 사용

메시징 IBM MQ 제공자를 통해, Java Message Service WebSphere Application Server (JMS) 메시징 애플리케이션은 귀하의 IBM MQ 시스템을 메시징 JMS 리소스의 외부 제공자로 활용할 수 있습니다.

이 태스크 정보

에서 작성되고 Java 에서 실행되는 애플리케이션은 메시징을 WebSphere Application Server 수행하기 위해 (JMS) Java Message Service 사양을 사용할 수 있습니다. 이 환경에서 메시징은 IBM MQ 큐 관리자에서 제공할 수 있습니다.

IBM MQ 큐 관리자를 사용할 때 얻게 되는 혜택은 연결하는 JMS 애플리케이션이 IBM MQ 네트워크의 기능에 전체 참여할 수 있으며, 이를 통해 애플리케이션이 여러 플랫폼에서 실행 중인 큐 관리자와 메시지를 교환할 수 있습니다.

애플리케이션은 큐 연결 팩토리 오브젝트에 대해 클라이언트 전송 또는 바인딩 전송을 사용할 수 있습니다. 바인딩 전송의 경우, 연결이 필요한 애플리케이션에 대해 큐 관리자가 로컬에 존재해야 합니다.

기본적으로 큐에 IBM MQ 보관된 메시지는 일부 JMS 메시지 헤더 정보를 보관하기 위해 `MQRFH2JMS` 헤더를 사용합니다. 많은 레거시 IBM MQ 애플리케이션은 이러한 헤더를 포함하는 메시지를 처리할 수 없으며 고유한 특성 헤더(예: CICS 브릿지의 경우 MQCIH, IBM MQ 워크플로우 애플리케이션의 경우 MQWIH)가 필요합니다. 이러한 특수 고려사항에 대한 자세한 정보는 [JMS 메시지를 IBM MQ 메시지로 맵핑](#)을 참조하십시오.

관련 태스크

[리소스 JMS 구성 WebSphere Application Server](#)

[최신 자원 어댑터 유지보수 레벨을 사용하도록 애플리케이션 서버 구성](#)

IBM MQ과(와) 함께 WebSphere Application Server 사용

IBM MQ 및 IBM MQ for z/OS는 WebSphere Application Server에 포함된 기본 메시징 제공자와 함께 또는 이에 대한 대체 방법으로 사용될 수 있습니다.

IBM MQ 메시징 제공자는 WebSphere Application Server의 일부로 설치됩니다. 이는 IBM MQ 자원 어댑터의 한 버전 및 IBM MQ 확장 트랜잭션 클라이언트 기능을 포함하며 이를 통해 애플리케이션 서버에서 관리하는 XA 트랜잭션에 큐 관리자가 참여할 수 있게 합니다. 자원 어댑터를 사용하여 MDB는 활성화 스펙 또는 리스너 포트 중 하나를 사용하도록 구성될 수 있습니다.

애플리케이션 서버를 지원하려면, IBM MQ 자원 어댑터 설치 검증 테스트 프로그램이 애플리케이션 서버에 배치되고 실행해야 합니다. IBM MQ 자원 어댑터 설치 검증 테스트 프로그램이 실행된 후 IBM MQ 자원 어댑터는 지원되는 IBM MQ 큐 관리자에도 연결할 수 있습니다.

JMS 연결점: 에서 WebSphere Application Server 까지 IBM MQ

애플리케이션 서버 내에서 실행되는 애플리케이션이 큐 IBM MQ 매니저에 연결하는 방식을 JMS이해하는 것이 Java Message Service 중요합니다. 이를 통해 WebSphere Application Server와 함께 사용할 IBM MQ 수 있는의 수준을 고려하기 전에

JMS 큐 IBM MQ 관리자의 리소스에 접근해야 하는 애플리케이션은 다음 전송 유형 중 하나를 사용하여 이를 수행할 수 있습니다:

BINDINGS

이 전송은 애플리케이션 서버와 큐 관리자가 동일한 시스템 및 운영 체제 이미지에 설치된 경우에 사용됩니다. BINDINGS 모드를 사용할 때 두 제품 사이의 통신은 프로세스 간 통신(IPC)을 사용하여 수행됩니다.

IBM MQ 메시징 제공자는 BINDINGS 모드에서 IBM MQ 큐 관리자에 연결하는 데 필요한 고유 라이브러리를 포함하지 않습니다. BINDINGS 모드 연결을 사용하기 위해서는 IBM MQ를 애플리케이션 서버와 동일한 시스템에 설치해야 하며 자원 어댑터의 기본 라이브러리 경로가 이러한 라이브러리가 위치한 IBM MQ 디렉토리를 가리키도록 구성되어야 합니다. 자세한 정보는 WebSphere Application Server 제품 문서를 참조하십시오.

- WebSphere Application Server의 경우, 고유 라이브러리로 IBM MQ 메시징 제공자 구성의 내용을 참조하십시오.
- WebSphere Liberty에 대해서는 IBM MQ 메시징 제공자를 사용하기 위해 JMS 애플리케이션을 Liberty에 배치를 참조하십시오.

z/OS z/OS에서, WebSphere Application Server 연결 팩토리를 바인딩 모드의 IBM MQ 큐 관리자에 연결하려면 올바른 IBM MQ 라이브러리를 WebSphere Application Server STEPLIB 연결로 지정해야 합니다. 자세한 내용은 제품 WebSphere Application Server 설명서의 라이브러리 및 IBM MQz/OS STEPLIB에 관한 WebSphere Application Server 부분을 참조하십시오.

CLIENT

클라이언트 전송은 WebSphere Application Server와 IBM MQ 간에 통신하기 위해 TCP/IP를 사용합니다. CLIENT 모드는 애플리케이션 서버와 큐 관리자가 서로 다른 시스템에 위치한 경우에 사용될 뿐 아니라 두 개의 제품이 동일한 시스템 및 운영 체제 이미지에 설치된 경우에도 사용될 수 있습니다.

JMS 애플리케이션은 또한 BINDINGS_THEN_CLIENT의 전송 유형을 지정할 수 있습니다. 이 전송 유형이 사용될 때 애플리케이션은 처음에 BINDINGS 모드를 사용하여 큐 관리자에 연결을 시도합니다. 그렇게 할 수 없을 경우에는 CLIENT 전송을 시도합니다. BINDINGS_THEN_CLIENT 전송 유형을 지정할 경우 ccdtURL 속성은 지원되지 않습니다.

내부에 설치된 리소스 IBM MQ 어댑터 버전을 확인하는 방법 WebSphere Application Server

설치된 리소스 IBM MQ 어댑터 버전에 대한 WebSphere Application Server 정보는 기술 노트 "WebSphere Application Server에 포함된 WebSphere MQ 리소스 어댑터(RA) 버전은 무엇인가요?"를 참조하십시오.

다음 Jython 및 JACL 명령을 사용하여 WebSphere Application Server에서 현재 사용 중인 자원 어댑터의 레벨을 판별할 수 있습니다.

Jython

```
wmqInfoBeansUnsplit = AdminControl.queryNames("WebSphere:type=WMQInfo,*")
wmqInfoBeansSplit = AdminUtilities.convertToList(wmqInfoBeansUnsplit)
for wmqInfoMBean in wmqInfoBeansSplit: print wmqInfoMBean; print
AdminControl.invoke(wmqInfoMBean, 'getInfo', '')
```

참고: 이 명령을 입력한 후 실행하기 위해서는 리턴을 두 번 클릭해야 합니다.

JACL

```
set wmqInfoBeans [$AdminControl queryNames WebSphere:type=WMQInfo,*]
foreach wmqInfoMBean $wmqInfoBeans {
  puts $wmqInfoMBean;
  puts [$AdminControl invoke $wmqInfoMBean getInfo [] []]
}
```

자원 어댑터 업데이트

애플리케이션 서버와 함께 설치된 IBM MQ 자원 어댑터에 대한 업데이트는 WebSphere Application Server 수정팩에 포함됩니다. 리소스 어댑터 **업데이트...**를 IBM MQ 사용하여 리소스 어댑터 업데이트 관리 WebSphere Application Server 콘솔에서 해당 기능을 비활성화하는 것은 권장되지 않습니다. 이렇게 하면 수정 WebSphere Application Server 팩에서 제공되는 업데이트가 적용되지 않기 때문입니다.

MQ_INSTALL_ROOT 변수

WebSphere Application Server 7.0부터, MQ_INSTALL_ROOT는 고유 라이브러리를 찾는 데에만 사용할 수 있으며 자원 어댑터에서 구성된 고유 라이브러리 경로로 대체됩니다.

WebSphere Application Server에서 IBM MQ에 연결



주의:

- 지원되는 WebSphere Application Server 버전은 함께 번들된 IBM MQ 자원 어댑터를 사용하여 지원되는 IBM MQ 버전에 연결할 수 있습니다.
- 바인딩 모드가 사용되는 경우에는 WebSphere Application Server의 특정 라이브러리가 연결하는 큐 관리자의 버전과 일치해야 합니다.
 - WebSphere Application Server는 IBM MQ 10.0에서 제공되는 고유 라이브러리를 로드하도록 구성되어야 합니다. 자세한 정보는 96 페이지의 『JNI(Java Native Interface) 라이브러리 구성』의 내용을 참조하십시오.
 - z/OS** z/OS에서는 올바른 IBM MQ 라이브러리를 WebSphere Application Server STEPLIB 연결로 지정해야 합니다.
필요한 IBM MQ 라이브러리의 세부사항은 IBM MQ 라이브러리 및 z/OS용 WebSphere Application Server STEPLIB를 참조하십시오.
LINKLIST(LINKLST)에 한 IBM MQ 버전을 위한 라이브러리가 있는 경우에는 STEPLIB으로 라이브러리를 대체하여 다른 IBM MQ 버전에 연결할 수 있습니다.
- IBM MQ 자원 어댑터 버전은 큐 관리자 설치에 의해 제공된 고유(공유) 라이브러리 버전과 관련이 없습니다.
예를 들어 WebSphere Application Server 8.5, 리소스 IBM WebSphere MQ 7.1 어댑터는 네이티브 IBM MQ 9.0 라이브러리를 사용하여 큐 IBM MQ 9.0 매니저에 대한 바인딩 연결을 여전히 관리할 수 있습니다.

자세한 정보는 388 페이지의 『지원의 IBM MQ 자원 어댑터 명령문』의 내용을 참조하십시오.

BINDINGS 및 CLIENT 전송 유형은 모든 버전의 에 연결하는 데 IBM MQ WebSphere Application Server 사용할 수 있습니다. BINDINGS 전송 유형의 경우 다음 제한 사항이 적용됩니다:

- IBM MQ은(는) 애플리케이션 서버와 동일한 시스템에 설치해야 합니다.
- WebSphere Application Server는 IBM MQ에서 제공되는 고유 라이브러리를 로드하도록 구성되어야 합니다.
- z/OS** z/OS에서, WebSphere Application Server 연결 팩토리를 BINDINGS 모드의 IBM MQ 큐 관리자에 연결하려면 올바른 IBM MQ 라이브러리를 WebSphere Application Server STEPLIB 연결로 지정해야 합니다.

다음 표는 각 버전의 리소스 IBM MQ 어댑터가 WebSphere Application Server 실행을 지원하는 버전을 보여줍니다.

표 71. 버전을 리소스 WebSphere Application Server IBM MQ 어댑터 버전에 매핑하기.	
IBM MQ 자원 어댑터의 버전	이 자원 어댑터 버전이 실행될 수 있는 WebSphere Application Server 버전
IBM MQ 9.0 이상	자원 어댑터는 다음 버전에서 실행될 수 있습니다. - 모든 Java EE 7 호환 버전의 WebSphere Liberty. - WebSphere Application Server 9.0
IBM MQ 8.0	이 자원 어댑터는 모든 Java EE 7 준수 WebSphere Liberty 버전에서 실행될 수 있습니다. IBM MQ 8.0 자원 어댑터는 WebSphere Application Server에서 실행할 수 없습니다. 이미 WebSphere Application Server에 설치된 자원 어댑터는 IBM MQ 8.0 큐 관리자에 연결하는 데 사용되어야 합니다.

관련 개념

388 페이지의 『지원의 IBM MQ 자원 어댑터 명령문』

응용 프로그램과 큐 관리자 간 통신에 사용해야 하는 리소스 IBM MQ 어댑터는 API를 Jakarta Messaging 3.0 사용하는지 아니면 API를 JMS 2.0 사용하는지에 따라 달라집니다.

관련 정보

[IBM MQ 시스템 요구사항](#)

WebSphere Application Server에서 IBM MQ에 대해 작성된 TCP/IP 연결 수 판별

공유 대화 기능을 사용하여 다중 대화에서 MQI 채널 인스턴스(TCI/IP 연결이라고도 함)를 공유할 수 있습니다.

이 태스크 정보

내부에서 실행되는 애플리케이션과 메시징 WebSphere Application Server 7 공급자의 일반 WebSphere Application Server 8 IBM MQ 모드를 사용하는 애플리케이션은 자동으로 이 기능을 사용합니다. 이것은 동일한 IBM MQ 큐 관리자에 연결하는 동일한 애플리케이션 서버 인스턴스 내에서 실행 중인 다중 애플리케이션이 동일한 채널 인스턴스를 공유할 수 있음을 의미합니다.

단일 채널 인스턴스에서 공유될 수 있는 대화 수는 IBM MQ 채널 특성 **SHARECNV**로 결정됩니다. 서버 연결 채널에 대한 이 특성의 기본값은 10입니다.

WebSphere Application Server 7 및 WebSphere Application Server 8에서 작성한 대화 수를 확인하여 작성된 채널 인스턴스 수를 판별할 수 있습니다.

IBM MQ 메시징 제공자 모드에 대한 자세한 정보는 [PROVIDERVERSION 정상 모드](#)를 참조하십시오.

관련 개념

[공유 대화 사용](#)

공유 대화가 허용되는 환경에서 대화는 MQI 채널 인스턴스를 공유할 수 있습니다.

291 페이지의 『IBM MQ classes for JMS에서 TCP/IP 연결 공유』

단일 TCP/IP 연결을 공유하도록 MQI 채널의 다중 인스턴스를 작성할 수 있습니다.

JMS 연결 팩토리

'IBM MQ 메시징 공급자 연결 팩토리를 사용하여 연결 및 세션을 만드는 'WebSphere Application Server' 내부에서 실행되는 애플리케이션은 연결 팩토리에서 생성된 모든 'JMS 연결과 'JMS' 연결에서 생성된 모든 'JMS' 세션에 대해 활성 대화가 있습니다.

연결 팩토리에서 작성된 모든 JMS 연결에 대해 하나의 대화

각 JMS 연결 팩토리에는 여유 풀과 활성 풀이라는 두 개의 섹션으로 나뉘어 지는 연관된 연결 풀이 있습니다. 두 개의 풀은 모두 처음에 비어 있습니다.

애플리케이션이 연결 팩토리에서 JMS 연결을 작성하면 WebSphere Application Server가 여유 풀에 JMS 연결이 있는지를 확인하는 검사를 수행합니다. 연결이 있으면 활성 풀로 이동되고 애플리케이션에 제공됩니다. 연결이 없으면 새 JMS 연결이 작성되고 활성 풀에 놓여지며 애플리케이션으로 돌아갑니다. 연결 팩토리에서 만들 수 있는 최대 연결 수는 연결 팩토리 연결 풀 속성 '**Maximum connections**'에 의해 지정됩니다. 이 특성의 기본값은 10입니다.

애플리케이션이 JMS 연결을 종료하고 닫히면 연결이 활성 풀에서 여유 풀로 이동하고 다시 사용할 수 있게 됩니다. 연결 풀 특성 **Unused timeout**은 연결이 끊어지기 전에 JMS 연결이 여유 풀에 보존될 수 있는 시간을 정의합니다. 이 특성의 기본값은 1800초(30분)입니다.

JMS 연결을 처음으로 작성하면 WebSphere Application Server 및 IBM MQ 사이의 대화가 시작됩니다. 무료 풀의 '**Unused timeout**' 속성 값이 초과되면 연결이 종료될 때까지 대화가 활성 상태로 유지됩니다.

JMS 연결에서 작성된 모든 JMS에 대해 하나의 대화

'IBM MQ 메시징 공급자 연결 팩토리에서 생성되는 모든 'JMS' 연결에는 연결된 'JMS' 세션 풀이 있습니다. 이러한 세션 풀은 연결 풀과 동일한 방식으로 작동합니다. 단일 'JMS' 연결에서 생성할 수 있는 최대 'JMS' 세션 수는 연결 팩토리 세션 풀 속성 '**Maximum connections**'에 의해 결정됩니다. 이 특성의 기본값은 10입니다.

대화는 'JMS 세션이 처음 생성될 때 시작되며, 'JMS' 세션이 세션 풀의 '**Unused timeout**' 속성 값보다 더 오랫동안 여유 풀에 남아 있기 때문에 대화가 종료될 때까지 활성 상태로 유지됩니다.

SHARECNV 특성 값 계산

다음 공식을 사용하여 단일 연결 팩토리에서 IBM MQ까지 최대 대화 수를 계산할 수 있습니다.

```
Maximum number of conversations =  
connection Pool Maximum Connections +  
(connection Pool Maximum Connections * Session Pool Maximum Connections)
```

이러한 수의 대화가 발생하는 것을 허용하도록 작성될 채널 인스턴스의 수는 다음 계산을 사용하여 알 수 있습니다.

```
Maximum number of channel instances =  
Maximum number of conversations / SHARECNV for the channel being used
```

이 계산식에서 나머지는 반올림할 수 있습니다.

연결 풀 '**Maximum connections**' 및 세션 풀 '**Maximum connections**' 속성의 기본값을 사용하는 단순 연결 팩토리의 경우 이 연결 팩토리에 대해 'WebSphere Application Server'와 'IBM MQ' 사이에 존재할 수 있는 최대 대화 수는 다음과 같습니다:

```
Maximum number of conversations =  
connection Pool Maximum Connections +  
(connection Pool Maximum Connections * Session Pool Maximum Connections)
```

예를 들면, 다음과 같습니다.

```
= 10 + (10 * 10)  
= 10 + 100  
= 110
```

이 연결 팩토리가 '**SHARECNV**' 속성이 10으로 설정된 채널을 사용하여 'IBM MQ'에 연결하는 경우 이 연결 팩토리에 대해 생성되는 채널 인스턴스의 최대 개수는 다음과 같습니다:

```
Maximum number of channel instances = Maximum number of conversations / SHARECNV for the  
channel being used
```

예를 들면, 다음과 같습니다.

```
= 110 / 10  
= 11 (rounded up to nearest connection)
```

활성화 스펙

활성화 스펙을 사용하도록 구성된 MDB(Message-driven bean) 애플리케이션은 메시지를 처리하기 위해 MDB 인스턴스를 실행하는 데 사용되는 모든 서버 세션에 대해 그리고 JMS 목적지를 모니터링하기 위한 활성화 스펙에 대해 활성화된 대화를 갖습니다.

다음 대화는 활성화 스펙을 사용하도록 구성된 MDB(message-driven bean) 애플리케이션에 대해 활성화됩니다.

- 적절한 메시지에 대한 JMS 목적지를 모니터링하기 위해 활성화 스펙에 대한 하나의 대화. 이 대화는 활성화 스펙이 시작되지 마자 시작되고 활성화 스펙이 중지될 때까지 활성으로 남아 있습니다.
- 메시지를 처리하기 위해 MDB 인스턴스를 실행하는 데 사용되는 모든 서버 세션에 대해 하나의 대화.

활성화 사양 고급 속성은 주어진 활성화 사양에 대해 동시에 **Maximum server sessions** 활성화될 수 있는 서버 세션의 최대 수를 지정합니다. 이 특성의 기본값은 10입니다. 서버 세션은 필요에 따라 작성되고 활성화 스펙 고급 특성 **Server session pool timeout**에서 지정한 시간 동안 유휴 상태인 경우 종료됩니다. 이 특성의 기본값은 300000 밀리초(5분)입니다.

서버 세션이 작성될 때 대화가 시작되고 활성화 스펙이 중지되거나 서버 세션이 제한시간 초과될 때 중지됩니다.

이는 단일 활성화 스펙에서 IBM MQ까지의 최대 대화 수가 다음 공식으로 계산될 수 있음을 의미합니다.

$$\text{Maximum number of conversations} = \text{Maximum server sessions} + 1$$

이러한 수의 대화가 발생하는 것을 허용하도록 작성되는 채널 인스턴스의 수는 다음 계산을 사용하여 알 수 있습니다.

$$\text{Maximum number of channel instances} = \frac{\text{Maximum number of conversations}}{\text{SHARECNV for the channel being used}}$$

이 계산식에서 나머지는 반올림할 수 있습니다.

기본 **Maximum server sessions** 속성값을 사용하는 단순한 활성화 사양의 경우, 이 활성화 사양에 대해 IBM MQ 와 WebSphere Application Server 사이에 존재할 수 있는 최대 대화 수는 다음과 같이 계산됩니다:

$$\text{Maximum number of conversations} = \text{Maximum server sessions} + 1$$

예를 들면, 다음과 같습니다.

$$\begin{aligned} &= 10 + 1 \\ &= 11 \end{aligned}$$

이 활성화 사양이 속성이 **SHARECNV** 10으로 설정된 채널을 사용하여 IBM MQ 연결하는 경우 생성되는 채널 인스턴스 수는 다음과 같이 계산됩니다:

$$\text{Maximum number of channel instances} = \frac{\text{Maximum number of conversations}}{\text{SHARECNV for the channel being used}}$$

예를 들면, 다음과 같습니다.

$$\begin{aligned} &= 11 / 10 \\ &= 2 \text{ (rounded up to nearest connection)} \end{aligned}$$

Application Server Facilities(ASF) 모드에서 실행 중인 리스너 포트

ASF 모드에서 실행 중인 MDB(message-driven bean) 애플리케이션에서 사용하는 리스너 포트가 각 서버 세션에 대해 대화를 작성합니다. 하나는 적절한 메시지에 대한 목적지를 모니터하고 다른 하나는 메시지를 처리하기 위해 MDB 인스턴스를 실행합니다. 각 리스너 포트에 대한 대화 수는 최대 세션 수에서 계산될 수 있습니다.

기본적으로 리스너 포트는 애플리케이션 서버가 메시지를 감지하고 처리를 위해 MDB에 전달하기 위해 사용해야 하는 메커니즘을 정의하는 1.1. 스펙의 일부로 ASF 모드에서 실행됩니다. 이 조작의 기본 모드에서 리스너 포트를 사용하기 위해 설정된 MDB 애플리케이션은 대화를 작성합니다.

적절한 메시지에 대한 목적지를 모니터하기 위해 리스너 포트에 대한 하나의 대화.

리스너 포트는 JMS 연결 팩토리를 사용하도록 구성됩니다. 리스너 포트가 시작되면 연결 팩토리 여유 풀에서 JMS 연결에 대한 요청이 작성됩니다. 리스너 포트가 중지되면 연결이 여유 풀로 돌아갑니다. 연결 풀이 사용되는 방법 및 이러한 방법이 IBM MQ에 대한 대화 수에 영향을 미치는 방법은 [473 페이지의 『JMS 연결 팩토리』](#)의 내용을 참조하십시오.

메시지를 처리하기 위해 MDB 인스턴스를 실행하는 데 사용되는 모든 서버 세션에 대해 하나의 대화.

리스너 포트 특성 **Maximum sessions**는 주어진 리스너 포트에 대해 한 번에 활성화할 수 있는 최대 서버 세션 수를 지정합니다. 이 특성의 기본값은 10입니다. 서버 세션은 필요에 따라 작성되며 리스너 포트가 사용 중인 JMS 연결과 연관된 JMS 세션을 사용합니다.

서버 세션이 메시지 리스너 서비스 사용자 정의 특성 **SERVER.SESSION.POOL.UNUSED.TIMEOUT**에서 지정한 기간 동안 유휴 상태인 경우 세션이 닫히고, 사용된 JMS 세션이 세션 풀 여유 풀로 리턴됩니다. JMS 세션은 필요할 때까지 세션 풀 여유 풀에 남아 있거나 **Unused timeout** 특성 값보다 긴 시간 동안 여유 풀에 유휴 상태로 있는 경우에는 닫힙니다.

세션 풀이 사용되는 방법 및 WebSphere Application Server와 IBM MQ 사이의 대화가 관리되는 방법에 대한 자세한 정보는 [473 페이지의 『JMS 연결 팩토리』](#)의 내용을 참조하십시오.

메시지 리스너 서비스 사용자 정의 특성 **SERVER.SESSION.POOL.UNUSED.TIMEOUT**, WebSphere Application Server 제품 문서의 [리스너 포트에 대한 서버 세션 풀 모니터링](#) 을 참조하십시오.

단일 리스너 포트에서 IBM MQ까지 최대 대화 수 계산

다음 공식을 사용하여 단일 리스너 포트에서 IBM MQ까지 최대 대화 수를 계산할 수 있습니다.

$$\text{Maximum number of conversations} = \text{Maximum sessions} + 1$$

이러한 수의 대화가 발생하는 것을 허용하도록 작성될 채널 인스턴스의 수는 다음 계산을 사용하여 알 수 있습니다.

$$\begin{aligned} \text{Maximum number of channel instances} = \\ \text{Maximum number of conversations} / \text{SHARECNV for the channel being used} \end{aligned}$$

이 계산식에서 나머지는 반올림할 수 있습니다.

Maximum sessions 특성의 기본값을 사용하는 단순 리스너 포트의 경우, 이 리스너 포트에 대해 WebSphere Application Server 와 IBM MQ 사이에 존재할 수 있는 최대 대화 수는 다음과 같이 계산됩니다.

$$\text{Maximum number of conversations} = \text{Maximum sessions} + 1$$

예를 들면, 다음과 같습니다.

$$\begin{aligned} &= 10 + 1 \\ &= 11 \end{aligned}$$

이 리스너 포트가 **SHARECNV** 특성이 10으로 설정된 채널을 사용하여 IBM MQ 에 연결 중인 경우, 작성될 채널 인스턴스의 수는 다음과 같이 계산됩니다.

$$\begin{aligned} \text{Maximum number of channel instances} = \\ \text{Maximum number of conversations} / \text{SHARECNV for the channel being used} \end{aligned}$$

예를 들면, 다음과 같습니다.

$$\begin{aligned} &= 11 / 10 \\ &= 2 \text{ (rounded up to nearest connection)} \end{aligned}$$

비Application Server Facilities(비ASF) 모드에서 실행 중인 리스너 포트

Non-ASF 모드에서 실행 중인 리스너 포트는 서버 세션을 사용하여 토픽 목적지 및 큐 목적지를 모니터링하도록 구성될 수 있습니다. 서버 세션은 복수의 대화를 가질 수 있으며 각 사례별로 최대 대화 수를 계산할 수 있습니다.

리스너 포트는 리스너 포트가 JMS 목적지를 모니터링하는 방식을 변경하는 Non-ASF 모드에서 실행하도록 구성될 수 있습니다. Non-ASF 모드의 조작에서 리스너 포트를 사용하는 MDB(Message-driven bean) 애플리케이션은 메시지를 처리하기 위해 MDB 인스턴스를 실행하는 데 사용하는 모든 서버 세션에 대한 대화를 작성합니다. 리스너 포트 특성 **Maximum sessions**는 주어진 리스너 포트에 대해 한 번에 활성화할 수 있는 최대 서버 세션 수를 지정합니다. 이 특성의 기본값은 10입니다.

Non-ASF 모드에서 실행할 때 큐 목적지를 모니터링하는 리스너 포트는 리스너 포트 특성 **최대 세션 수**에서 지정한 서버 세션 수를 자동으로 작성합니다. 이러한 모든 서버 세션은 리스너 포트가 사용 중인 JMS 연결과 연관된 세션 풀에서 가져온 JMS 세션을 이용하며 계속해서 적절한 메시지에 대해 JMS 목적지를 모니터링합니다.

리스너 포트가 토픽 목적지를 모니터링하도록 구성된 경우 **최대 세션 수**가 무시되고 단일 서버 세션이 사용됩니다.

Non-ASF 모드로 실행 중인 리스너 포트에서 사용하는 서버 세션은 사용된 JMS 세션이 리스너 포트가 사용 중이었던 JMS 연결에 대한 세션 풀인 여유 풀로 되돌아가는 시점에서 중단될 때까지 활성으로 남아 있습니다.

세션 풀이 사용되는 방법 및 WebSphere Application Server와 IBM MQ 사이의 대화가 관리되는 방법에 대한 자세한 정보는 [473 페이지의 『JMS 연결 팩토리』](#)의 내용을 참조하십시오.

'WebSphere Application Server'을 사용한 ASF 및 비-ASF 모드 작동 방식과 비-ASF 모드를 사용하도록 리스너 포트를 구성하는 방법에 대한 자세한 내용은 [ASF 모드 및 비-ASF 모드에서의 메시지 처리를 참조](#)하십시오.

큐 목적지를 모니터링하는 동안 최대 대화 수 계산

비ASF 모드에서 실행하고 IBM MQ에 대한 큐 목적지를 모니터링하는 단일 리스너 포트의 최대 대화 수를 다음 공식으로 계산할 수 있습니다.

$$\text{Maximum number of conversations} = \text{Maximum sessions}$$

이러한 수의 대화가 발생하는 것을 허용하도록 작성될 채널 인스턴스의 수는 다음 계산을 사용하여 알 수 있습니다.

$$\text{Maximum number of channel instances} = \frac{\text{Maximum number of conversations}}{\text{SHARECNV for the channel being used}}$$

이 계산식에서 나머지는 반올림할 수 있습니다.

최대 세션 수 특성에 대해 기본값을 사용하고 큐 목적지를 모니터링하고 있는 Non-ASF 모드에서 실행 중인 단순 리스너 포트의 경우, 이 리스너에 대해 WebSphere Application Server와 IBM MQ 사이에 존재할 수 있는 최대 대화 수는 다음과 같습니다.

$$\text{Maximum number of conversations} = \text{Maximum sessions}$$

예를 들면, 다음과 같습니다.

$$= 10$$

이 리스너 포트가 '**SHARECNV**' 속성이 10으로 설정된 채널을 사용하여 'IBM MQ'에 연결되는 경우 생성되는 채널 인스턴스 수는 다음과 같이 계산됩니다:

$$\text{Maximum number of channel instances} = \frac{\text{Maximum number of conversations}}{\text{SHARECNV for the channel being used}}$$

예를 들면, 다음과 같습니다.

$$\begin{aligned} &= 10 / 10 \\ &= 1 \end{aligned}$$

토픽 목적지를 모니터링하는 동안 최대 대화 수 계산

Non-ASF 모드에서 실행 중이고 토픽 목적지를 모니터링하도록 구성된 리스너 포트의 경우 리스너 포트에서 IBM MQ로의 대화 수는 다음과 같습니다.

$$\text{Maximum number of conversations} = 1$$

이러한 수의 대화가 발생하는 것을 허용하도록 작성될 채널 인스턴스의 수는 다음 계산을 사용하여 알 수 있습니다.

$$\text{Maximum number of channel instances} = \frac{\text{Maximum number of conversations}}{\text{SHARECNV for the channel being used}}$$

이 계산식에서 나머지는 반올림할 수 있습니다.

최대 세션 수 특성에 대해 기본값을 사용하고 토픽 목적지를 모니터링하고 있는 Non-ASF 모드에서 실행 중인 단순 리스너 포트의 경우, 이 리스너에 대해 WebSphere Application Server와 IBM MQ 사이에 존재할 수 있는 최대 대화 수는 다음과 같습니다.

$$\text{Maximum number of conversations} = \text{Maximum sessions}$$

예를 들면, 다음과 같습니다.

$$= 10$$

이 리스너 포트가 'SHARECNV' 속성이 10으로 설정된 채널을 사용하여 'IBM MQ'에 연결되는 경우 생성되는 채널 인스턴스 수는 다음과 같이 계산됩니다:

$$\text{Maximum number of channel instances} = \frac{\text{Maximum number of conversations}}{\text{SHARECNV for the channel being used}}$$

예를 들면, 다음과 같습니다.

$$\begin{aligned} &= 10 / 10 \\ &= 1 \end{aligned}$$

IBM MQ 에 대한 WebSphere Application Server 연결을 보호하기 위한 인증 별명 구성

인증 별명은 WebSphere Application Server와 IBM MQ 사이의 보안 연결에 사용될 수 있는 사용자 이름과 비밀번호 조합에 매핑됩니다. 인증 별명으로 연결 팩토리를 구성할 수 있습니다.

엔터프라이즈 애플리케이션과 인증 별명 사용

내부에서 실행 중인 엔터프라이즈 WebSphere Application Server 애플리케이션이 에 대한 JMS 연결을 IBM MQ 생성하려고 시도할 때, 애플리케이션은 애플리케이션 서버의 (JNDI) Java Naming Directory Interface 저장소에서 메시징 공급자 연결 IBM MQ 팩토리 정의를 조회합니다.

IBM MQ 메시징 공급자 연결 팩토리 정의가 애플리케이션 서버의 JNDI 저장소 내에 위치할 때 다음 메소드 중 하나가 호출됩니다.

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

연결 팩토리가 정의된 J2C 인증 별명으로 구성되면 인증 별명의 사용자 이름과 비밀번호는 인증 팩토리가 연결 작성에 사용될 때 IBM MQ로 전달될 수 있습니다.

연결 팩토리 및 인증 별명

IBM MQ 메시징 공급자 연결 팩토리는 IBM MQ 큐 관리자에 연결하는 방법에 대한 정보를 포함합니다. 내부에서 실행되는 엔터프라이즈 애플리케이션은 연결 WebSphere Application Server 팩토리를 사용하여 에 대한 연결을 JMS IBM MQ 생성할 수 있습니다.

WebSphere Application Server는 JNDI를 사용하여 액세스할 수 있는 저장소에 연결 팩토리 정의를 저장합니다. 연결 팩토리가 작성되면 연결 팩토리에 연결 팩토리가 정의된 애플리케이션 서버 범위(셀, 노드 또는 서버 범위)에서 이를 고유하게 식별할 수 있는 JNDI 이름이 지정됩니다.

예를 들어, WebSphere Application Server 셀 범위에 정의된 IBM MQ 메시징 공급자 연결 팩토리는 BINDINGS 전송을 사용하여 큐 관리자(myQM)에 연결하는 방법에 관한 정보를 포함합니다. 이 연결 팩토리에는 고유하게 식별하기 위해 JNDI 이름 `jms/myCF`이(가) 지정됩니다.

연결 팩토리는 또한 인증 별명을 사용하도록 구성될 수 있습니다. 인증 별명은 사용자 이름과 비밀번호 조합에 매핑됩니다. 연결 팩토리의 사용 방식에 따라, 인증 별칭에 지정된 사용자 이름과 암호는 연결 JMS 생성 시 하위 IBM MQ 계층으로 전달될 수도 있고 전달되지 않을 수도 있습니다.

중요사항: IBM MQ 8.0 이전에서 기본 IBM MQ Object Authority Manager(OAM)는 인증 확인만 수행했습니다. 이는 연결이 작성될 때 IBM MQ에 전달된 사용자 이름이 큐 관리자에 대한 액세스 권한을 갖는지 확인하기 위한 목적이었습니다.

지정된 비밀번호가 올바른지 확인하기 위한 검사가 수행되지 않았습니다. 인증 확인을 수행하고 사용자 ID와 비밀번호가 일치하는지 검증하려면 IBM MQ 채널 보안 엑시트를 작성해야 합니다. 이를 수행하는 방법에 대한 자세한 내용은 [에서 확인할 수 717 페이지의 『채널 보안 엑시트 프로그램』](#) 있습니다.

IBM MQ 8.0 이후부터는 큐 관리자가 사용자 이름 외에 비밀번호도 확인합니다.

연결 팩토리 사용

다음 주제에서는 직접 및 간접 검색을 사용하여 연결 팩토리를 사용하는 방법에 대한 정보를 포함합니다.

- 481 페이지의 『직접 검색을 통해 연결 팩토리 사용』
- 482 페이지의 『간접 검색을 통해 연결 팩토리 사용』

CLIENT 전송 사용

CLIENT 전송을 사용하도록 구성된 연결 팩토리는 큐 관리자에 연결하기 위해 사용하려는 IBM MQ SVECONN(Server Connection Channel)을 지정해야 합니다.

IBM MQ 채널 에이전트 사용자 ID(MCAUSER) 특성은 연결 팩토리가 사용하도록 구성된 채널에 대해 공백을 유지한 다음 연결 팩토리가 직접 검색 또는 간접 검색 중 하나와 사용될 수 있습니다.

MCAUSER 특성이 사용자 ID에 설정되면 엔터프라이즈 애플리케이션이 직접 또는 간접 검색을 사용하는지에 관계없이 연결 팩토리를 사용하여 IBM MQ에 대한 연결을 작성할 때 이 사용자 ID가 IBM MQ에 전달됩니다.

요약 표

다음 표에서는 BINDINGS 전송과 CLIENT 전송이 각각 사용될 때 사용자 ID가 IBM MQ로 전달되는 내용을 요약합니다.

표 72. BINDINGS 모드		
구성	애플리케이션 호출 <code>ConnectionFactory.createConnection()</code>	애플리케이션 호출 <code>ConnectionFactory.createConnection(String username, String password)</code>
애플리케이션의 배치 스크립트는 연결 팩토리에 대한 자원 참조를 포함하지 않습니다.	애플리케이션 서버 프로세스에 대한 사용자 ID가 IBM MQ로 전달됩니다.	<code>ConnectionFactory.createConnection(String username, String password)</code> 메소드로 전달된 사용자 ID와 비밀번호는 IBM MQ에 전달됩니다.
애플리케이션의 배치 스크립트는 연결 팩토리에 대한 자원 참조를 포함하며 res-auth 특성이 "Application"으로 설정됩니다.	애플리케이션 서버 프로세스에 대한 사용자 ID가 IBM MQ로 전달됩니다.	<code>ConnectionFactory.createConnection(String username, String password)</code> 메소드로 전달된 사용자 ID와 비밀번호는 IBM MQ에 전달됩니다.
애플리케이션의 배치 스크립트는 연결 팩토리에 대한 자원 참조를 포함하며 res-auth 특성이 "Container"로 설정됩니다.	연결 팩토리에 대한 인증 별명에 지정된 사용자 ID와 비밀번호는 IBM MQ에 전달됩니다.	연결 팩토리에 대한 인증 별명에 지정된 사용자 ID와 비밀번호는 IBM MQ에 전달됩니다.
애플리케이션의 배치 스크립트는 res-auth 특성이 "Container"로 설정된 연결 팩토리에 대한 자원 참조를 포함하고 애플리케이션은 인증 별명으로 구성되어 있습니다.	애플리케이션이 사용하도록 구성된 인증 별명에 지정된 사용자 ID와 비밀번호는 IBM MQ에 전달됩니다.	애플리케이션이 사용하도록 구성된 인증 별명에 지정된 사용자 ID와 비밀번호는 IBM MQ에 전달됩니다.

표 73. CLIENT 모드

구성	애플리케이션 호출 ConnectionFactory.createC onnection()	애플리케이션 호출 ConnectionFactory.createC onnection(String username, String password)
애플리케이션의 배치 디스크립터는 연결 팩토리에 대한 자원 참조를 포함하지 않으며 연결 팩토리는 MCAUSER 특성이 설정되지 않은 IBM MQ 채널을 사용하도록 구성됩니다.	애플리케이션 서버 프로세스에 대한 사용자 ID가 IBM MQ로 전달됩니다.	ConnectionFactory.createC onnection(String username, String password) 메소드로 전달된 사용자 ID와 비밀번호는 IBM MQ에 전달됩니다.
애플리케이션의 배치 디스크립터는 연결 팩토리에 대한 자원 참조를 포함하지 않으며 연결 팩토리는 MCAUSER 특성이 사용자 ID로 설정된 IBM MQ 채널을 사용하도록 구성됩니다.	채널 팩토리가 사용하도록 구성된 IBM MQ 채널의 MCAUSER 특성으로 지정된 사용자 ID가 IBM MQ로 전달됩니다.	채널 팩토리가 사용하도록 구성된 IBM MQ 채널의 MCAUSER 특성으로 지정된 사용자 ID가 IBM MQ로 전달됩니다.
애플리케이션의 배치 디스크립터는 res-auth 특성이 <i>Application</i> 으로 설정된 연결 팩토리에 대한 자원 참조를 포함하며 연결 팩토리는 MCAUSER 특성이 설정되지 않은 IBM MQ 채널을 사용하도록 구성됩니다.	애플리케이션 서버 프로세스에 대한 사용자 ID가 IBM MQ로 전달됩니다.	ConnectionFactory.createC onnection(String username, String password) 메소드로 전달된 사용자 ID와 비밀번호는 IBM MQ에 전달됩니다.
애플리케이션의 배치 디스크립터는 res-auth 특성이 <i>Application</i> 으로 설정된 연결 팩토리에 대한 자원 참조를 포함하며 연결 팩토리는 MCAUSER 특성이 사용자 ID로 설정된 IBM MQ 채널을 사용하도록 구성됩니다.	채널 팩토리가 사용하도록 구성된 IBM MQ 채널의 MCAUSER 특성으로 지정된 사용자 ID가 IBM MQ로 전달됩니다.	채널 팩토리가 사용하도록 구성된 IBM MQ 채널의 MCAUSER 특성으로 지정된 사용자 ID가 IBM MQ로 전달됩니다.
애플리케이션의 배치 디스크립터는 res-auth 특성이 <i>Container</i> 로 설정된 연결 팩토리에 대한 자원 참조를 포함하며 연결 팩토리는 MCAUSER 특성이 설정되지 않은 IBM MQ 채널을 사용하도록 구성됩니다.	연결 팩토리에 대한 인증 별명에 지정된 사용자 ID와 비밀번호는 IBM MQ에 전달됩니다.	연결 팩토리에 대한 인증 별명에 지정된 사용자 ID와 비밀번호는 IBM MQ에 전달됩니다.
애플리케이션의 배치 디스크립터는 res-auth 특성이 <i>Container</i> 로 설정된 연결 팩토리에 대한 자원 참조를 포함하며 연결 팩토리는 MCAUSER 특성이 사용자 ID로 설정된 IBM MQ 채널을 사용하도록 구성됩니다.	채널 팩토리가 사용하도록 구성된 IBM MQ 채널의 MCAUSER 특성으로 지정된 사용자 ID가 IBM MQ로 전달됩니다.	채널 팩토리가 사용하도록 구성된 IBM MQ 채널의 MCAUSER 특성으로 지정된 사용자 ID가 IBM MQ로 전달됩니다.

표 73. CLIENT 모드 (계속)

구성	애플리케이션 호출 <code>ConnectionFactory.createConnection()</code>	애플리케이션 호출 <code>ConnectionFactory.createConnection(String username, String password)</code>
애플리케이션의 배치 디스크립터는 res-auth 특성이 <i>Container</i> 로 설정된 연결 팩토리에 대한 자원 참조를 포함하고 애플리케이션은 인증 별명으로 구성되어 있으며 연결 팩토리는 MCAUSER 특성이 설정되지 않은 IBM MQ 채널을 사용하도록 구성됩니다.	애플리케이션이 사용하도록 구성된 인증 별명에 지정된 사용자 ID와 비밀번호는 IBM MQ에 전달됩니다.	애플리케이션이 사용하도록 구성된 인증 별명에 지정된 사용자 ID와 비밀번호는 IBM MQ에 전달됩니다.
애플리케이션의 배치 디스크립터는 res-auth 특성이 <i>Container</i> 로 설정된 연결 팩토리에 대한 자원 참조를 포함하고 애플리케이션은 인증 별명으로 구성되어 있으며 연결 팩토리는 MCAUSER 특성이 사용자 ID로 설정된 IBM MQ 채널을 사용하도록 구성됩니다.	채널 팩토리가 사용하도록 구성된 IBM MQ 채널의 MCAUSER 특성으로 지정된 사용자 ID가 IBM MQ로 전달됩니다.	채널 팩토리가 사용하도록 구성된 IBM MQ 채널의 MCAUSER 특성으로 지정된 사용자 ID가 IBM MQ로 전달됩니다.

직접 검색을 통해 연결 팩토리 사용

메시징 공급자 IBM MQ 연결 팩토리가 정의된 후, 엔터프라이즈 애플리케이션은 연결 팩토리 정의를 조회하여 이를 활용해 메시징 큐 IBM MQ 매니저에 대한 연결을 JMS 생성할 수 있습니다. 이는 직접 검색을 통해 수행될 수 있습니다.

직접 검색을 사용하기 위해 엔터프라이즈 애플리케이션은 다음 메소드 호출을 작성하여 애플리케이션 서버의 JNDI 저장소에 연결합니다.

```
InitialContext ctx = new InitialContext();
```

일단 JNDI 저장소에 연결하면 엔터프라이즈 애플리케이션이 다음과 같이 연결 팩토리의 JNDI 이름을 사용하여 연결 팩토리 정의를 식별합니다.

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("jms/myCF");
```

참고:

- 애플리케이션 개발자는 엔터프라이즈 애플리케이션을 개발할 때 필수 연결 팩토리의 JNDI 이름을 알아야 합니다. JNDI 이름이 애플리케이션 내부에 하드 코딩되기 때문에 JNDI 이름이 변경되면 애플리케이션을 다시 작성하고 다시 배치해야 합니다.
- 연결 팩토리 정의가 이러한 방식으로 사용되면 (연결 팩토리가 사용하도록 구성된) 인증 별명에 지정된 사용자 이름과 비밀번호가 IBM MQ에 전달되지 않습니다. 이는 권한 없는 애플리케이션이 연결 팩토리를 식별하는 것과 이를 사용하여 보안 IBM MQ 시스템에 연결하는 것을 방지하기 위해서입니다.

IBM MQ에 전달된 사용자 이름과 비밀번호는 연결 팩토리에서 JMS 연결을 작성하는 데 사용된 메소드에 따라 달라집니다.

애플리케이션이 다음 방법을 사용하여 JMS 연결을 작성하는 경우:

```
ConnectionFactory.createConnection()
```

기본 사용자 ID가 IBM MQ에 전달됩니다. 이는 엔터프라이즈 애플리케이션을 실행 중인 애플리케이션 서버를 시작한 사용자 이름과 비밀번호입니다.

대체 방법으로, 애플리케이션이 다음 메소드를 호출하여 JMS 연결을 작성할 수 있습니다.

```
ConnectionFactory.createConnection(String username, String password)
```

애플리케이션이 연결 팩토리의 직접 검색을 수행하고 이 메소드를 호출한 경우, `createConnection()` 메소드에 전달된 사용자 이름과 비밀번호가 IBM MQ에 전달됩니다.

중요사항: IBM MQ 8.0 이전 버전에서는, 전달된 사용자 이름이 큐 관리자에 액세스할 수 있는 권한이 있는지 확인하기 위해서만 IBM MQ가 권한 검사를 수행했습니다.

비밀번호는 검사하지 않습니다. 인증 확인을 수행하고 사용자 이름과 비밀번호가 유효한지 검증하려면 IBM MQ 채널 보안 엑시트를 작성해야 합니다. 이를 수행하는 방법에 대한 자세한 내용은 [에서 확인할 수 717 페이지의 『채널 보안 엑시트 프로그램』](#) 있습니다.

IBM MQ 8.0 이후부터는 큐 관리자가 사용자 이름 외에 비밀번호도 확인합니다.

간접 검색을 통해 연결 팩토리 사용

엔터프라이즈 애플리케이션을 작성할 때 연결 팩토리의 JNDI 이름을 알 수 없거나, (설치된 애플리케이션 서버에 따라) 다른 JNDI 이름으로 다른 연결 팩토리를 사용하여 애플리케이션을 다른 애플리케이션 서버에 설치하는 경우, 자원 참조를 사용하여 연결 팩토리를 검색할 수 있습니다. 이는 간접 검색을 통해 수행될 수 있습니다.

예

'jms/myCF' 을 사용하여 연결 팩토리를 직접 조회하는 대신 리소스 참조가 포함된 엔터프라이즈 애플리케이션은 로컬 'JNDI' 이름을 가집니다: jms/myResourceReferenceCF 입니다.

이 JNDI 이름을 사용하기 위해 애플리케이션이 직접 검색을 수행하는 것과 동일한 방식으로 애플리케이션은 애플리케이션 서버의 JNDI 저장소에 연결합니다.

```
InitialContext ctx = new InitialContext();
```

애플리케이션은 이제 jms/myCF를 직접 식별하지 않고 자원 참조의 JNDI 이름을 식별합니다.

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("java:comp/env/jms/myResourceReferenceCF");
```

엔터프라이즈 애플리케이션이 간접 조회를 수행 중임을 애플리케이션 서버에 알려려면 로컬 'JNDI' 이름에 'java:comp/env' 접두사를 붙여야 합니다.

애플리케이션이 배치되면 사용자는 자원 참조의 JNDI 이름 jms/myResourceReferenceCF를, 애플리케이션이 이미 작성한 연결 팩토리의 JNDI 이름 jms/myCF에 매핑합니다.

애플리케이션이 실행되면 애플리케이션 서버가 매핑하는 로컬 'JNDI' 이름을 사용하여 'JMS' 연결 팩토리를 조회합니다: jms/myCF. 그런 다음 IBM MQ에 대한 연결을 작성하기 위해 애플리케이션에서 이 연결 팩토리를 사용합니다.

인증 별명과 간접 검색

자원 참조는 또한 제공된 연결 팩토리의 작동을 변경하는 추가 특성을 정의할 수 있게 허용합니다. 자원 참조의 특성 중 하나는 **res-auth**입니다. 이 특성의 값은 IBM MQ에 대한 연결을 작성할 때 엔터프라이즈 애플리케이션이 자원 참조가 매핑하는 연결 팩토리의 인증 별명을 사용해야 하는지(인증 별명이 정의된 경우) 또는 애플리케이션이 자체 사용자 이름 및 비밀번호를 지정하는지 여부를 지정합니다.

이 특성의 기본값은 *Application*입니다. 이는 JMS 연결이 작성될 때 큐 관리자로 전달되는 사용자 이름 및 비밀번호를 애플리케이션 자체에서 결정함을 의미합니다. 자원 참조가 매핑되는 연결 팩토리의 인증 별명은 사용되지 않습니다.

애플리케이션은 다음 메소드 중 하나를 사용하여 JMS 연결을 작성할 수 있습니다.

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

애플리케이션이 `ConnectionFactory.createConnection()`을 사용하고 **res-auth**가 *Application*으로 설정되면 기본 사용자 ID가 IBM MQ에 전달됩니다. 이는 엔터프라이즈 애플리케이션을 실행 중인 애플리케이션 서버를 시작한 사용자 이름과 비밀번호입니다.

애플리케이션이 `ConnectionFactory.createConnection(String username, String password)`을 사용하고 **res-auth**가 애플리케이션으로 설정된 경우 메서드에 전달된 사용자 이름과 비밀번호는 'IBM MQ'으로 전송됩니다.

연결을 작성할 때 자원 참조가 맵핑하는 연결 팩토리에 정의된 인증 별명을 사용하기 위해서는 **res-auth** 특성을 값 *Container*로 설정해야 합니다. 애플리케이션이 JMS 연결을 작성할 때 `createConnection` 호출이 사용자 이름과 비밀번호를 지정한 경우에도, 인증 별명 세부사항이 사용됩니다.

간접 검색을 사용할 때 인증 별명 대체

res-auth 특성이 *Container*로 설정된 자원 참조를 애플리케이션이 사용하는 경우 JMS 연결이 작성될 때 사용되는 인증 별명을 대체할 수 있습니다.

인증 별명을 대체하려면 애플리케이션이 배치될 애플리케이션 서버 환경에 이미 작성되어 있는 기존 인증 별명에 맵핑하는 **authDataAlias**이라는 추가 특성을 자원 참조에 포함해야 합니다. 'IBM'에서 제공하는 'Rational®' 툴을 사용하여 생성된 모든 리소스 레퍼런스에 이 프로퍼티를 지정할 수 있습니다.

이 메소드를 사용하여, 간접 검색된 JMS 연결 팩토리를 사용할 때 다른 인증 별명을 사용할 수 있습니다. 지정된 인증 별명이 존재하지 않으면 엔터프라이즈 애플리케이션이 설치된 후에 새 인증 별명을 지정할 수 있습니다. 자세한 내용은 'WebSphere Application Server 제품 설명서에서 리소스 참조를 참조하세요.

WebSphere Application Server 8.5.5에 대한 관련 정보

[자원 참조](#)

WebSphere Application Server 8.0에 대한 관련 정보

[자원 참조](#)

WebSphere Application Server 7.0에 대한 관련 정보

[자원 참조](#)

WebSphere Application Server 클러스터를 사용할 때 메시지 구동 Bean의 워크로드 밸런싱

WebSphere Application Server 7.0 및 WebSphere Application Server 8.0 클러스터에 배치되고 IBM MQ 메시징 제공자 정상 모드에서 실행하도록 구성된 MDB 애플리케이션을 사용할 때 클러스터 멤버 중 하나가 메시지의 대부분을 처리합니다. 둘 이상의 클러스터 멤버에서 메시지 처리를 분산시키기 위해 클러스터 멤버의 워크로드의 균형을 맞출 수 있습니다.

IBM MQ에는 라는 기능이 포함되어 **Asynchronous consume** 있으며, 이를 통해 애플리케이션은 및 **MQCB** 라는 API를 사용하여 큐에서 메시지를 **MQCTL** 비동기적으로 소비할 수 있습니다.

내부에서 실행되는 메시지 주도 빈 애플리케이션은 WebSphere Application Server 7.0 WebSphere Application Server 8.0 메시징 공급자의 일반 IBM MQ 모드를 사용하는 경우 이 기능을 자동으로 활용합니다. 애플리케이션이 시작되면, 모니터링하도록 구성된 대상에 JMS 대해 비동기 소비자(asynchronous consumer)를 설정합니다. 이를 위해 해당 애플리케이션은 **MQCB**. 그런 다음 **MQCTL** API가 호출되어 애플리케이션이 JMS 목적지로부터 메시지를 수신할 준비가 되어 있음을 표시합니다.

MDB 애플리케이션이 WebSphere Application Server 클러스터에 배치되어 있으면 각 클러스터 멤버는 MDB가 메시지를 모니터링하는 JMS 목적지에 대해 비동기 처리자를 설정합니다. 해당 JMS 목적지를 호스팅하는 큐 IBM MQ 관리자는 클러스터 구성원이 처리할 수 있는 적절한 메시지가 목적지에 JMS 도착했을 때 이를 구성원에게 알리는 역할을 담당합니다.

큐 IBM MQ 매니저에 연결할 WebSphere Application Server 때, 특정 JMS 대상에 도착한 메시지는 해당 JMS 대상에 등록된 모든 비동기 소비자들에게 더 균등하게 분배됩니다. WebSphere Application Server 7.0 및 WebSphere Application Server 8.0 클러스터 내부에 배치된 MDB 애플리케이션의 경우 이는 메시지가 클러스터 멤버 사이에서 더 균등하게 분배됨을 의미합니다.

관련 태스크

[JMS PROVIDERVERSION 특성 구성](#)

IBM MQ 헤더 패키지 사용

IBM MQ 헤더 패키지는 메시지의 IBM MQ 헤더를 조작하기 위해 사용할 수 있는 헬퍼 인터페이스 및 클래스 세트를 제공합니다. 일반적으로 사용자가 명령 서버를 사용하여(PCF(Programmable Command Format) 메시지를 사용하여) 관리 서비스를 수행하기를 원하므로 IBM MQ 헤더 패키지를 사용합니다.

이 태스크 정보

IBM MQ 헤더 패키지는 `com.ibm.mq.headers` 및 `com.ibm.mq.headers.pcf` 패키지에 있습니다. 이 기능을 애플리케이션에서 Java 사용하기 위해 IBM MQ 제공하는 두 가지 대체 API 모두에 사용할 수 있습니다:

- IBM MQ classes for Java(IBM MQ 기반 Java라고도 함).
- IBM MQ classes for Java Message Service(IBM MQ classes for JMS, IBM MQ JMS라고도 함).

IBM MQ 기반 Java 애플리케이션은 IBM MQ 기반 Java 인터페이스를 근본적으로 이해하므로 일반적으로 `MQMessage` 오브젝트를 조작하며, 헤더 지원 클래스는 이러한 오브젝트와 직접 상호작용할 수 있습니다.

IBM MQ JMS에서는 메시지에 대한 페이로드가 일반적으로 `DataInput` 및 `DataOutput` 스트림으로 조작될 수 있는 문자열 또는 바이트 배열 오브젝트입니다. IBM MQ 헤더 패키지는 이러한 데이터 스트림과 상호작용하는 데 사용할 수 있으며 IBM MQ JMS 애플리케이션에 의해 송수신되는 MQ 메시지를 조작하는 데 적합합니다.

그러므로 IBM MQ 헤더 패키지에 IBM MQ 기반 Java 패키지에 대한 참조가 포함되어 있더라도 이는 IBM MQ JMS 애플리케이션 내에서 사용하기 위한 용도이기도 하며 Java Platform, Enterprise Edition(Java EE) 환경 내에서 사용하기에 적합합니다.

IBM MQ 헤더 패키지를 사용할 수 있는 일반적인 방법은 예를 들면 다음과 같은 이유를 위해 PCF(Programmable Command Format)에서 관리 메시지를 조작하는 것입니다.

- IBM MQ 자원에 대한 세부사항에 액세스합니다.
- 큐의 깊이를 모니터링합니다.
- 큐에 대한 액세스를 금합니다.

PCF 메시지를 API와 IBM MQJMS 함께 사용함으로써, 이러한 애플리케이션 중심 리소스 관리 작업을 기본 JavaIBM MQ API를 사용하지 않고도 애플리케이션 Java EE 내부에서 수행할 수 있습니다.

프로시저

- IBM MQ 헤더 패키지를 사용하여 IBM MQ classes for Java에 대한 메시지 헤더를 조작하려면 [484 페이지의 『IBM MQ classes for Java로 사용』](#)의 내용을 참조하십시오.
- IBM MQ 헤더 패키지를 사용하여 IBM MQ classes for JMS에 대한 메시지 헤더를 조작하려면 [485 페이지의 『IBM MQ classes for JMS로 사용』](#)의 내용을 참조하십시오.

IBM MQ classes for Java로 사용

IBM MQ classes for Java 애플리케이션은 일반적으로 `MQMessage` 오브젝트를 조작하며, 헤더 지원 클래스는 IBM MQ classes for Java 인터페이스를 근본적으로 파악하므로 이러한 오브젝트와 직접 상호작용할 수 있습니다.

이 태스크 정보

IBM MQ 헤더 IBM MQ 패키지를 IBM MQ Base Java API(IBM MQ classes for Java)와 함께 사용하는 방법을 보여주는 몇 가지 샘플 애플리케이션을 제공합니다.

샘플은 다음 두 가지 방법을 보여줍니다.

- 관리 조치를 수행하고 자원 메시지를 구문 분석하기 위해 PCF 메시지를 작성하는 방법.
- IBM MQ classes for Java를 사용하여 이 PCF 메시지를 송신하는 방법.

사용 중인 플랫폼에 따라, 이러한 샘플은 IBM MQ 설치의 `samples` 또는 `tools` 디렉토리에 있는 `pcf` 디렉토리에 설치됩니다([327 페이지의 『IBM MQ classes for Java의 설치 디렉토리』](#) 참조).

프로시저

1. 관리 조치를 수행하고 자원 메시지를 구문 분석하기 위해 PCF 메시지를 작성하십시오.
2. IBM MQ classes for Java를 사용하여 이 PCF 메시지를 송신하십시오.

관련 개념

351 페이지의 『IBM MQ classes for Java 를 사용하여 IBM MQ 메시지 헤더 처리』

여러 다른 유형의 메시지 헤더를 표시하는 Java 클래스가 제공됩니다. 두 개의 헬퍼 클래스도 제공됩니다.

356 페이지의 『IBM MQ classes for Java로 PCF 메시지 핸들링』

Java 클래스는 PCF 구조화된 메시지를 작성하고 구문 분석하며 쉽게 PCF 요청을 송신하고 PCF 응답을 수집하기 위해 제공됩니다.

IBM MQ classes for JMS로 사용

IBM MQ 헤더를 IBM MQ classes for JMS와 함께 사용하기 위해 IBM MQ classes for Java에 대해서와 동일한 필수 단계를 수행합니다. PCF 메시지를 작성할 수 있으며 응답은 IBM MQ 헤더 패키지 및 IBM MQ classes for Java에 대해서와 동일한 샘플 코드를 사용하여 동일한 방법으로 구문 분석될 수 있습니다.

이 태스크 정보

IBM MQ API를 사용하여 PCF 메시지를 송신하려면 메시지 페이로드가 JMS 바이트 메시지에 기록되고 표준 JMS API를 사용하여 송신되어야 합니다. 유일한 고려사항은 메시지에 JMS RFH2 또는 다른 헤더가 MQMD의 특정 값과 함께 포함되지 않아야 한다는 점입니다.

PCF 메시지를 송신하려면 다음 단계를 완료하십시오. PCF 메시지가 작성되는 방법 및 응답 메시지에서 정보가 추출되는 방법은 IBM MQ classes for Java의 경우와 동일합니다(484 페이지의 『IBM MQ classes for Java로 사용』 참조).

프로시저

1. SYSTEM.ADMIN.COMMAND.QUEUE를 나타내는 JMS Queue Destination을 작성하십시오.

IBM MQ JMS 애플리케이션은 PCF 메시지를 SYSTEM.ADMIN.COMMAND.QUEUE에 송신하며, 이 큐를 나타내는 JMS 목적지 오브젝트에 대한 액세스가 필요합니다. 목적지는 다음 특성 세트를 가져야 합니다.

```
WMQ_MQMD_WRITE_ENABLED = YES
WMQ_MESSAGE_BODY = MQ
```

WebSphere Application Server를 사용 중인 경우, 이러한 특성을 목적지의 사용자 정의 특성으로 정의해야 합니다.

애플리케이션 내에서 프로그래밍 방식으로 목적지를 작성하려면 다음 코드를 사용하십시오.

```
Queue q1 = session.createQueue("SYSTEM.ADMIN.COMMAND.QUEUE");
((MQQueue) q1).setIntProperty(WMQConstants.WMQ_MESSAGE_BODY,
    WMQConstants.WMQ_MESSAGE_BODY_MQ);
((MQQueue) q1).setMQMDWriteEnabled(true);
```

2. PCF 메시지를 올바른 MQMD 값이 포함된 JMS 바이트 메시지로 변환하십시오.

JMS 바이트 메시지를 작성해야 하며 PCF 메시지를 이에 기록해야 합니다. 응답 큐를 작성해야 하지만 특정 설정을 갖지 않아야 합니다.

다음 샘플 코드 스니펫은 JMS 바이트 메시지를 작성하고 com.ibm.mq.headers.pcf.PCFMessage 오브젝트를 이에 기록하는 방법을 보여줍니다. PCFMessage 오브젝트(pcfCmd)는 IBM MQ 헤더 패키지를 사용하여 이전에 빌드되었습니다 (PCFMessage를 로드하기 위한 패키지는 com.ibm.mq.headers.pcf.PCFMessage임).

```
// create the JMS Bytes Message
final BytesMessage msg = session.createBytesMessage();

// Create the wrapping streams to put the bytes into the message payload
ByteArrayOutputStream baos = new ByteArrayOutputStream();
DataOutput dataOutput = new DataOutput(baos);

// Set the JMSReplyTo so the answer comes back
```

```

msg.setJMSReplyTo(new MQQueue("adminResp"));

// write the pcf into the stream
pcfCmd.write(dataOutput);
baos.flush();
msg.writeBytes(baos.toByteArray());

// we have taken control of the MD, so need to set all
// flags in the MD that we require - main one is the format
msg.setJMSPriority(4);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_PERSISTENCE,
    CMQC.MQPER_NOT_PERSISTENT);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_EXPIRY, 300);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_REPORT,
    CMQC.MQRO_PASS_CORREL_ID);
msg.setStringProperty(WMQConstants.JMS_IBM_MQMD_FORMAT, "MQADMIN");

// and send the message
sender.send(msg);

```

3. 메시지를 송신하고, 표준 JMS API를 사용하여 응답을 수신하십시오.

4. 처리를 위해 응답 메시지를 PCF 메시지로 변환하십시오.

응답 메시지를 검색하고 이를 PCF 메시지로 처리하려면 다음 코드를 사용하십시오.

```

// Get the message back
BytesMessage msg = (BytesMessage) consumer.receive();

// get the size of the bytes message & read into an array
int bodySize = (int) msg.getBodyLength();
byte[] data = new byte[bodySize];
msg.readBytes(data);

// Read into Stream and DataInput Stream
ByteArrayInputStream bais = new ByteArrayInputStream(data);
DataInput dataInput = new DataInputStream(bais);

// Pass to PCF Message to process
PCFMessage response = new PCFMessage(dataInput);

```

관련 개념

140 페이지의 『JMS 메시지』

JMS 메시지는 헤더, 특성 및 본문으로 구성됩니다. JMS는 5가지 유형의 메시지 본문을 정의합니다.

IBM i 설정하기 IBM MQ 에 IBM i 와 Java 함께 JMS

이 주제 모음은 CL 명령어나 qshell 환경을 사용하여 IBM i 에서 JavaJMS 와 를 설정하고 IBM MQ 테스트하는 방법에 대한 개요를 제공합니다.

참고:

- IBM MQ 8.0부터, ldap.jar, jndi.jar 및 jta.jar은(는) JDK의 일부입니다.
- **JM 3.0** IBM MQ 9.3.0부터, Jakarta Messaging 3.0 는 새로운 애플리케이션 개발에 지원됩니다. IBM MQ 9.3.0 그리고 향후에도 기존 애플리케이션에 대한 지원을 JMS 2.0 계속할 예정입니다. 동일한 애플리케이션에서 두 JMS 2.0Jakarta Messaging 3.0 API를 모두 사용하는 것은 지원되지 않습니다. 자세한 내용은 'JMS/Jakarta Messaging용 IBM MQ 클래스 사용'을 참조하십시오.

CL 명령 사용

설정된 CLASSPATH는 MQ 기반 테스트, JNDI를 Java사용한 JMS 테스트, JNDI를 사용하지 않은 JMS 테스트를 위한 것입니다.

디렉터리 /home/Userprofile 아래에 파일을 .profile 사용하지 않는 경우, 시스템 수준 환경 변수를 설정해야 합니다. 이러한 변수가 **WRKENVVAR** 명령을 사용하여 설정되었는지 확인할 수 있습니다.

1. 전체 시스템에 대한 환경 변수를 보려면 다음 명령을 발행하십시오. **WRKENVVAR LEVEL(*SYS)**
2. 사용자 작업의 고유 환경 변수를 보려면 다음 명령을 발행하십시오. **WRKENVVAR LEVEL(*JOB)**
3. CLASSPATH가 설정되지 않은 경우 다음 명령을 실행하십시오:

JM 3.0

```
ADDENVVAR ENVVAR(CLASSPATH)
VALUE('.:QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
:QIBM/ProdData/mqm/java/lib/connector.jar:QIBM/ProdData/mqm/java/lib
:QIBM/ProdData/mqm/java/samples/base
:QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar
:QIBM/ProdData/mqm/java/lib/jms.jar
:QIBM/ProdData/mqm/java/lib/providerutil.jar
:QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

JMS2.0

```
ADDENVVAR ENVVAR(CLASSPATH)
VALUE('.:QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
:QIBM/ProdData/mqm/java/lib/connector.jar:QIBM/ProdData/mqm/java/lib
:QIBM/ProdData/mqm/java/samples/base
:QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar
:QIBM/ProdData/mqm/java/lib/jms.jar
:QIBM/ProdData/mqm/java/lib/providerutil.jar
:QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

4. QIBM_MULTI_THREADED가 설정되지 않은 경우 다음 명령을 발행하십시오.

```
ADDENVVAR ENVVAR(QIBM_MULTI_THREADED) VALUE('Y') LEVEL(*SYS)
```

5. QIBM_USE_DESCRIPTOR_STDIO가 설정되지 않은 경우 다음 명령을 발행하십시오.

```
ADDENVVAR ENVVAR(QIBM_USE_DESCRIPTOR_STDIO) VALUE('I') LEVEL(*SYS)
```

6. QSH_REDIRECTION_TEXTDATA가 설정되지 않은 경우 다음 명령을 발행하십시오.

```
ADDENVVAR ENVVAR(QSH_REDIRECTION_TEXTDATA) VALUE('Y') LEVEL(*SYS)
```

qshell 환경 사용

Qshell 환경을 사용하는 경우 /home/Userprofile 디렉토리에 .profile을(를) 설정할 수 있습니다. 자세한 정보는 Qshell Interpreter(qsh) 문서를 참조하십시오.

.profile에 다음을 지정하십시오. CLASSPATH문은 단일 행에 있어야 하며 다음과 같이 \ 문자를 사용할 경우 다른 행으로 분리할 수 있음에 유의하십시오.

JM 3.0

```
CLASSPATH=.:QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

JMS2.0

```
CLASSPATH=.:QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
```

```
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

DSPLIBL 명령을 발행하여 QMQMJAVA 라이브러리가 라이브러리 목록에 있는지 확인하십시오.

QMQMJAVA 라이브러리가 해당 목록에 없으면, 다음 명령을 사용하여 이를 추가하십시오. **ADDLIB LIB(QMQMJAVA)**

▶ IBM i Java로 IBM i에서 IBM MQ 테스트

MQIVP 샘플 프로그램을 사용하여 Java IBM MQ 테스트하는 방법.

IBM MQ 기반 Java 테스트

다음 프로시저를 수행하십시오.

1. 다음 명령을 발행하여 큐 관리자가 시작되었는지와 큐 관리자의 상태가 활성인지 확인하십시오.

```
WRKMQM MQMNAME(QMGRNAME)
```

2. 다음 명령을 발행하여 JAVA.CHANNEL 서버 연결 채널이 작성되었는지 확인하십시오.

```
WRKMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
```

- a. JAVA.CHANNEL이 없는 경우 다음 명령을 발행하십시오.

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
```

3. **WRKMQMLSR** 명령을 발행하여 사용 중인 포트 또는 포트 1414에 대해 큐 관리자 리스너가 실행 중인지 확인하십시오.

- a. 리스너가 큐 관리자에 대해 시작되지 않은 경우 다음 명령을 발행하십시오.

```
STRMQMLSR PORT(XXXX) MQMNAME(QMGRNAME)
```

MQIVP 샘플 테스트 프로그램 실행

1. STRQSH 명령을 발행하여 명령행에서 qshell을 시작하십시오.
2. 올바른 CLASSPATH가 **export** 명령을 발행하여 설정되었는지 확인한 후에 다음과 같이 **cd** 명령을 발행하십시오.

```
cd /qibm/proddata/mqm/java/samples/wmqjava/samples
```

3. 다음 명령을 발행하여 **java** 프로그램을 실행하십시오.

```
java MQIVP
```

다음에 대한 프롬프트가 표시될 때 ENTER 키를 누르면

- 연결 유형
- IP 주소
- 큐 관리자 이름

기본값을 사용합니다. 이는 제품 바인딩을 확인하며, 이는 QMQMJAVA 라이브러리에 있습니다.

다음 예제와 유사한 출력을 수신합니다. 저작권 문항은 사용 중인 제품의 버전에 따라 다름에 유의하십시오.

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2026. All Rights Reserved.
=====

Please enter the IP address of the MQ server :>
Please enter the queue manager name :>
Attaching Java program to QIBM/ProdData/mqm/java/lib/connector.JAR.
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
```

```
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager
```

```
Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

IBM MQ Java 클라이언트 연결 테스트

다음을 지정해야 합니다.

- 연결 유형
- IP 주소
- 포트
- 서버 연결 채널
- 큐 관리자

다음 예제와 유사한 출력을 수신합니다. 저작권 문항은 사용 중인 제품의 버전에 따라 다름에 유의하십시오.

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2026. All Rights Reserved.
=====

Please enter the IP address of the MQ server :> x.xx.xx.xx
Please enter the port to connect to : (1414)> 1470
Please enter the server connection channel name :> JAVA.CHANNEL
Please enter the queue manager name :> KAREN01
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager

Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

IBM i JMS로 IBM i에서 IBM MQ 테스트

JNDI 사용 JMS 시와 미사용 IBM MQ 시 테스트 방법

IVTRun 샘플을 사용하여 JNDI 없이 JMS 테스트

다음 프로시저를 수행하십시오.

1. 다음 명령을 발행하여 큐 관리자가 시작되었는지와 큐 관리자의 상태가 활성인지 확인하십시오.

```
WRKMQM MQMNAME(QMGRNAME)
```

2. **STRQSH** 명령을 발행하여 명령행에서 qshell을 시작하십시오.
3. 다음과 같이 디렉토리를 변경하려면 **cd** 명령을 사용하십시오.

```
cd /qibm/proddata/mqm/java/bin
```

4. 스크립트 파일을 실행하십시오.

```
IVTRun -nojndi [-m qmgrname]
```

다음 예제와 유사한 출력을 수신합니다. 저작권 설명은 사용 중인 제품의 버전에 따라 다릅니다.

```
IVTRun -nojndi -m ELC RTP19

Attaching Java program to
/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.JAR.
Attaching Java program to
/QIBM/ProdData/mqm/java/lib/jms.JAR.

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2026.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
Installation Verification Test

Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f94040403ccf041f0000c012
JMSTimestamp: 1020273404500
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:QPOZSPWT STANLEY 170302
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMS_IBM_PutTime:13441354
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$>
$
```

JNDI 없이 IBM MQ JMS 클라이언트 모드 테스트

다음 프로시저를 수행하십시오.

1. 다음 명령을 발행하여 큐 관리자가 시작되었는지와 큐 관리자의 상태가 활성인지 확인하십시오.

```
WRKMQM MQMNAME(QMGRNAME)
```

2. 다음 명령을 발행하여 서버 연결 채널이 작성되었는지 확인하십시오.

```
WRKMQMCHL CHLNAME( SYSTEM.DEF.SVRCONN ) CHLTYPE(*SVRCN)
MQMNAME(QMGRNAME)
```

3. **WRKMQMLSR** 명령을 발행하여 리스너가 올바른 포트에 대해 시작되었는지 확인하십시오.
4. **STRQSH** 명령을 발행하여 명령행에서 qshell을 시작하십시오.
5. **export** 명령을 발행하여 CLASSPATH가 정확한지 확인하십시오.

6. 다음과 같이 디렉토리를 변경하려면 **cd** 명령을 사용하십시오.

```
cd /qibm/proddata/mqm/java/bin
```

7. 스크립트 파일을 실행하십시오.

```
IVTRun -nojndi -client -m QMgrName -host hostname [-port port] [-channel channel]
```

다음 예제와 유사한 출력을 수신합니다. 저작권 설명은 사용 중인 제품의 버전에 따라 다릅니다.

```
> IVTRun -nojndi -client -m ELCRTP19 -host ELCRTP19 -port 1414 -channel SYSTEM.DEF.SVRCONN

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2026.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
Installation Verification Test
```

```
Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000d012
JMSTimestamp: 1020274009970
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:MQSeries Client for Java
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:28
JMS_IBM_MsgType:8
JMSXUserID:QMOM
JMS_IBM_PutTime:14085237
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$
```

JNDI로 IBM MQ JMS 테스트

다음 명령을 발행하여 큐 관리자가 시작되었는지와 큐 관리자의 상태가 활성화인지 확인하십시오.

```
WRKMQM MQMNAME(QMGRNAME)
```

IVTRun 샘플 테스트 스크립트 사용

다음 프로시저를 수행하십시오.

1. JMSAdmin.config 파일을 적절히 변경하십시오. 이 파일을 편집하려면 IBM i 명령줄에서 (파일 편집 **EDTF**) 명령을 사용하십시오

```
EDTF '/qibm/proddata/mqm/java/bin/JMSAdmin.config'
```

- a. Weblogic용 LDAP을 사용하려면 다음으로부터 주석을 제거하십시오.

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
```

- b. WebSphere Application Server용 LDAP을 사용하려면 다음으로부터 주석을 제거하십시오.

```
INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
```

- c. 파일 시스템을 테스트하려면 다음으로부터 주석을 제거하십시오.

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
```

- d. 적절한 행에서 주석을 제거하여 올바른 PROVIDER_URL을 선택했는지 확인하십시오.

- e. # 기호를 사용하여 다른 모든 행을 주석 처리하십시오.

- f. 모든 변경사항을 완료하고 나면 **F2=Save** 및 **F3=Exit**를 누르십시오.

2. **STRQSH** 명령을 발행하여 명령행에서 qshell을 시작하십시오.

3. **export** 명령을 발행하여 CLASSPATH가 정확한지 확인하십시오.

4. 다음과 같이 디렉토리를 변경하려면 **cd** 명령을 사용하십시오.

```
cd /qibm/proddata/mqm/java/bin
```

5. **IVTSetup** 명령을 발행하여 관리 대상 오브젝트(*MQQueueConnectionFactory* 및 *MQQueue*)를 작성하기 위해 **IVTSetup** 스크립트를 시작하십시오.

6. 다음 명령을 발행하여 IVTRun 스크립트를 실행하십시오.

```
IVTRun -url providerURL [-icf initCtxFact]
```

다음 예제와 유사한 출력을 수신합니다. 저작권 설명은 사용 중인 제품의 버전에 따라 다릅니다.

```
> IVTSetup
+ Creating script for object creation within JMSAdmin
+ Calling JMSAdmin in batch mode to create objects
Ignoring unknown flag: -i

5724-H72 (c) Copyright IBM Corp. 2011, 2026. All Rights Reserved.
Starting WebSphere MQ classes for Java Message Service Administration

InitCtx>
InitCtx>
InitCtx>
InitCtx>
InitCtx>
InitCtx>
Stopping MQSeries classes for Java Message Service Administration

+ Administration done; tidying up files
+ Done!
$
> IVTRun -url file:///tmp/mqjms -icf com.sun.jndi.fscontext.RefFSContextFactory

5724-H72 (c) Copyright IBM Corp. 2011, 2026. All Rights Reserved.
MQSeries classes for Java Message Service
Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
```

```

JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000e012
JMSTimestamp: 1020274903770
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMSXDeliveryCount:1
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMSXAppID:QPOZSPWT STANLEY 170308
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$

```

Maven 저장소를 사용한 Java 애플리케이션 개발

애플리케이션을 Java IBM MQ 개발할 때 Maven 저장소를 사용하여 종속성을 자동으로 설치하면, 인터페이스를 IBM MQ 사용하기 전에 명시적으로 아무것도 설치할 필요가 없습니다.

Maven Central 저장소

Maven은 애플리케이션 빌드를 위한 도구이며, 애플리케이션이 액세스해야 하는 아티팩트를 저장하는 저장소 또한 제공합니다.

Maven 저장소(또는 Central 저장소)에는 JAR 파일과 같은 파일이 고유한 버전을 갖도록 하는 구조가 있으며, 이러한 구조가 적용된 파일은 잘 알려진 이름 지정 메커니즘을 통해 쉽게 검색됩니다. 빌드 도구는 이러한 이름을 사용하여 애플리케이션의 종속 항목을 동적으로 가져올 수 있습니다. Maven을 빌드 도구로 사용할 때 애플리케이션의 정의 파일인 POM 파일에서 종속성을 명시하면, 빌드 프로세스가 이를 기반으로 수행할 작업을 파악합니다.

IBM MQ 클라이언트 파일

IBM MQ Java 클라이언트 인터페이스의 사본은 com.ibm.mq groupId 아래의 중앙 저장소에서 사용 가능합니다. 파일 ([Jakarta Messaging 3.0](#))과 [com.ibm.mq.jakarta.client.jar](#) 파일 (JMS 2.0 [com.ibm.mq.allclient.jar](#))을 찾을 수 있습니다. 이 파일들은 일반적으로 독립 실행형 프로그램에 사용됩니다. 또한 파일 [wmq.jakarta.jmsra.rar](#) (Jakarta Messaging 3.0)과 애플리케이션 Java EE 서버에서 사용하기 위한 파일 [wmq.jmsra.rar](#) (JMS 2.0)도 찾을 수 있습니다. 와 [jakarta.client.jar](#) 모두 [allclient.jar](#) 와 IBM MQ classes for JMS 를 IBM MQ classes for Java 포함한다.

중요사항: Apache Maven Assembly Plugin *jar-with-dependencies* 형식을 사용하여 IBM MQ 재배포 가능 JAR 파일을 포함하는 애플리케이션을 빌드하는 작업은 지원되지 않습니다.

maven 명령으로 처리된 pom.xml 파일에서, 다음 예에 표시된 대로 이러한 JAR 파일에 대한 종속 항목을 추가합니다.

- **JM 3.0** 애플리케이션 코드와 com.ibm.mq.jakarta.client.jar 간의 관계를 표시하려면 다음을 수행하십시오.

```

<dependency>
  <groupId>com.ibm.mq</groupId>
  <artifactId>com.ibm.mq.jakarta.client</artifactId>
  <version>10.0.0.0</version>
</dependency>

```

- **JMS 2.0** 애플리케이션 코드와 com.ibm.mq.allclient.jar 간의 관계를 표시하려면 다음을 수행하십시오.

```
<dependency>
  <groupId>com.ibm.mq</groupId>
  <artifactId>com.ibm.mq.allclient</artifactId>
  <version>10.0.0.0</version>
</dependency>
```

- **JM 3.0** Jakarta EE 자원 어댑터 사용을 위한 경우:

```
<dependency>
  <groupId>com.ibm.mq</groupId>
  <artifactId>wmq.jakarta.jmsra</artifactId>
  <version>10.0.0.0</version>
</dependency>
```

- **JMS 2.0** 리소스 JMS 2.0Java EE 어댑터 사용을 위해:

```
<dependency>
  <groupId>com.ibm.mq</groupId>
  <artifactId>wmq.jmsra</artifactId>
  <version>10.0.0.0</version>
</dependency>
```

Eclipse 에서 간단한 프로젝트를 JMS 실행하는 예제는 '[Maven으로 MQ 애플리케이션 Java 개발이 훨씬 쉬워졌습니다](#)' 문서를 IBM Developer 참조하십시오.

C++ 애플리케이션 개발

IBM MQ에서는 IBM MQ 오브젝트에 해당하는 C++ 클래스와 배열 데이터 유형에 해당하는 추가 클래스를 제공합니다. MQI를 통해 사용할 수 없는 여러 기능을 제공합니다.

IBM WebSphere MQ 7.0, IBM MQ 프로그래밍 인터페이스에 대한 개선사항은 C++ 클래스에 적용되지 않습니다.

IBM MQ C++에서는 다음 기능을 제공합니다.

- IBM MQ 데이터 구조의 자동 초기화.
- JIT(Just-In-Time) 큐 관리자 연결 및 큐 열기.
- 암시적 큐 닫기 및 큐 관리자 연결 끊기.
- 데드 레터 헤더 전송 및 수신.
- IMS 브릿지 헤더 전송 및 수신.
- 참조 메시지 헤더 전송 및 수신.
- 트리거 메시지 수신.
- CICS bridge 헤더 전송 및 수신.
- 작업 헤더 전송 및 수신.
- 클라이언트 채널 정의.

다음 Booch 클래스 다이어그램은 핸들 또는 데이터 구조가 있는 프로시저에 따른 MQI(예: C 사용)의 IBM MQ 엔티티와 모든 클래스가 병렬임을 보여줍니다. 모든 클래스는 ImqError 클래스([ImqError C++ 클래스 참조](#))에서 상속하므로, 오류 조건이 각 오브젝트와 연관될 수 있습니다.

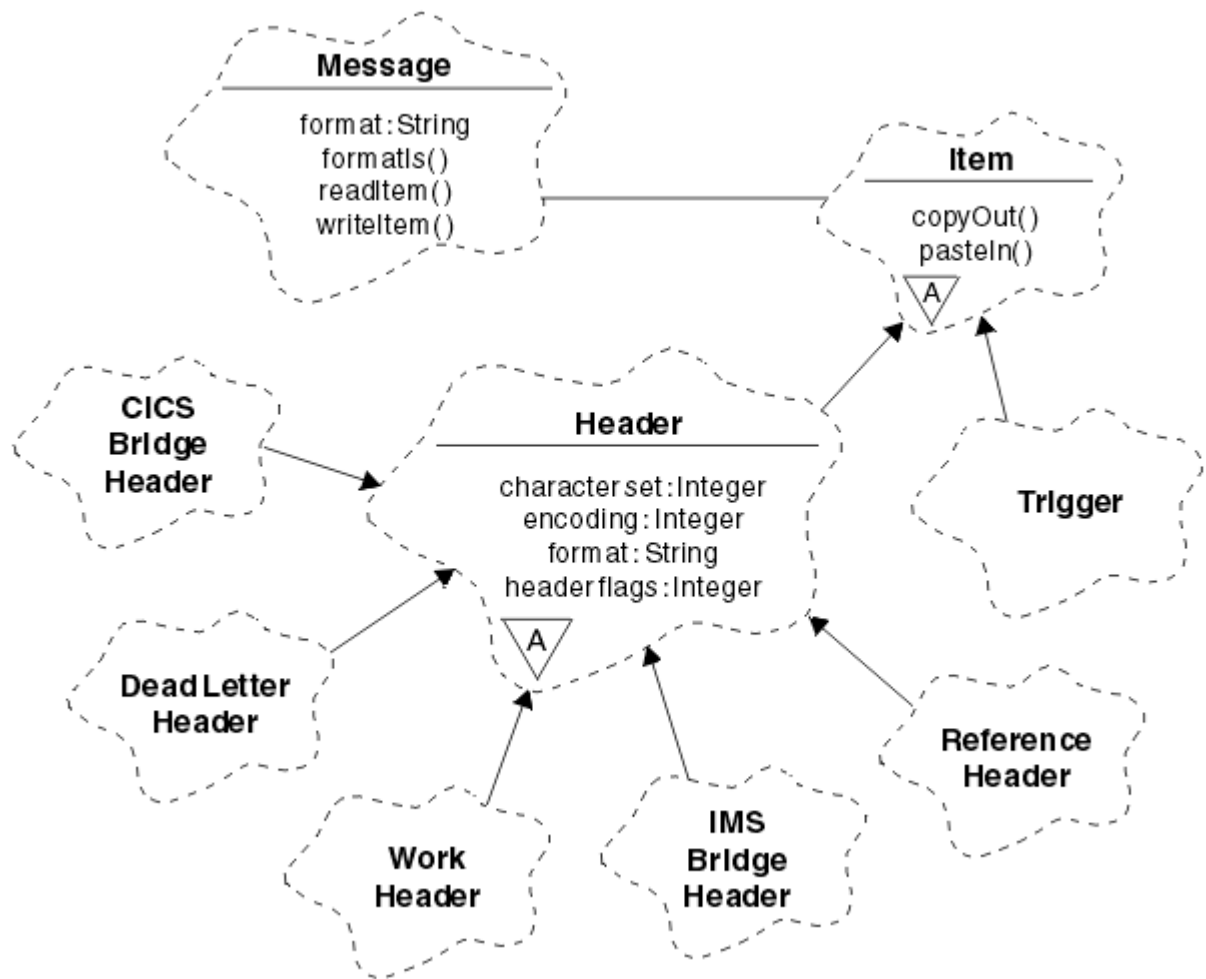


그림 65. IBM MQ C++ 클래스(항목 핸들링)

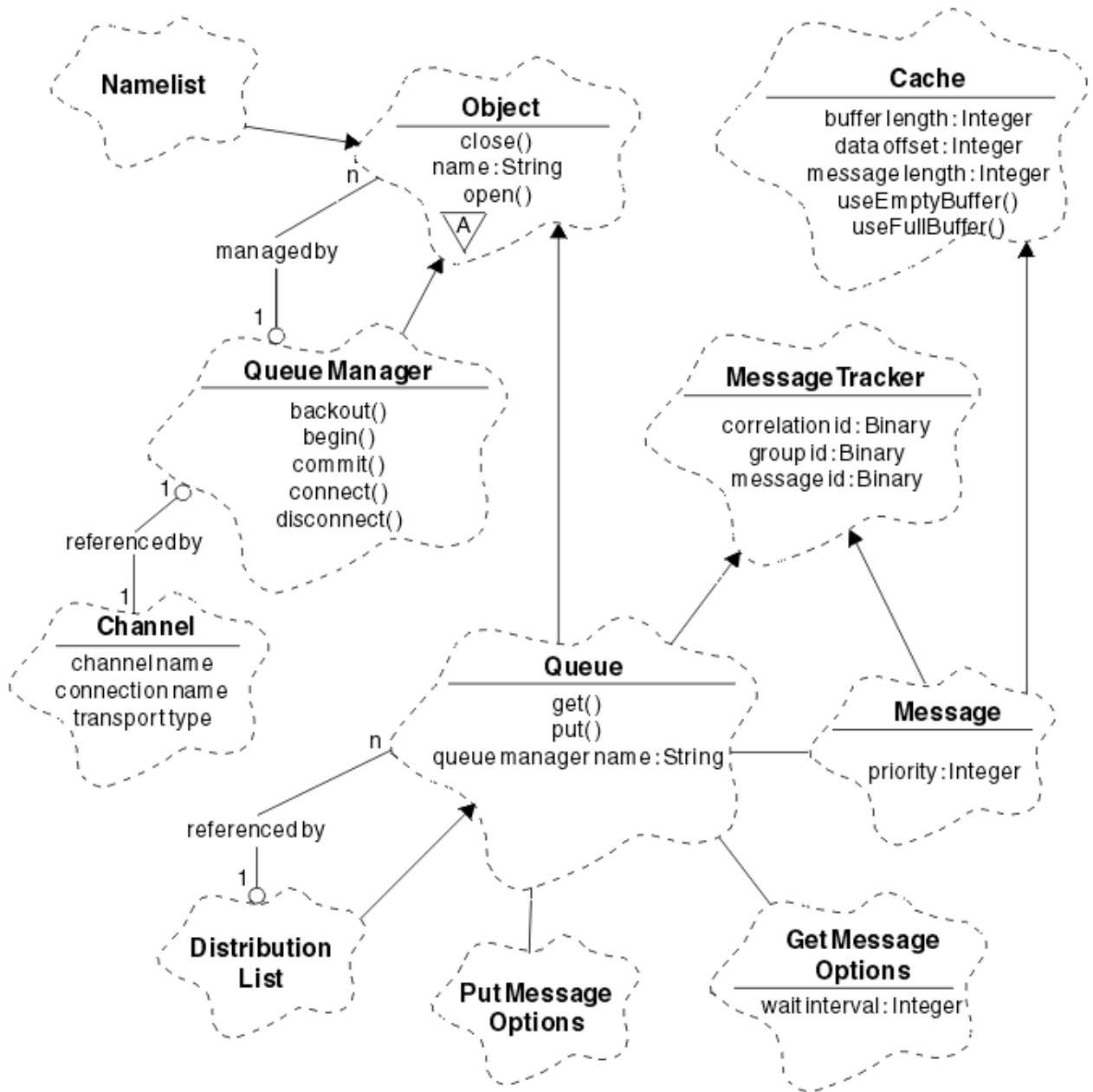


그림 66. IBM MQ C++ 클래스(큐 관리)

Booch 클래스 다이어그램을 올바르게 해석하려면 다음 규칙을 알아야 합니다.

- 메소드 및 중요한 속성은 클래스 이름 아래 나타냅니다.
- 클라우드 내 작은 삼각형은 추상 클래스를 나타냅니다.
- 상속은 상위 클래스로 이어진 화살표로 나타냅니다.
- 클라우드 사이에 있는 밑줄은 클래스 사이의 협업 관계를 나타냅니다.
- 숫자가 추가된 행은 두 클래스 사이의 참조 관계를 나타냅니다. 숫자는 한 번에 특정 관계에 참여할 수 있는 오브젝트 수를 나타냅니다.

다음 클래스와 데이터 유형은 큐 관리 클래스(496 페이지의 그림 66 참조) 및 항목 핸들링 클래스(495 페이지의 그림 65 참조)의 C++ 메소드 서명에서 사용됩니다.

- MQBYTE24와 같은 바이트 배열을 캡슐화하는 `ImqBinary` 클래스([ImqBinary C++ 클래스 참조](#)).
- **typedef** 부호 없는 `char ImqBoolean`으로 정의되는 `ImqBoolean` 데이터 유형.
- MQCHAR64와 같은 문자 배열을 캡슐화하는 `ImqString` 클래스([ImqString C++ 클래스 참조](#)).

데이터 구조를 포함하는 엔티티는 적절한 오브젝트 클래스에 포함됩니다. 개별 데이터 구조 필드(C++ 및 MQI 상호 참조 확인)는 메소드를 통해 액세스합니다.

핸들이 있는 엔티티는 ImqObject 클래스 계층 아래 있으며(ImqObject C++ 클래스 참조) 캡슐화된 인터페이스를 MQI를 제공합니다. 이러한 클래스의 오브젝트는 프로시저에 따른 MQI에 상대적으로 필요한 메소드 호출 수를 줄일 수 있는 지능적 동작은 금지합니다. 예를 들어 필요한 경우 큐 관리자 연결을 설정 및 제거하거나 적절한 옵션으로 큐를 연 다음 닫을 수 있습니다.

ImqMessage 클래스(ImqMessage C++ 클래스 참조)는 MQMD 데이터 구조를 캡슐화하고 캐시된 버퍼 기능을 제공하여 사용자 데이터와 항목(506 페이지의 『C++의 메시지 읽기』 참조)을 보유하는 위치의 역할도 합니다. 사용자 데이터에 대해 고정 길이 버퍼를 제공하고 버퍼를 여러 번 사용할 수 있습니다. 버퍼에 표시되는 데이터 크기는 한 번 사용에서 다음 사용으로 가면 달라질 수 있습니다. 또는 시스템에서 고정 길이의 버퍼를 제공 및 관리할 수 있습니다. 버퍼 크기(메시지 수신 시 사용 가능한 크기) 및 실제로 사용한 크기(실제로 수신한 바이트 수 또는 전송을 위한 바이트 수) 모두 중요한 고려사항입니다.

관련 개념

기술 개요

497 페이지의 『C++ 샘플 프로그램』

메시지 가져오기 및 넣기를 시연하기 위해 네 개의 샘플 프로그램이 제공됩니다.

501 페이지의 『C++ 언어 고려사항』

MQI (메시지 큐 인터페이스)를 사용하는 애플리케이션 프로그램을 작성할 때는 C++ 언어의 사용법과 관례에 관한 다음 사항들을 고려하십시오.

505 페이지의 『C++에서 메시지 데이터 준비』

시스템 또는 애플리케이션에서 제공할 수 있는 버퍼에서 메시지 데이터를 준비합니다. 각 메소드에는 장점이 있습니다. 버퍼를 사용하는 예가 제공됩니다.

5 페이지의 『IBM MQ용 애플리케이션 개발』

메시지를 송신하고 수신하며, 큐 관리자와 관련 자원을 관리하기 위한 애플리케이션을 개발할 수 있습니다. IBM MQ는 많은 다양한 언어와 프레임워크로 작성된 애플리케이션을 지원합니다.

관련 참조

511 페이지의 『IBM MQ C++ 프로그램 빌드』

IBM MQ 플랫폼에서 C++ 프로그램과 샘플을 컴파일, 링크 및 실행하는 데 사용할 명령과 함께 지원되는 컴파일러의 URL이 나열됩니다.

C++ 및 MQI 상호 참조

IBM MQ C++ 클래스

C++ 샘플 프로그램

메시지 가져오기 및 넣기를 시연하기 위해 네 개의 샘플 프로그램이 제공됩니다.

샘플 프로그램은 다음과 같습니다.

- HELLO WORLD (imqwld.cpp)
- SPUT (imqsput.cpp)
- SGET (imqsget.cpp)
- DPUT (imqdput.cpp)

샘플 프로그램은 497 페이지의 표 74에 표시된 디렉토리에 있습니다.

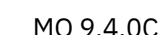
MQ_INSTALLATION_PATH(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

표 74. 샘플 프로그램의 위치		
환경	소스를 포함하는 디렉토리	빌드된 프로그램을 포함하는 프로그램
 AIX	MQ_INSTALLATION_PATH/샘플	MQ_INSTALLATION_PATH/샘플 /bin/ia (참고 1 참조)

표 74. 샘플 프로그램의 위치 (계속)

환경	소스를 포함하는 디렉토리	빌드된 프로그램을 포함하는 프로그램
 AIX	<code>MQ_INSTALLATION_PATH/샘플</code>	<code>MQ_INSTALLATION_PATH/샘플/bin/ca</code> (498 페이지의 『2』참고를 참조)
 IBM i	<code>/QIBM/ProdData/mqm/samp/</code>	(498 페이지의 『3』참고를 참조)
 Linux	<code>MQ_INSTALLATION_PATH/샘플</code>	없음
 Windows	<code>MQ_INSTALLATION_PATH\도구\C++\예제</code>	<code>MQ_INSTALLATION_PATH\도구\C++\samples\bin\vn</code> (498 페이지의 『4』참고를 참조)
 z/OS	<code>thlqual.SCSQCPPS</code>	

참고:

-  XL C/C++ for AIX 16 IBM 컴파일러로 빌드된 프로그램은 "ia" 폴더에 위치합니다.  이 컴파일러에 대한 지원은 (Long Term Support 릴리스의 경우, 릴리스의 **IBM MQ 9.3.5** **IBM MQ 9.4.0** Continuous Delivery 경우) 중단되었습니다.  XL C/C++ 의 AIX 16 컴파일러는 에서 제거되었습니다 IBM MQ 9.4.2.
-  Open XL C/C++ for AIX 17 IBM 컴파일러로 빌드된 프로그램은 "ca" 폴더에 위치합니다.
-  IBM i용 ILE C++ 컴파일러를 사용하여 빌드된 프로그램은 라이브러리 QMQM에 있습니다. 소스 파일은. `/QIBM/ProdData/mqm/samp`
-  를 사용하여 구축된 Microsoft Visual Studio 프로그램은 에 있습니다 `MQ_INSTALLATION_PATH\tools\cplusplus\samples\bin\vn`. 이러한 컴파일러에 대한 자세한 정보는 516 페이지의 『Windows에 C++ 프로그램 빌드』의 내용을 참조하십시오.

샘플 프로그램 HELLO WORLD(imqwrld.cpp)

이 C++ 샘플 프로그램은 `ImqMessage` 클래스를 사용하여 정규 데이터그램(C 구조)을 넣고 가져오는 방법을 보여줍니다.

이 프로그램은 `ImqMessage` 클래스를 사용하여 정규 데이터그램(C 구조)을 넣고 가져오는 방법을 보여줍니다. 이 샘플에서는 몇 가지 메소드 호출을 사용하여 `open`, `close` 및 `disconnect`와 같은 암시적 메소드 호출을 이용합니다.

Multiplatforms



IBM MQ에 대한 서버 연결을 사용하는 경우 다음 프로시저 중 하나를 따르십시오.

- 기존 기본 큐인 `SYSTEM.DEFAULT.LOCAL.QUEUE`를 사용하려면 매개변수를 전달하지 않고 **imqwrlds** 프로그램을 실행합니다.
- 동적으로 지정된 임시 큐를 사용하려면 **imqwrlds**를 실행하여 기본 모델 큐의 이름, `SYSTEM.DEFAULT.MODEL.QUEUE`를 전달합니다.

클라이언트 연결을 사용하여 에 접속하는 IBM MQ 경우, 다음 절차 중 하나를 완료하십시오:

- 환경 **MQSERVER** 변수를 설정하고 실행하거나 **imqwrldc**,
- 또는 **imqwrldc**를 실행하고 매개변수로 **queue-name**, **queue-manager-name** 및 **channel-definition**을 전달합니다. 여기서 일반적인 **channel-definition**은 **SYSTEM.DEF.SVRCONN/TCP/hostname (1414)**입니다.

z/OS의 경우



샘플 JCL **imqwrldc**를 사용하여 배치 작업을 구성하고 실행하십시오.

자세한 내용은 을 참조하십시오 [z/OS 배치](#), [RRS 배치](#) 및 [CICS](#).

샘플 코드

```
extern "C" {
#include <stdio.h>
}

#include <imqi.hpp> // IBM MQ C++

#define EXISTING_QUEUE "SYSTEM.DEFAULT.LOCAL.QUEUE"

#define BUFFER_SIZE 12

static char gpszHello[ BUFFER_SIZE ] = "Hello world" ;
int main ( int argc, char * * argv ) {
    ImqQueueManager manager ;
    int iReturnCode = 0 ;

    // Connect to the queue manager.
    if ( argc > 2 ) {
        manager.setName( argv[ 2 ] );
    }
    if ( manager.connect( ) ) {
        ImqQueue * pqueue = new ImqQueue ;
        ImqMessage * pmsg = new ImqMessage ;

        // Identify the queue which will hold the message.
        pqueue -> setConnectionReference( manager );
        if ( argc > 1 ) {
            pqueue -> setName( argv[ 1 ] );

            // The named queue can be a model queue, which will result in
            // the creation of a temporary dynamic queue, which will be
            // destroyed as soon as it is closed. Therefore we must ensure
            // that such a queue is not automatically closed and reopened.
            // We do this by setting open options which will avoid the need
            // for closure and reopening.
            pqueue -> setOpenOptions( MQOO_OUTPUT | MQOO_INPUT_SHARED |
                                     MQOO_INQUIRE );
        } else {
            pqueue -> setName( EXISTING_QUEUE );

            // The existing queue is not a model queue, and will not be
            // destroyed by automatic closure and reopening. Therefore we
            // will let the open options be selected on an as-needed basis.
            // The queue will be opened implicitly with an output option
            // during the "put", and then implicitly closed and reopened
            // with the addition of an input option during the "get".
        }

        // Prepare a message containing the text "Hello world".
        pmsg -> useFullBuffer( gpszHello , BUFFER_SIZE );
        pmsg -> setFormat( MQFMT_STRING );

        // Place the message on the queue, using default put message
        // Options.
        // The queue will be automatically opened with an output option.
        if ( pqueue -> put( * pmsg ) ) {
            ImqString strQueue( pqueue -> name( ) );

            // Discover the name of the queue manager.
            ImqString strQueueManagerName( manager.name( ) );
            printf( "The queue manager name is %s.\n",
```

```

        (char *)strQueueManagerName );

// Show the name of the queue.
printf( "Message sent to %s.\n", (char *)strQueue );

// Retrieve the data message just sent ("Hello world" expected)
// from the queue, using default get message options. The queue
// is automatically closed and reopened with an input option
// if it is not already open with an input option. We get the
// message just sent, rather than any other message on the
// queue, because the "put" will have set the ID of the message
// so, as we are using the same message object, the message ID
// acts as in the message object, a filter which says that we
// are interested in a message only if it has this
// particular ID.

if ( pqueue -> get( * pmsg ) ) {
    int iDataLength = pmsg -> dataLength( );

    // Show the text of the received message.
    printf( "Message of length %d received, ", iDataLength );

    if ( pmsg -> formatIs( MQFMT_STRING ) ) {
        char * pszText = pmsg -> bufferPointer( );

        // If the last character of data is a null, then we can
        // assume that the data can be interpreted as a text
        // string.
        if ( ! pszText[ iDataLength - 1 ] ) {
            printf( "text is \"%s\".\n", pszText );
        } else {
            printf( "no text.\n" );
        }
    }

    } else {
        printf( "non-text message.\n" );
    }
} else {
    printf( "ImqQueue::get failed with reason code %ld\n",
           pqueue -> reasonCode( ) );
    iReturnCode = (int)pqueue -> reasonCode( );
}

} else {
    printf( "ImqQueue::open/put failed with reason code %ld\n",
           pqueue -> reasonCode( ) );
    iReturnCode = (int)pqueue -> reasonCode( );
}

// Deletion of the queue will ensure that it is closed.
// If the queue is dynamic then it will also be destroyed.
delete pqueue ;
delete pmsg ;

} else {
    printf( "ImqQueueManager::connect failed with reason code %ld\n",
           manager.reasonCode( ) );
    iReturnCode = (int)manager.reasonCode( );
}

// Destruction of the queue manager ensures that it is
// disconnected. If the queue object were still available
// and open (which it is not), the queue would be closed
// prior to disconnection.

return iReturnCode ;
}

```

샘플 프로그램 SPUT (imqspout.cpp) 및 SGET (imqsget.cpp)

이러한 C++ 프로그램은 이름 지정된 큐에 메시지를 두고 해당 큐에서 메시지를 검색합니다.

이러한 샘플은 다음 클래스의 사용을 보여줍니다.

- ImqError([ImqError C++ 클래스 참조](#))
- ImqMessage([ImqMessage C++ 클래스 참조](#))
- ImqObject([ImqObject C++ 클래스 참조](#))

- ImqQueue(ImqQueue C++ 클래스 참조)
- ImqQueueManager(ImqQueueManager C++ 클래스 참조)

프로그램을 실행하려면 적절한 지시사항을 따르십시오.

Multiplatforms

Multi

1. `imqspu`ts *queue-name*을(를) 실행하십시오.
2. 콘솔에서 텍스트 행을 입력하십시오. 이러한 행은 메시지로 지정된 큐에 놓입니다.
3. 입력을 종료하려면 널 행을 입력하십시오.
4. 모든 라인을 가져와 `imqsget`s *queue-name* 콘솔에 표시합니다.

▶ **z/OS** 자세한 정보는 518 페이지의 『z/OS 배치, RRS 배치 및 CICS에서 C++ 프로그램 빌드』의 내용을 참조하십시오.

z/OS의 경우

z/OS

1. 샘플 JCL `imqspu`tr을 사용하여 배치 작업을 구성하고 실행하십시오. 메시지는 SYSIN 데이터 세트에서 읽습니다.
2. 샘플 JCL `imqsget`r을 사용하여 배치 작업을 구성하고 실행하십시오. 큐에서 메시지를 검색하여 SYSPRINT 데이터 세트에 송신합니다.

샘플 프로그램 DPUT(imqdput.cpp)

이 C++ 샘플 프로그램은 두 개의 큐로 구성된 분배 목록에 메시지를 넣습니다.

DPUT는 ImqDistributionList 클래스의 사용을 표시합니다(ImqDistributionList C++ 클래스 참조). 이 샘플은 z/OS에서 지원되지 않습니다.

1. 지정된 두 개의 큐에 메시지를 배치하기 위해 `imqdpu`ts *queue-name-1 queue-name-2* 실행하십시오.
2. 실행하고 `imqsget`s *queue-name-1* 해당 큐에서 메시지를 `imqsget`s *queue-name-2* 가져옵니다.

C++ 언어 고려사항

MQI (메시지 큐 인터페이스)를 사용하는 애플리케이션 프로그램을 작성할 때는 C++ 언어의 사용법과 관례에 관한 다음 사항들을 고려하십시오.

C++ 헤더 파일

헤더 파일은 C++ 언어로 IBM MQ 애플리케이션 프로그램을 작성하는 데 도움이 되도록 MQI 정의의 일부로 제공됩니다. 이러한 헤더 파일은 다음 표에 요약되어 있습니다.

표 75. C/C++ 헤더 파일	
파일 이름	컨텐츠
IMQI.HPP	C++ MQI 클래스(CMQC.H 및 IMQTYPE.H 포함)
IMQTYPE.H	ImqBoolean 데이터 유형 정의
CMQC.H	MQI 데이터 구조 및 Manifest 상수

애플리케이션의 이식성을 향상시키려면 `#include` 프리프로세서 지시문에서 소문자로 헤더 파일의 이름을 코딩하십시오.

```
#include <imqi.hpp> // C++ classes
```

C++ 메소드 및 속성

*const*인 메소드의 매개변수는 입력 전용입니다. 포인터(*) 또는 참조(&)를 포함하는 시그니처를 가진 매개변수는 참조로 전달됩니다. 포인터나 참조를 포함하지 않는 반환 값은 값으로 전달됩니다. 반환된 객체들은 호출자의 책임이 되는 새로운 개체들입니다.

일부 메소드 서명에는 지정되지 않은 경우 기본값을 사용하는 항목이 포함되어 있습니다. 이러한 항목은 항상 서명의 끝에 있으며 등호(=)를 사용하여 표시합니다. 등호 다음에 오는 값은 항목이 생략된 경우 적용되는 기본값을 나타냅니다.

이러한 클래스의 모든 메소드 이름은 대소문자를 혼용하며 소문자로 시작됩니다. 메소드 이름에 포함된 첫 단어를 제외한 각 단어는 대문자로 시작합니다. 의미가 광범위하게 이해되는 경우가 아니면 약어는 사용하지 않습니다. 사용되는 약어에는 *id* (정체성을 의미)와 *sync* (동기화를 의미)가 포함됩니다.

객체 속성은 *set* 및 *get* 메서드를 사용하여 접근합니다. *set* 메소드는 *set* 단어로 시작하며 *get* 메소드에는 접두부가 없습니다. 속성이 읽기 전용인 경우, 해당 속성에는 설정 메서드가 없습니다.

오브젝트 구성 중에 올바른 상태로 속성이 초기화되며, 오브젝트 상태는 항상 일관됩니다.

C++의 데이터 유형

모든 데이터 유형은 C *typedef*문으로 정의합니다.

ImqBoolean 유형은 *IMQTYPE.H*에 **unsigned char**로 정의되며 값은 TRUE 및 FALSE입니다. **MQBYTE** 배열 대신 **ImqBinary** 클래스 오브젝트를 사용하고 **char *** 대신 **ImqString** 클래스 오브젝트를 사용할 수 있습니다. 스토리 관리를 쉽게 수행하도록 많은 메소드에서 **char** 또는 **MQBYTE** 포인터 대신 오브젝트를 리턴합니다. 모든 반환 값은 호출자의 책임이 되며, 반환된 객체의 경우 *delete*를 사용하여 저장 공간을 해제할 수 있습니다.

C++의 2진 문자열 조작

2진 데이터 문자열은 **ImqBinary** 클래스의 오브젝트로 선언됩니다. C 연산자를 사용하여 이 클래스의 객체를 복사, 비교 및 설정할 수 있습니다. 다음 예제 코드는 이진 문자열에 대한 연산을 보여줍니다:

```
#include <imqi.hpp> // C++ classes

ImqMessage message ;
ImqBinary id, correlationId ;
MQBYTE24 byteId ;

correlationId.set( byteId, sizeof( byteId ) ); // Set.
id = message.id( );                          // Assign.
if ( correlationId == id ) {                  // Compare.
    ...
```

C++에서 문자열 조작

문자 데이터는 종종 클래스 **ImqString** 객체 형태로 반환되며, 변환 연산자를 사용하여 **char ***로 형변환할 수 있습니다. 이 **ImqString** 클래스는 문자열 처리를 돕는 메서드들을 포함합니다.

MQI의 C++ 메서드를 사용하여 문자 데이터를 수신하거나 반환할 때, 해당 문자 데이터는 항상 널 문자로 종료되며 길이는 임의로 설정할 수 있습니다. 그러나 정보의 생략을 초래할 수 있는 몇 가지 제한을 부과합니다 IBM MQ. 스토리지 관리를 용이하게 하도록 문자 데이터는 종종 **ImqString** 클래스 오브젝트로 리턴됩니다. 제공된 변환 연산자를 사용하여 이러한 객체를 **char ***로 형변환할 수 있으며, **char ***가 필요한 많은 상황에서 읽기 전용 목적으로 사용할 수 있습니다.

참고: **ImqString** 클래스 오브젝트의 **char *** 변환 결과는 널일 수 있습니다.

char *에 C 함수를 사용할 수 있지만, 해당 **ImqString** 클래스의 특수 메서드를 사용하는 것이 바람직합니다: **length()** 연산자는 **strlen**과 동등하며, **storage()**는 문자 데이터에 할당된 메모리를 나타냅니다.

C++에서 오브젝트의 초기 상태

모든 객체는 속성에 반영되는 일관된 초기 상태를 가진다. 초기 상태는 클래스 설명에 정의됩니다.

C++에서 C 사용

C++ 프로그램에서 C 함수를 사용할 때 적절한 헤더를 포함하십시오. 다음 예제 코드는 C++ 프로그램에 포함된 `string.h` 내용을 보여줍니다:

```
extern "C" {  
#include <string.h>  
}
```

C++ 표기 규칙

이 코드 예제는 메서드를 호출하고 매개변수를 선언하는 방법을 보여줍니다. 이 예제는 **ImqQueue::get** (**ImqMessage & msg**) 메서드와 매개변수 **ImqBoolean** 를 사용합니다. 다음과 같이 매개변수를 선언하고 사용합니다.

```
ImqQueueManager * pmanager ;    // Queue manager  
ImqQueue * pqueue ;            // Message queue  
ImqMessage msg ;               // Message  
char szBuffer[ 100 ] ;         // Buffer for message data  
  
pmanager = new ImqQueueManager ;  
pqueue = new ImqQueue ;  
pqueue -> setName( "myreplyq" );  
pqueue -> setConnectionReference( pmanager );  
  
msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );  
  
if ( pqueue -> get( msg ) ) {  
    long lDataLength = msg.dataLength( );  
  
    ...  
}
```

C++의 암시적 조작

성공적으로 메소드를 실행하기 위한 필수조건을 만족시키기 위해 여러 조작을 암시적으로 적시에 수행할 수 있습니다. 이러한 암시적 조작은 연결, 열기, 다시 열기, 닫기 및 연결 끊기입니다. 클래스 속성을 사용하여 연결 및 암시적 열기 동작을 제어할 수 있습니다.

연결

ImqQueueManager 오브젝트는 MQI에 대한 임의의 호출이 되는 임의의 메소드에 대해 자동으로 연결됩니다(C++ 및 MQI 교차 참조를 참조).

열기

MQGET, MQINQ, MQPUT 또는 MQSET을 호출하는 모든 메소드용으로 ImqObject 오브젝트가 자동으로 열립니다. **openFor** 메소드를 사용하여 하나 이상의 관련 **open option** 값을 지정하십시오.

다시 열기

MQGET, MQINQ, MQPUT 또는 MQSET 호출을 유발하는 메서드의 경우, 해당 객체가 이미 열려 있지만 기존 열기 옵션만으로는 MQI 호출이 성공적으로 수행될 수 없는 상황에서는 ImqObject 가 자동으로 다시 열립니다. 객체는 MQCO_NONE 임시 닫기 옵션 값을 사용하여 일시적으로 닫힙니다. **openFor** 메서드를 사용하여 관련 오픈 옵션을 추가하십시오.

다시 열면 특정 환경에서 다음과 같은 문제점이 발생할 수 있습니다.

- 임시 동적 큐는 닫힐 때 파괴되며, 다시 열 수 없습니다.
- 독점적 입력(명시적이든 기본값이든)을 위해 열린 큐는 폐쇄 및 재개 시의 기회 창 동안 다른 프로세스에 의해 접근될 수 있습니다.
- 큐를 닫으면 찾아보기 커서 위치가 유실됩니다. 이 상황은 닫기와 다시 열기를 방해하지 않지만, MQGMO_BROWSE_FIRST가 다시 사용될 때까지 커서의 후속 사용을 차단합니다.
- 큐가 닫힐 때 마지막으로 검색될 메시지의 컨텍스트가 손실됩니다.

이러한 상황이 발생하거나 예상되는 경우 오브젝트를 열기 전에(명시적 또는 암시적) 적절한 **open options** 를 명시적으로 설정하여 다시 열기를 방지하십시오.

복잡한 큐 처리 상황에서 **오픈** 옵션을 명시적으로 설정하면 성능이 향상되고 reopen 사용과 관련된 문제를 피할 수 있습니다.

닫기

오브젝트 상태가 더 이상 실행 가능하지 않으면, 예를 들어 ImqObject 연결 참조가 끊기거나 ImqObject 오브젝트가 영구 삭제된 경우 언제든지 ImqObject가 자동으로 닫힙니다.

연결 끊기

연결이 더 이상 실행 가능하지 않으면, 예를 들어 ImqObject 연결 참조가 끊겼거나 ImqQueueManager 오브젝트가 영구 삭제된 경우 언제든지 ImqQueueManager의 연결이 자동으로 끊깁니다.

C++의 2진 및 문자열

ImqString 클래스는 기존 *char ** 데이터 형식을 캡슐화합니다. ImqBinary 클래스는 2진 바이트 배열을 캡슐화합니다. 문자 데이터를 설정하는 일부 메소드에서 데이터를 자를 수 있습니다.

문자(**char ***) 데이터를 설정하는 메소드는 항상 데이터의 사본을 받지만 IBM MQ를 통해 일부 한계가 지정되므로 일부 메소드에서 사본을 자를 수도 있습니다.

ImqString 클래스([ImqString C++ 클래스 참조](#))는 기존의 **char ***를 캡슐화하며 다음과 같은 작업을 지원합니다:

- 비교
- 연결
- 복사
- 정수를 텍스트로 및 텍스트를 정수로 변환
- 토큰(단어) 추출
- 대문자 변환

ImqBinary 클래스([ImqBinary C++ 클래스 참조](#))는 임의 크기의 2진 바이트 배열을 캡슐화합니다. 특히, 이 클래스는 다음 속성을 보유하기 위해 사용됩니다:

- 회계 토큰(MQBYTE32)
- 연결 태그(MQBYTE128)
- 상관 ID(MQBYTE24)
- 기능 토큰(MQBYTE8)
- 그룹 ID(MQBYTE24)
- 인스턴스 ID(MQBYTE24)
- 메시지 ID(MQBYTE24)
- 메시지 토큰(MQBYTE16)
- 트랜잭션 인스턴스 ID(MQBYTE16)

이러한 속성은 다음 클래스의 객체에 속합니다:

- ImqCICSBridgeHeader([ImqCICSBridgeHeader C++ 클래스 참조](#))
- ImqGetMessageOptions([ImqGetMessageOptions C++ 클래스 참조](#))
- ImqIMSBridgeHeader([ImqIMSBridgeHeader C++ 클래스 참조](#))
- ImqMessageTracker([ImqMessageTracker C++ 클래스 참조](#))
- ImqQueueManager([ImqQueueManager C++ 클래스 참조](#))
- ImqReferenceHeader([ImqReferenceHeader C++ 클래스 참조](#))
- ImqWorkHeader([ImqWorkHeader C++ 클래스 참조](#))

ImqBinary 클래스는 비교와 복사 지원도 제공합니다.

C++에서 지원되지 않는 함수

C++ IBM MQ 클래스와 메서드는 플랫폼에 IBM MQ 독립적이므로 특정 플랫폼에서 지원되지 않는 일부 기능을 제공할 수 있습니다.

지원되지 않는 플랫폼에서 함수를 사용하려고 하면 IBM MQ에서는 함수를 감지하지만 C++ 언어 바인딩에서는 감지하지 못합니다. IBM MQ에서 다른 MQI 오류와 같은 오류를 프로그램에 보고합니다.

C++의 메시징

이 주제 콜렉션에서는 C++로 메시지를 준비하고 읽고 쓰는 방법을 자세히 설명합니다.

C++에서 메시지 데이터 준비

시스템 또는 애플리케이션에서 제공할 수 있는 버퍼에서 메시지 데이터를 준비합니다. 각 메소드에는 장점이 있습니다. 버퍼를 사용하는 예가 제공됩니다.

메시지를 보낼 때 `ImqCache` 오브젝트에서 관리하는 버퍼에 먼저 메시지 데이터를 준비합니다([ImqCache C++ 클래스 참조](#)). 버퍼가 상속을 통해 각 `ImqMessage` 오브젝트와 연관됩니다([ImqMessage C++ 클래스 참조](#)): 애플리케이션(**`useEmptyBuffer`** 또는 **`useFullBuffer`** 메소드 사용)을 사용하거나 시스템에서 자동으로 제공할 수 있습니다. 메시지 버퍼를 제공하는 애플리케이션의 이점은 애플리케이션에서 준비된 데이터 영역을 직접 사용할 수 있으므로 대부분의 경우 데이터를 복사하지 않아도 된다는 것입니다. 단점은 제공된 버퍼의 길이가 고정된다는 것입니다.

버퍼는 재사용할 수 있고 전송하기 전에 **`setMessageLength`** 메소드를 사용하여 매번 전송되는 바이트 수를 변경할 수 있습니다.

시스템에서 자동으로 제공한 경우 시스템에서 사용 가능한 바이트 수를 관리하고 `ImqCache write` 메소드나 `ImqMessage writeItem` 메소드 등을 사용하여 메시지에 데이터를 복사할 수 있습니다. 메시지 버퍼는 필요에 따라 증가합니다. 버퍼가 증가하므로 이전에 작성된 데이터가 유실되지 않습니다. 대형 또는 다중 파트 메시지는 순차적으로 작성할 수 있습니다.

다음 예에서는 간소화된 메시지 송신을 보여줍니다.

1. 사용자 제공 버퍼의 준비된 데이터 사용

```
char szBuffer[ ] = "Hello world" ;  
  
msg.useFullBuffer( szBuffer, sizeof( szBuffer ) );  
msg.setFormat( MQFMT_STRING );
```

2. 버퍼 크기가 데이터 크기를 초과하는 사용자 제공 버퍼의 준비된 데이터 사용

```
char szBuffer[ 24 ] = "Hello world" ;  
  
msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );  
msg.setFormat( MQFMT_STRING );  
msg.setMessageLength( 12 );
```

3. 사용자 제공 버퍼에 데이터를 복사합니다.

```
char szBuffer[ 12 ];  
  
msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );  
msg.setFormat( MQFMT_STRING );  
msg.write( 12, "Hello world" );
```

4. 시스템 제공 버퍼에 데이터를 복사합니다.

```
msg.setFormat( MQFMT_STRING );  
msg.write( 12, "Hello world" );
```

5. 오브젝트를 사용하여 시스템 제공 버퍼에 데이터 복사(오브젝트가 콘텐츠뿐 아니라 메시지 형식도 설정)

```
ImqString strText( "Hello world" );  
  
msg.writeItem( strText );
```

C++의 메시지 읽기

버퍼는 애플리케이션이나 시스템에서 제공할 수 있습니다. 데이터는 버퍼에서 직접 액세스하거나 순차적으로 읽을 수 있습니다. 각 메시지 유형에 해당하는 클래스가 있습니다. 샘플 코드가 제공됩니다.

데이터를 수신할 때 애플리케이션이나 시스템이 적절한 메시지 버퍼를 제공할 수 있습니다. 특정 `ImqMessage` 오브젝트의 다중 전송 및 다중 수신에 동일한 버퍼를 사용할 수 있습니다. 메시지 버퍼가 자동으로 제공되는 경우 수신되는 모든 데이터 길이를 수용하도록 버퍼가 증가합니다. 그러나 애플리케이션에서 제공한 메시지 버퍼가 수신된 데이터를 보유하는 데 충분히 크지 않을 수 있습니다. 그러면 메시지 수신에 사용한 옵션에 따라 자르거나 실패가 발생할 수 있습니다.

수신되는 데이터는 메시지 버퍼에서 직접 액세스할 수 있으며, 데이터 길이가 수신되는 데이터의 총 크기를 나타냅니다. 또는 메시지 버퍼에서 수신되는 데이터를 순차적으로 읽을 수 있습니다. 이 경우 데이터 포인터가 수신되는 데이터의 다음 바이트를 주소로 지정하며 데이터를 읽을 때마다 데이터 포인터와 데이터 길이가 업데이트됩니다.

항목은 모두 순차적이고 개별적으로 처리해야 하는 메시지 버퍼의 사용자 영역에 있는 메시지입니다. 일반 사용자 데이터와 별개로 항목은 데드 레터 헤더 또는 트리거 메시지일 수 있습니다. 항목은 항상 메시지 형식과 연관됩니다. 메시지 형식은 항목과 연관되지 않을 수도 있습니다.

인식 가능한 IBM MQ 메시지 형식에 해당하는 각 항목의 오브젝트 클래스가 있습니다. 데드 레터 헤더와 트리거 메시지용으로 각각 하나씩 있습니다. 사용자 데이터용 오브젝트 클래스는 없습니다. 즉, 인식 가능한 형식이 소진되면 애플리케이션에서 나머지를 처리해야 합니다. 사용자 데이터의 클래스는 `ImqItem` 클래스를 특수화하여 작성할 수 있습니다.

다음 예에서는 가상의 상황에서 사용자 데이터에 선행할 수 있는 여러 잠재적 항목을 고려하는 메시지 수신을 표시합니다. 비항목 사용자 데이터는 식별할 수 있는 항목 다음에 발생하는 사항으로 정의됩니다. 자동 버퍼(기본 값)를 사용하여 임의의 크기의 메시지 데이터를 보유합니다.

```
ImqQueue queue ;
ImqMessage msg ;

if ( queue.get( msg ) ) {

    /* Process all items of data in the message buffer. */
    do while ( msg.dataLength( ) ) {
        ImqBoolean bFormatKnown = FALSE ;
        /* There remains unprocessed data in the message buffer. */

        /* Determine what kind of item is next. */

        if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
            ImqDeadLetterHeader header ;
            /* The next item is a dead-letter header. */
            /* For the next statement to work and return TRUE, */
            /* the correct class of object pointer must be supplied. */
            bFormatKnown = TRUE ;

            if ( msg.readItem( header ) ) {
                /* The dead-letter header has been extricated from the */
                /* buffer and transformed into a dead-letter object. */
                /* The encoding and character set of the dead-letter */
                /* object itself are MQENC_NATIVE and MQCCSI_Q_MGR. */
                /* The encoding and character set from the dead-letter */
                /* header have been copied to the message attributes */
                /* to reflect any remaining data in the buffer. */

                /* Process the information in the dead-letter object. */
                /* Note that the encoding and character set have */
                /* already been processed. */
                ...
            }
            /* There might be another item after this, */
            /* or just the user data. */
        }
        if ( msg.formatIs( MQFMT_TRIGGER ) ) {
            ImqTrigger trigger ;
            /* The next item is a trigger message. */
            /* For the next statement to work and return TRUE, */
            /* the correct class of object pointer must be supplied. */
            bFormatKnown = TRUE ;
            if ( msg.readItem( trigger ) ) {

                /* The trigger message has been extricated from the */
```

```

        /* buffer and transformed into a trigger object. */
        /* Process the information in the trigger object. */
        ...
    }

    /* There is usually nothing after a trigger message. */
}

if ( msg.formatIs( FMT_USERCLASS ) ) {
    UClass object ;
    /* The next item is an item of a user-defined class. */
    /* For the next statement to work and return TRUE, */
    /* the correct class of object pointer must be supplied. */
    bFormatKnown = TRUE ;

    if ( msg.readItem( object ) ) {
        /* The user-defined data has been extricated from the */
        /* buffer and transformed into a user-defined object. */

        /* Process the information in the user-defined object. */
        ...
    }

    /* Continue looking for further items. */
}
if ( ! bFormatKnown ) {
    /* There remains data that is not associated with a specific */
    /* item class. */
    char * pszDataPointer = msg.dataPointer( ); /* Address. */
    int iDataLength = msg.dataLength( ); /* Length. */

    /* The encoding and character set for the remaining data are */
    /* reflected in the attributes of the message object, even */
    /* if a dead-letter header was present. */
    ...
}
}
}

```

이 예에서 FMT_USERCLASS는 UClass 클래스의 오브젝트와 연관된 8자 형식 이름을 나타내는 상수이며 애플리케이션에서 정의합니다.

UClass는 ImqItem 클래스에서 파생되며([ImqItem C++ 클래스 참조](#)) 이 클래스에서 가상 **copyOut**과 **pasteIn** 메소드를 구현합니다.

다음 두 예에서는 ImqDeadLetterHeader 클래스([ImqDeadLetterHeader C++ 클래스 참조](#))의 코드를 보여줍니다. 첫 번째 예는 사용자 정의 캡슐화된 메시지 쓰기 코드를 표시합니다.

```

// Insert a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: copyOut ( ImqMessage & msg ) {
    ImqBoolean bSuccess ;
    if ( msg.moreBytes( sizeof( omqdlh ) ) ) {
        ImqCache cacheData( msg ); // Preserve original message content.
        // Note original message attributes in the dead-letter header.
        setEncoding( msg.encoding( ) );
        setCharacterSet( msg.characterSet( ) );
        setFormat( msg.format( ) );

        // Set the message attributes to reflect the dead-letter header.
        msg.setEncoding( MQENC_NATIVE );
        msg.setCharacterSet( MQCCSI_Q_MGR );
        msg.setFormat( MQFMT_DEAD_LETTER_HEADER );
        // Replace the existing data with the dead-letter header.
        msg.clearMessage();
        if ( msg.write( sizeof( omqdlh ), (char *) & omqdlh ) ) {
            // Append the original message data.
            bSuccess = msg.write( cacheData.messageLength( ),
                                cacheData.bufferPointer( ) );
        } else {
            bSuccess = FALSE ;
        }
    } else {
        bSuccess = FALSE ;
    }
    // Reflect and cache error in this object.
    if ( ! bSuccess ) {

```

```

        setReasonCode( msg.reasonCode( ) );
        setCompletionCode( msg.completionCode( ) );
    }

    return bSuccess ;
}

```

두 번째 예는 사용자 정의 캡슐화된 메시지 읽기 코드를 표시합니다.

```

// Read a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: pasteIn ( ImqMessage & msg ) {
    ImqBoolean bSuccess = FALSE ;

    // First check that the eye-catcher is correct.
    // This is also our guarantee that the "character set" is correct.
    if ( ImqItem::structureIdIs( MQDLH_STRUC_ID, msg ) ) {
        // Next check that the "encoding" is correct, as the MQDLH
        // contains numeric data.
        if ( msg.encoding( ) == MQENC_NATIVE ) {

            // Finally check that the "format" is correct.
            if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
                char * pszBuffer = (char *) &omqdlh ;
                // Transfer the MQDLH from the message and move pointer on.
                if ( bSuccess = msg.read( sizeof( omdlh ), pszBuffer ) ) {
                    // Update the encoding, character set and format of the
                    // message to reflect the remaining data.
                    msg.setEncoding( encoding( ) );
                    msg.setCharacterSet( characterSet( ) );
                    msg.setFormat( format( ) );
                } else {

                    // Reflect the cache error in this object.
                    setReasonCode( msg.reasonCode( ) );
                    setCompletionCode( msg.completionCode( ) );
                }
            } else {
                setReasonCode( MQRC_INCONSISTENT_FORMAT );
                setCompletionCode( MQCC_FAILED );
            }
        } else {
            setReasonCode( MQRC_ENCODING_ERROR );
            setCompletionCode( MQCC_FAILED );
        }
    } else {
        setReasonCode( MQRC_STRUC_ID_ERROR );
        setCompletionCode( MQCC_FAILED );
    }

    return bSuccess ;
}

```

자동 버퍼에서 버퍼 스토리지는 휘발성입니다. 즉, 각 **get** 메소드 호출 후에 다른 물리적 위치에 버퍼 데이터가 보유될 수 있습니다. 따라서 버퍼 데이터를 참조할 때마다 **bufferPointer** 또는 **dataPointer** 메소드를 사용하여 메시지 데이터에 액세스하십시오.

프로그램에서 메시지 데이터를 수신하기 위한 고정 영역을 따로 확보하게 할 수 있습니다. 이 경우 **get** 메소드를 사용하기 전에 **useEmptyBuffer** 메소드를 호출하십시오.

자동화되지 않은 고정 영역을 사용하면 메시지를 최대 크기로 제한하므로, **ImqGetMessageOptions** 오브젝트의 **MQGMO_ACCEPT_TRUNCATED_MSG** 옵션을 고려하는 것이 중요합니다. 이 옵션이 지정되지 않으면(기본값) **MQRC_TRUNCATED_MSG_FAILED** 이유 코드가 표시될 수 있습니다. 이 옵션이 지정된 경우 애플리케이션 디자인에 따라 **MQRC_TRUNCATED_MSG_ACCEPTED** 이유 코드가 표시될 수 있습니다.

다음 예는 스토리지의 고정 영역을 사용하여 메시지를 수신하는 방법을 보여줍니다.

```

char * pszBuffer = new char[ 100 ];

msg.useEmptyBuffer( pszBuffer, 100 );
gmo.setOptions( MQGMO_ACCEPT_TRUNCATED_MSG );
queue.get( msg, gmo );

delete [ ] pszBuffer ;

```


이 코드 단편에서는 **bufferPointer** 메소드를 사용하지 않고 항상 *pszBuffer*로 버퍼의 주소를 지정할 수 있습니다. 그러나 범용 액세스를 위해 **dataPointer** 메소드를 사용하는 것이 좋습니다. 애플리케이션(ImqCache 클래스 오브젝트가 아님)이 사용자 정의(비자동) 버퍼를 제거해야 합니다.

주의: **useEmptyBuffer**로 널 포인터와 길이 0을 지정하면 길이가 0인 고정 길이 버퍼가 지정되지 않습니다. 이 결합은 이전에 사용자 정의된 버퍼를 무시하고 자동 버퍼를 사용하도록 되돌리는 요청으로 해석됩니다.

C++에서 메시지 작성하기

다음 상황에 대한 코드 예제를 확인하세요: - 데드레터 큐(dead-letter queue)에 메시지 작성 - 큐브릿지(IMS)에 메시지 작성 - CICS bridge에 메시지 작성 - 워크로드 관리자(Workload z/OS Manager)가 관리하는 큐에 메시지 작성

C++로 데드-레터 큐에 메시지 작성

다중 부분 메시지의 전형적인 사례는 데드 레터 헤더를 포함하는 경우이다. 처리할 수 없는 메시지의 데이터가 데드-레터 헤더에 추가됩니다.

```
ImqQueueManager mgr ;           // The queue manager.
ImqQueue queueIn ;              // Incoming message queue.
ImqQueue queueDead ;           // Dead-letter message queue.
ImqMessage msg ;                // Incoming and outgoing message.
ImqDeadLetterHeader header ;    // Dead-letter header information.

// Retrieve the message to be rerouted.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the dead-letter header information.
header.setDestinationQueueManagerName( mgr.name( ) );
header.setDestinationQueueName( queueIn.name( ) );
header.setPutApplicationName( /* ? */ );
header.setPutApplicationType( /* ? */ );
header.setPutDate( /* TODAY */ );
header.setPutTime( /* NOW */ );
header.setDeadLetterReasonCode( FB_APPL_ERROR_1234 );

// Insert the dead-letter header information. This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the dead-letter queue.
queueDead.setConnectionReference( mgr );
queueDead.setName( mgr.deadLetterQueueName( ) );
queueDead.put( msg );
```

C++로 IMS 브릿지에 메시지 작성

- IMSIBM MQ 브리지로 전송되는 메시지는 특수 헤더를 사용할 수 있습니다. IMS 헤더는 정규 메시지 데이터 앞에 접두부로 지정됩니다.

```
ImqQueueManager mgr;           // The queue manager.
ImqQueue queueBridge;          // IMS bridge message queue.
ImqMessage msg;                // Outgoing message.
ImqIMSBridgeHeader header;     // IMS bridge header.

// Set up the message.
//
// Here we are constructing a message with format
// MQFMT_IMS_VAR_STRING, and appropriate data.
//
msg.write( 2, /* ? */ );        // Total message length.
msg.write( 2, /* ? */ );        // IMS flags.
msg.write( 7, /* ? */ );        // Transaction code.
msg.write( /* ? */ , /* ? */ ); // String data.
msg.setFormat( MQFMT_IMS_VAR_STRING ); // The format attribute.

// Set up the IMS bridge header information.
//
// The reply-to-format is often specified.
```

```
// Other attributes can be specified, but all have default values.
//
header.setReplyToFormat( /* ? */ );

// Insert the IMS bridge header into the message.
//
// This will:
// 1) Insert the header into the message buffer, before the existing
//    data.
// 2) Copy attributes out of the message descriptor into the header,
//    for example the IMS bridge header format attribute will now
//    be set to MQFMT_IMS_VAR_STRING.
// 3) Set up the message attributes to describe the header, in
//    particular setting the message format to MQFMT_IMS.
//
msg.writeItem( header );

// Send the message to the IMS bridge queue.
//
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

C++로 CICS bridge에 메시지 작성

를 사용하여 IBM MQ for z/OS 로 전송되는 CICS bridge 메시지에는 특별한 헤더가 필요합니다. CICS bridge 헤더는 정규 메시지 데이터 앞에 접두부로 지정됩니다.

```
ImqQueueManager mgr ;           // The queue manager.
ImqQueue queueIn ;             // Incoming message queue.
ImqQueue queueBridge ;         // CICS bridge message queue.
ImqMessage msg ;               // Incoming and outgoing message.
ImqCicsBridgeHeader header ;    // CICS bridge header information.

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the CICS bridge header information.
// The reply-to format is often specified.
// Other attributes can be specified, but all have default values.
header.setReplyToFormat( /* ? */ );

// Insert the CICS bridge header information. This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the CICS bridge queue.
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

C++로 작업 헤더가 있는 메시지 작성

특정 z/OS 워크로드 관리자가 IBM MQ for z/OS 관리하는 큐로 전송되는 메시지에는 특별한 헤더가 필요합니다. 일반 메시지 데이터의 접두부로 작업 헤더가 지정됩니다.

```
ImqQueueManager mgr ;           // The queue manager.
ImqQueue queueIn ;             // Incoming message queue.
ImqQueue queueWLM ;           // WLM managed queue.
ImqMessage msg ;               // Incoming and outgoing message.
ImqWorkHeader header ;         // Work header information

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Insert the Work header information. This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the WLM managed queue.
```

```
queueWLM.setConnectionReference( mgr );
queueWLM.setName( /* ? */ );
queueWLM.put( msg );
```

IBM MQ C++ 프로그램 빌드

IBM MQ 플랫폼에서 C++ 프로그램과 샘플을 컴파일, 링크 및 실행하는 데 사용할 명령과 함께 지원되는 컴파일러의 URL이 나열됩니다.

IBM MQ의 지원되는 각 플랫폼 및 버전에 대한 컴파일러 목록은 [시스템 요구 사항 IBM MQ](#)의 내용을 참조하십시오.

IBM MQ C++ 프로그램을 컴파일하고 링크하는 데 필요한 명령은 설치와 요구사항에 따라 다릅니다. 다음 예에서는 여러 플랫폼에서 기본 IBM MQ 설치를 사용하는 컴파일러의 일반 컴파일 및 링크 명령을 보여줍니다.

AIX에 C++ 프로그램 빌드

XL C/C++IBM 를 AIX 사용하여 C++ AIX IBM MQ 프로그램을 컴파일합니다.

 **Deprecated** XL C/C++ 의 AIX 16 컴파일러 지원은 (IBM MQ 9.4.0Long Term Support 릴리스에서는, 그리고 IBM MQ 9.3.5Continuous Delivery 릴리스에서는) 더 이상 사용되지 않습니다. 이 컴파일러는 17 AIX 컴파일러용 Open XL C/C++IBM 로 대체되었습니다. 이러한 컴파일러 버전 간 컴파일러 옵션의 서로 다른 매핑에 대한 정보는 Open XL C/C++ 의 [옵션 매핑](#) 문서를 AIX 참조하십시오.

 **V 9.4.2**  **Removed** XL C/C++ 는 16 AIX 컴파일러용으로 제공되던 기능이 AIX 에서 제거되었습니다 IBM MQ 9.4.2.

클라이언트

`MQ_INSTALLATION_PATH`은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

이 명령어 예시는 IBMOpen XL C/C++ 를 사용하여 17 AIX 컴파일러를 대상으로 합니다:

32비트 스레드되지 않은 애플리케이션

```
ibm-clang++_r -o imqsputc_32 imqsputc.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca -limqb23ca -lmqic
```

32비트 스레드 애플리케이션

```
ibm-clang++_r -o imqsputc_32_r imqsputc.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca_r -limqb23ca_r -lmqic_r
```

64비트 스레드되지 않은 애플리케이션

```
ibm-clang++_r -m64 -o imqsputc_64 imqsputc.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca -limqb23ca -lmqic
```

64비트 스레드 애플리케이션

```
ibm-clang++_r -m64 -o imqsputc_64_r imqsputc.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca_r -limqb23ca_r -lmqic_r
```

이 명령어 예제는 16 AIX 컴파일러용 IBMXL C/C++ 를 사용합니다:

32비트 스레드되지 않은 애플리케이션

```
xlC -o imqsputc_32 imqsputc.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

32비트 스레드 애플리케이션

```
xlC_r -o imqsputc_32_r imqsputc.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

64비트 스레드되지 않은 애플리케이션

```
xlC -q64 -o imqsputc_64 imqsputc.cpp -I MQ_INSTALLATION_PATH/inc  
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

64비트 스레드 애플리케이션

```
xlC_r -q64 -o imqsputc_64_r imqsputc.cpp -I MQ_INSTALLATION_PATH/inc  
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

서버

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

이 명령어 예시는 IBMOpen XL C/C++ 를 사용하여 17 AIX 컴파일러를 대상으로 합니다:

32비트 스레드되지 않은 애플리케이션

```
ibm-clang++_r -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc  
-L MQ_INSTALLATION_PATH/lib -limqs23ca -limqb23ca -lmqm
```

32비트 스레드 애플리케이션

```
ibm-clang++_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc  
-L MQ_INSTALLATION_PATH/lib -limqs23ca_r -limqb23ca_r -lmqm_r
```

64비트 스레드되지 않은 애플리케이션

```
ibm-clang++_r -m64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc  
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca -limqb23ca -lmqm
```

64비트 스레드 애플리케이션

```
ibm-clang++_r -m64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc  
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca_r -limqb23ca_r -lmqm_r
```

이 명령어 예제는 16 AIX 컴파일러용 IBMXL C/C++ 를 사용합니다:

32비트 스레드되지 않은 애플리케이션

```
xlC -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc  
-L MQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

32비트 스레드 애플리케이션

```
xlC_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc  
-L MQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

64비트 스레드되지 않은 애플리케이션

```
xlC -q64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc  
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

64비트 스레드 애플리케이션

```
xlC_r -q64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc  
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

IBM i IBM i에 C++ 프로그램 빌드

ILE C++ 컴파일러를 사용하여 'IBM i'에서 'IBM MQ C++ 프로그램을 빌드합니다.

IBM ILE C++ for IBM i는 C++ 프로그램을 위한 네이티브 컴파일러입니다. 다음 지침은 이 컴파일러를 사용하여 '안녕하세요.'를 사용하여 'IBM MQ' C++ 애플리케이션을 생성하는 방법을 설명합니다 예로서의 IBM MQ 샘플 프로그램.

1. '먼저 읽어보세요.'의 지침에 따라 'IBM i' 컴파일러용 ILE C++를 설치합니다 제품을 수반하는 매뉴얼.

2. QCXXN 라이브러리가 라이브러리 목록에 있는지 확인하십시오.
3. HELLO WORLD 샘플 프로그램을 작성합니다.
 - a. 모듈을 작성합니다.

```
CRTCPPMOD MODULE(MYLIB/IMQWRLD) +
SRCSTMF('/QIBM/ProdData/mqm/samp/imqwrlld.cpp') +
INCDIR('/QIBM/ProdData/mqm/inc') DFTCHAR(*SIGNED) +
TERASPACE(*YES)
```

C++ 샘플 프로그램의 소스는 /QIBM/ProdData/mqm/samp에 있으며 포함 파일은 /QIBM/ProdData/mqm/inc에 있습니다.

또는 SRCFILE(QCPPSRC/LIB) SRCMBR(IMQWRLD) 라이브러리에 소스가 있습니다.

- b. 다음과 같이 IBM MQ 제공 서비스 프로그램과 바인딩하여 프로그램 오브젝트를 생성하십시오.

```
CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
BNDSRVPGM(QMQM/IMQB23I4 QMQM/IMQS23I4)
```

스레드 애플리케이션을 빌드하려면 다음과 같이 재입력 서비스 프로그램을 사용하십시오.

```
CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
BNDSRVPGM(QMQM/IMQB23I4[_R] QMQM/IMQS23I4[_R])
```

- c. 다음과 같이 SYSTEM.DEFAULT.LOCAL.QUEUE를 사용하여 HELLO WORLD 샘플 프로그램을 실행하십시오.

```
CALL PGM(MYLIB/IMQWRLD)
```

Linux에 C++ 프로그램 빌드

GNU g++ 컴파일러를 사용하여 'Linux'에서 'IBM MQ' C++ 프로그램을 빌드합니다.

System p

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

클라이언트: System p

32비트 스레드되지 않은 애플리케이션

```
g++ -m32 -o imqsputc_32 imqspcut.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

32비트 스레드 애플리케이션

```
g++ -m32 -o imqsputc_r32 imqspcut.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r
```

64비트 스레드되지 않은 애플리케이션

```
g++ -m64 -o imqsputc_64 imqspcut.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

64비트 스레드 애플리케이션

```
g++ -m64 -o imqsputc_r64 imqspcut.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r
```

서버: System p

32비트 스레드되지 않은 애플리케이션

```
g++ -m32 -o imqspu32 imqspu32.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

32비트 스레드 애플리케이션

```
g++ -m32 -o imqspu32_r imqspu32.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r
```

64비트 스레드되지 않은 애플리케이션

```
g++ -m64 -o imqspu64 imqspu64.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

64비트 스레드 애플리케이션

```
g++ -m64 -o imqspu64_r imqspu64.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r
```

IBM Z

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

클라이언트: IBM Z

32비트 스레드되지 않은 애플리케이션

```
g++ -m31 -fsigned-char -o imqspuc32 imqspuc32.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

32비트 스레드 애플리케이션

```
g++ -m31 -fsigned-char -o imqspuc32_r imqspuc32.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r
-lpthread
```

64비트 스레드되지 않은 애플리케이션

```
g++ -m64 -fsigned-char -o imqspuc64 imqspuc64.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

64비트 스레드 애플리케이션

```
g++ -m64 -fsigned-char -o imqspuc64_r imqspuc64.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

서버: IBM Z

32비트 스레드되지 않은 애플리케이션

```
g++ -m31 -fsigned-char -o imqspu32 imqspu32.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```


32비트 스레드 애플리케이션

```
g++ -m32 -fsigned-char -o imqspu32_r imqspu32.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

64비트 스레드되지 않은 애플리케이션

```
g++ -m64 -fsigned-char -o imqspu64 imqspu64.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

64비트 스레드 애플리케이션

```
g++ -m64 -fsigned-char -o imqspu64_r imqspu64.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

x86-64(32비트)

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

클라이언트: x86-64(32비트)

32비트 스레드되지 않은 애플리케이션

```
g++ -m32 -fsigned-char -o imqspu32c imqspu32c.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -L
MQ_INSTALLATION_PATH/lib -Wl,
-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl -limqb23gl -lmqic
```

32비트 스레드 애플리케이션

```
g++ -m32 -fsigned-char -o imqspu32c_r imqspu32c.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

64비트 스레드되지 않은 애플리케이션

```
g++ -m64 -fsigned-char -o imqspu64c imqspu64c.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl -limqb23gl
-lmqic
```

64비트 스레드 애플리케이션

```
g++ -m64 -fsigned-char -o imqspu64c_r imqspu64c.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

서버: x86-64(32비트)

32비트 스레드되지 않은 애플리케이션

```
g++ -m32 -fsigned-char -o imqspu32c imqspu32c.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl -limqb23gl -lmqm
```

32비트 스레드 애플리케이션

```
g++ -m32 -fsigned-char -o imqspu32c_r imqspu32c.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

64비트 스레드되지 않은 애플리케이션

```
g++ -m64 -fsigned-char -o imqspu64 imqspu64.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

64비트 스레드 애플리케이션

```
g++ -m64 -fsigned-char -o imqspu64_r imqspu64.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Windows Windows에 C++ 프로그램 빌드

C++ Microsoft Visual Studio 컴파일러를 사용하여 C++ Windows 프로그램을 IBM MQ 빌드하십시오.



주의: IBM MQ에서 제공하는 라이브러리는 정적 라이브러리가 아니라 동적 라이브러리입니다. IBM MQ는 컴파일 시간 중에만 사용할 수 있는 "import libraries"와 같은 항목을 제공합니다. 런타임의 경우 동적 라이브러리를 사용해야 합니다.

IBM MQ 8.0.0 Fix Pack 4부터 IBM MQ는 IBM MQ 애플리케이션 실행에 필요한 라이브러리가 포함된 재분배 가능한 클라이언트를 제공합니다. 이러한 라이브러리는 클라이언트 애플리케이션과 함께 패키지되고 재분배될 수 있습니다. 자세한 정보는 Windows의 재분배 가능 클라이언트를 참조하십시오.

32비트 애플리케이션에 사용하기 위한 라이브러리(.lib) 파일 및 dll 파일은 MQ_INSTALLATION_PATH/Tools/Lib에 설치됩니다. 64비트 애플리케이션에서 사용할 파일은 MQ_INSTALLATION_PATH/Tools/Lib64에 설치됩니다. MQ_INSTALLATION_PATH는 IBM MQ가 설치되어 있는 상위 레벨 디렉토리를 나타냅니다.

클라이언트

```
cl -MD imqspu64.cpp /Feimqspu64.exe imqb23vn.lib imqs23vn.lib
```

서버

```
cl -MD imqspu64.cpp /Feimqspu64.exe imqb23vn.lib imqs23vn.lib
```

유니버설 C 런타임 설치

Windows 8.1 또는 Windows Server 2012 R2를 사용 중인 경우에는 Microsoft에서 유니버설 C 런타임 업데이트(유니버설 CRT)를 설치해야 합니다. 이 런타임은 Windows 10 및 Windows Server 2016의 일부로서 포함되어 있습니다.

유니버설 CRT 업데이트는 Microsoft 업데이트 KB3118401입니다. C:\Windows\System32 디렉토리에서 ucrtbase.dll(이)라는 파일을 검색하여 이 업데이트가 있는지 확인할 수 있습니다. 보유 중이 아니면 Microsoft 페이지: <https://www.catalog.update.microsoft.com/Search.aspx?q=kb3118401>에서 업데이트를 다운로드할 수 있습니다.

런타임이 설치되지 않은 상태에서 프로그램이나 를 사용하여 직접 IBM MQ 컴파일한 프로그램을 Microsoft Visual Studio 2017 실행하려고 하면 다음과 같은 오류가 발생합니다:

```
The program can't start because api-ms-win-crt-runtime-l1-1-0.dll
is missing from your computer. Try reinstalling the program to
fix this problem.
```

Microsoft Visual Studio 2012 프로그램용 런타임 제공

를 사용하여 Microsoft Visual Studio 2012 프로그램을 IBM MQ 컴파일한 경우, 설치 IBM MQ 프로그램이 C/C++ + Microsoft Visual Studio 2012 런타임을 설치하지 않는다는 점에 유의하십시오. IBM MQ의 이전 버전이 동일한 컴퓨터에 설치되어 있으면 해당 설치에서 Microsoft Visual Studio 2012 런타임을 사용할 수 있습니다.

그러나 Microsoft Visual Studio 2012를 사용하여 빌드된 프로그램을 사용 중이며 IBM MQ의 이전 버전이 설치되어 있지 않으면 다음 작업 중 하나를 수행해야 합니다.

- 에서 **Microsoft Visual C++ Redistributable for VisualStudio 2017 (32 and 64-bit versions)** 를 다운로드하여 Microsoft 설치하십시오.
- Microsoft Visual Studio 2017 또는 런타임이 설치된 다른 Microsoft Visual Studio 레벨을 사용하여 프로그램을 다시 컴파일하십시오.

Microsoft Visual Studio 2015 컴파일러를 사용하여 빌드된 C++ 클라이언트 라이브러리

IBM MQ는 Microsoft Visual Studio 2015 C++ 컴파일러 및 Microsoft Visual Studio 2017 C++ 컴파일러로 빌드된 C++ 클라이언트 라이브러리를 제공합니다.

IBM MQ C++ 라이브러리의 32비트 및 64비트 버전이 모두 제공됩니다. 32비트 라이브러리는 bin\vs2015 폴더에 설치되고 64비트 라이브러리는 bin64\vs2015 폴더에 설치됩니다.

기본적으로 IBM MQ는 Microsoft Visual Studio 2017 라이브러리를 사용하도록 구성됩니다. 라이브러리를 Microsoft Visual Studio 2015 사용하려면 설치 전에 또는 또는 **setmqinst setmqenv** 명령을 사용하기 MQ_PREFIX_VS_LIBRARIES=vs2015 전에 IBM MQ 환경 MQ_PREFIX_VS_LIBRARIES 변수를 로 설정해야 합니다.

서로 다르게 이름 지정된 IBM MQ C++ 라이브러리 사용

IBM MQ는 이름이 서로 다른 일부 추가 C++ 클라이언트 라이브러리를 제공합니다. 이러한 라이브러리는 Microsoft Visual Studio 2015 및 Microsoft Visual Studio 2017 C++ 컴파일러로 빌드합니다. 이러한 라이브러리는 Microsoft Visual Studio 2017 C++ 컴파일러로 빌드된 기존 C++ 라이브러리와 함께 제공됩니다. 이러한 추가 IBM MQ C++ 라이브러리는 서로 다른 이름을 가지므로, 동일한 컴퓨터에서 C++을 IBM MQ 사용하여 빌드되고 해당 제품의 이전 버전으로 Microsoft Visual Studio 2017 컴파일된 C++ IBM MQ 애플리케이션을 실행할 수 있습니다.

추가 Microsoft Visual Studio 2017 라이브러리의 이름은 다음과 같습니다.

- imqb23vnvs2017.dll
- imqc23vnvs2017.dll
- imqs23vnvs2017.dll
- imqx23vnvs2017.dll

추가 Microsoft Visual Studio 2015 라이브러리의 이름은 다음과 같습니다.

- imqb23vnvs2015.dll
- imqc23vnvs2015.dll
- imqs23vnvs2015.dll
- imqx23vnvs2015.dll

이러한 라이브러리의 32비트 및 64비트 버전이 모두 제공됩니다. 32비트 라이브러리는 bin 폴더에 설치되고 64비트 라이브러리는 bin64 폴더에 설치됩니다. 해당 가져오기 라이브러리는 Tools\lib 및 Tools\lib64 디렉토리에 설치됩니다.

애플리케이션이 imq*vs2015.lib 파일을 사용하는 경우에는 Microsoft Visual Studio 2015 컴파일러를 사용하여 애플리케이션을 컴파일해야 합니다. 동일한 컴퓨터에서 로 컴파일된 C++ Microsoft Visual Studio 2015 애플리케이션 또는 제품의 이전 IBM MQ 버전으로 컴파일된 애플리케이션을 실행하려면 다음 예시와 같이 PATH 환경 변수에 접두사를 붙여야 합니다:

- 32비트 애플리케이션:

```
SET PATH=installation_folder\bin\vs2015;%PATH%
```

- 64비트 애플리케이션:

```
SET PATH=installation_folder\bin64\vs2015;%PATH%
```

관련 개념

Windows: IBM MQ 8.0의 변경사항

z/OS 배치, RRS 배치 및 CICS에서 C++ 프로그램 빌드

z/OS 에서 배치, RRS 배치 또는 CICS 환경에 대한 IBM MQ C++ 프로그램을 빌드하고 샘플 프로그램을 실행하십시오.

IBM MQ for z/OS에서 지원하는 세 환경에 맞는 C++ 프로그램을 작성할 수 있습니다.

- 배치
- RRS 배치
- CICS

컴파일, 사전 링크 및 링크

C++ 소스 코드를 컴파일, 사전 링크 및 링크-편집하여 z/OS 애플리케이션을 작성하십시오.

IBM MQ C++ for z/OS 는 IBM C++ for z/OS 언어의 z/OS DLL로 구현됩니다. DLL을 사용하여 사전 링크 시 제공된 정의의 사이트데크를 컴파일러 출력과 병합합니다. 그러면 링커가 IBM MQ C++ 멤버 함수에 대한 호출을 확인할 수 있습니다.

참고: 세 환경에는 각각 세 개의 사이트데크 세트가 있습니다.

IBM MQ for z/OS C++ 애플리케이션을 빌드하려면 JCL을 작성하여 실행하십시오. 다음 프로시저를 사용하십시오.

1. CICS에서 애플리케이션이 실행되는 경우 프로그램으로 CICS 제공 프로시저를 사용하여 CICS 명령을 변환하십시오.

또한 CICS 애플리케이션에서는 다음을 수행해야 합니다.

- a. SCSQLOAD 라이브러리를 DFHRPL 병합에 추가합니다.
- b. SCSQPROC 라이브러리의 멤버 IMQ4B100을 사용하여 CSQCAT1 CEDA 그룹을 정의합니다.
- c. CSQCAT1을 설치합니다.

2. 프로그램을 컴파일하여 오브젝트 코드를 생성합니다. 컴파일용 JCL에는 컴파일러에서 제품 데이터 정의 파일을 사용 가능하게 하는 명령문이 포함되어야 합니다. 데이터 정의는 다음 IBM MQ for z/OS 라이브러리에서 제공됩니다:

- **thlqual**.SCSQC370
- **thlqual**.SCSQHPPS

/cxx 컴파일러 옵션을 지정하십시오.

참고: **thlqual** 이름은 z/OS에서 IBM MQ 설치 라이브러리의 상위 레벨 규정자입니다.

3. **thlqual**.SCSQDEFS에서 제공되는 다음 정의의 사이트데크를 포함하여 [518 페이지의 『2』](#) 단계에서 작성한 오브젝트 코드를 사전 링크하십시오.

- a. 배치용 imqs23dm 및 imqb23dm
- b. RRS 배치용 imqs23dr 및 imqb23dr
- c. CICS용 imqs23dc 및 imqb23dc

해당 DLL은 다음과 같습니다.

- a. 배치용 imqs23im 및 imqb23im
- b. RSS 배치용 imqs23ir 및 imqb23ir

c. CICS용 imqs23ic 및 imqb23ic

4. 518 페이지의 『3』 단계에서 작성한 오브젝트 코드를 링크-편집하여 로드 모듈을 생성하고 애플리케이션 로드 라이브러리에 저장하십시오.

배치 또는 RRS 배치 프로그램을 실행하려면 STEPLIB 또는 JOBLIB 데이터 세트 병합에 **thlqual.SCSQAUTH** 및 **thlqual.SCSQLOAD** 라이브러리를 포함하십시오.

CICS 프로그램을 실행하려면 먼저 시스템 관리자가 해당 프로그램을 CICS에 IBM MQ 프로그램 및 트랜잭션으로 정의해야 합니다. 그런 다음 일반적인 방식으로 실행할 수 있습니다.

샘플 프로그램 실행

프로그램은 497 페이지의 『C++ 샘플 프로그램』에서 설명합니다.

샘플 프로그램은 소스 양식으로만 제공됩니다. 파일은 다음과 같습니다.

표 76. z/OS 샘플 프로그램 파일		
샘플	소스 프로그램(thlqual.SCSQCPPS 라이브러리에 있음)	JCL(thlqual.SCSQPROC 라이브러리에 있음)
HELLO WORLD	imqwrlld	imqwrlldr
SPUT	imqsptr	imqsptrr
SGET	imqsget	imqsgetr

샘플을 실행하려면 C++ 프로그램으로 컴파일하고 링크-편집하십시오(518 페이지의 『z/OS 배치, RRS 배치 및 CICS에서 C++ 프로그램 빌드』 참조). 제공된 JCL을 사용하여 배치 작업을 구성하고 실행하십시오. 포함된 주석에 따라 처음에 JCL을 사용자 정의해야 합니다.

z/OS UNIX System Services에 C++ 프로그램 빌드

'z/OS UNIX System Services'(z/OS UNIX)에서 'IBM MQ' C++ 프로그램을 빌드합니다.

z/OS UNIX 셸에서 애플리케이션을 빌드하려면 IBM MQ 포함 파일(thlqual.SCSQC370 및 thlqual.SCSQHPPS에 위치)에 대한 컴파일러 액세스를 제공하고 DLL 사이드데크(thlqual.SCSQDEFS에 위치) 중 두 개에 대해 링크해야 합니다. 런타임에 애플리케이션은 IBM MQ 데이터 세트 thlqual.SCSQLOAD, thlqual.SCSQAUTH 및 언어별 데이터 세트 중 하나(예: thlqual.SCSQANLE)에 대한 액세스가 필요합니다.⁶

컴파일

1. TSO **oput** 명령을 사용하여 샘플을 파일 시스템에 복사하거나 FTP를 사용하십시오. 이 예제의 나머지 부분에서는 샘플을 /u/fred/sample(이)라는 디렉토리에 복사하고 이름을 imqwrlld.cpp(으)로 지정했다고 가정합니다.
2. z/OS UNIX 셸에 로그인하여 샘플을 배치한 디렉토리로 변경하십시오.
3. DLL sidedeck과 .cpp 파일을 입력으로 승인할 수 있도록 다음과 같이 C++ 컴파일러를 설정하십시오.

```
/u/fred/sample:> export _CXX_EXTRA_ARGS=1
/u/fred/sample:> export _CXX_CXXSUFFIX=".cpp"
```

4. 샘플 프로그램을 컴파일하고 링크하십시오. 다음 명령을 배치 sidedeck과 프로그램을 링크합니다. RSS 배치 sidedeck을 대신 사용할 수 있습니다. \ 문자는 두 개 이상의 행으로 명령을 분할하는 데 사용합니다. 이 문자를 입력하지 마십시오. 한 행으로 명령을 입력하십시오.

```
/u/fred/sample:> c++ -o imqwrlld -I "'thlqual.SCSQC370'" \
-I "'thlqual.SCSQHPPS'" imqwrlld.cpp \
-I "'thlqual.SCSQDEFS(IMQS23DM)'" -I "'thlqual.SCSQDEFS(IMQB23DM)'"
```

⁶ 오브젝트 코드 사전 링크에 나열된 사이드데크 중 하나와 링크하여 세 가지 환경(518 페이지의 『z/OS 배치, RRS 배치 및 CICS에서 C++ 프로그램 빌드』) 중 하나에서 z/OS UNIX을(를) 실행할 수 있습니다

TSO **oput** 명령에 대한 자세한 정보는 [z/OS UNIX Command Reference](#)를 참조하십시오.

make 유틸리티를 사용해서도 C++ 프로그램 빌드를 간소화할 있습니다. 다음은 HELLO WORLD C++ 샘플 프로그램을 빌드하는 샘플 make 파일입니다. 컴파일과 링크 단계를 분리합니다. make를 실행하기 전에 [519 페이지의 『3』](#) 단계에서와 같이 환경을 설정하십시오.

```
flags = -I "'thlqual.SCSQC370'" -I "'thlqual.SCSQHPPS'"
decks = "'thlqual.SCSQDEFS(IMQS23DM)'" "'thlqual.SCSQDEFS(IMQB23DM)'"

imqwrl: imqwrl.o
    c++ -o imqwrl imqwrl.o $(decks)

imqwrl.o: imqwrl.cpp
    c++ -c -o imqwrl $(flags) imqwrl.cpp
```

make 사용에 대한 자세한 정보는 [z/OS UNIX System Services Programming Tools](#)를 참조하십시오.

실행 중

1. z/OS UNIX 셸에 로그인하여 샘플을 빌드한 디렉토리로 변경하십시오.
2. IBM MQ 데이터 세트를 포함하도록 STEPLIB 환경 변수를 설정하십시오.

```
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQLOAD
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQAUTH
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQANLE
```

3. 다음과 같이 샘플을 실행하십시오.

```
/u/fred/sample:> ./imqwrl
```

.NET 애플리케이션 개발

IBM MQ classes for .NET 애플리케이션이 로 IBM MQ MQI client IBM MQ 연결하거나 .NET IBM MQ 서버에 직접 연결할 수 있도록 허용합니다.

시작하기 전에

Removed 에서 비추천된 로 .NET Standard 2.0 구축된 IBM MQ 9.3.1 클라이언트 IBM MQ .NET 라이브러리는 에서 제품에서 IBM MQ 9.4.0 제거되었습니다.

V 9.4.0 **V 9.4.0** **Deprecated** 부터 IBM MQ 9.4.0, 에 IBM MQ classes for .NET, 데이터 직렬화 및 역직렬화에 사용되는 메서드 WriteObject(), ReadObject(), CreateObjectMessage(), 그리고 클래스 ObjectMessage 및 XmsObjectMessageImpl 는 더 이상 사용되지 않습니다.

Removed **V 10.0.0** 부터 IBM MQ classes for .NET IBM MQ 10.0.0, 객체의 직렬화 및 역직렬화에 사용되는 메서드 WriteObject(), ReadObject(), CreateObjectMessage() 및 클래스 ObjectMessage, XmsObjectMessageImpl 가 제거됩니다.

이 태스크 정보

IBM MQ classes for .NET는 .NET 애플리케이션과 IBM MQ의 상호 작용을 가능하게 하는 클래스 세트입니다. 애플리케이션에서 사용하는 IBM MQ의 다양한 구성요소(예: 큐 관리자, 큐, 채널 및 메시지)를 나타냅니다. 이러한 클래스에 대한 자세한 내용은 [클래스와 IBM MQ .NET 인터페이스](#)를 참조하십시오.

IBM MQ 9.4.0 대상 .NET 6 프레임워크를 기반으로 빌드된 클라이언트 IBM MQ .NET 라이브러리를 제공합니다. 자세한 정보는 [521 페이지의 『설치 IBM MQ classes for .NET』](#)의 내용을 참조하십시오.

, 에서 IBM MQ 9.4.0, IBM MQ classes for .NET 를 사용하는 IBM MQ 애플리케이션을 .NET 8 지원합니다. 자세한 정보는 [521 페이지의 『설치 IBM MQ classes for .NET』](#)의 내용을 참조하십시오.

V 10.0.0 에서 IBM MQ 10.0.0, 런타임 시 를 IBM MQ 사용하는 .NET 10 애플리케이션에 대한 IBM MQ classes for .NET 를 지원합니다. 새로운 런타임 또는 컴포넌트에서 실행될 때 기존 바이너리는 로드 또는 실행 실패와 같

은 동작상의 중대한 변경 사항을 겪을 수 있으며, 재컴파일이 필요할 수 있습니다. 자세한 정보는 [521 페이지의 『설치 IBM MQ classes for .NET』](#)의 내용을 참조하십시오.

를 사용하는 애플리케이션이 있고 의 기능을 Microsoft.NET Framework IBM MQ 활용하려면 를 IBM MQ classes for .NET Framework 사용하여 합니다. 자세한 정보는 [527 페이지의 『설치 IBM MQ classes for .NET Framework』](#)의 내용을 참조하십시오.

IBM MQ classes for .NET IBM MQ classes for .NET Framework ph의 차이점에 대한 자세한 내용은 을 참조하십시오 [521 페이지의 『설치 IBM MQ classes for .NET』](#).

.NET IBM MQ 관리되는 애플리케이션은 클러스터화된 큐 관리자 간에 연결을 자동으로 분산할 수 있습니다. IBM MQ classes for .NET 라이브러리와 lib IBM MQ classes for .NET Framework 라이브러리 모두 지원됩니다. 자세한 정보는 [균등 클러스터 정보 및 자동 애플리케이션 밸런싱](#)을 참조하십시오.

오브젝트 지향 IBM MQ .NET 인터페이스는 MQI 동사를 사용하지 않고 오브젝트의 메소드를 사용한다는 점에서 MQI 인터페이스와 다릅니다. 절차적 IBM MQ API(Application Programming Interface)는 목록에 있는 것과 같은 동사를 중심으로 빌드됩니다.

```
MQCONN, MQDISC, MQOPEN, MQCLOSE,
MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

이러한 동사는 모두 작동할 IBM MQ 오브젝트에 대한 핸들을 매개변수로 사용합니다. .NET은 오브젝트 지향이므로 .NET 프로그래밍 인터페이스는 사용하기가 더 쉽습니다. 사용자의 프로그램은 IBM MQ 오브젝트 세트로 구성되며, 사용자가 이러한 오브젝트의 메소드를 호출하여 작업을 수행합니다. .NET에서 지원되는 언어로 프로그램을 작성할 수 있습니다.

프로시저 인터페이스를 사용하는 경우 호출 MQDISC(*Hconn*, *CompCode*, *Reason*)를 사용하여 큐 관리자와 연결을 끊습니다. 여기서 *Hconn*은 큐 관리자에 대한 핸들입니다. .NET 인터페이스에서 큐 관리자는 MQQueueManager 클래스의 오브젝트로 표시됩니다. 해당 클래스에서 Disconnect() 메소드를 호출하여 큐 관리자와 연결을 끊습니다.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.Disconnect();
```

관련 개념

기술 개요

[5 페이지의 『IBM MQ용 애플리케이션 개발』](#)

메시지를 송신하고 수신하며, 큐 관리자와 관련 자원을 관리하기 위한 애플리케이션을 개발할 수 있습니다. IBM MQ는 많은 다양한 언어와 프레임워크로 작성된 애플리케이션을 지원합니다.

관련 태스크

[문제 해결 정보 수집](#)

[IBM MQ .NET 문제점 해결](#)

[1247 페이지의 『개발하는 Microsoft Windows Communication Foundation 애플리케이션 IBM MQ』](#)

WCF Windows Communication Foundation Microsoft (Windows Communication Foundation) 사용자 지정 채널은 WCF 클라이언트와 서비스 IBM MQ 간에 메시지를 송수신합니다.

[581 페이지의 『XMS .NET 애플리케이션 개발』](#)

IBM MQ Message Service Client (XMS) for .NET (XMS .NET)는 라는 애플리케이션 프로그래밍 XMS 인터페이스(API)를 제공하며, 이는 (JMS) Java Message Service API와 동일한 인터페이스 집합을 가집니다. IBM MQ Message Service Client (XMS) for .NET에는 XMS의 전체 관리 구현이 포함되며, 이는 .NET을 준수하는 어에서 사용할 수 있습니다.

Windows

Linux

설치 IBM MQ classes for .NET

IBM MQ classes for .NET 샘플을 포함한 모든 구성 요소가 에 IBM MQ 설치됩니다 Windows . Linux

필수조건 및 설치

IBM MQ 9.4.0 대상 .NET 6 프레임워크를 기반으로 빌드된 클라이언트 IBM MQ.NET 라이브러리를 제공합니다. 에서 IBM MQ 9.4.0, Microsoft .NET 6.0 는 타겟 프레임워크로 사용하여 .NET 6 빌드된 라이브러리를 IBM MQ 사용하는 애플리케이션을 실행하는 데 필요한 최소 버전입니다. 대상 프레임워크를 .NET 6 사용하여 빌드된 클라이언트 IBM MQ.NET 라이브러리는 에서 WindowsMQ_INSTALLATION_PATH/bin 로, 에서 MQ_INSTALLATION_PATH/lib64 Linux 로 제공됩니다.

, 에서 IBM MQ 9.4.0, IBM MQ classes for .NET 를 사용하는 IBM MQ 애플리케이션을 .NET 8 지원합니다. 해당 .NET 6 애플리케이션을 사용 중이라면, 파일을 runtimeconfig 약간 수정하여 값을 targetframeworkversion 로 설정함으로써 재컴파일 없이도 이 "net8.0" 애플리케이션을 실행할 수 있습니다.

V 9.4.0 **V 9.4.0** **Deprecated** 부터 IBM MQ 9.4.0, 에 IBM MQ classes for .NET, 데이터 직렬화 및 역직렬화에 사용되는 메서드 WriteObject(), ReadObject(), CreateObjectMessage(), 그리고 클래스 ObjectMessage 및 XmsObjectMessageImpl 는 더 이상 사용되지 않습니다.

Removed 에서 비추천된 로 .NET Standard 2.0 구축된 IBM MQ 9.3.1 클라이언트 IBM MQ.NET 라이브러리는 에서 제품에서 IBM MQ 9.4.0 제거되었습니다.

최신 버전의 IBM MQ classes for .NET Java 및 .NET 메시징 및 웹 서비스 기능의 표준 IBM MQ 설치 과정의 일부로 기본적으로 설치됩니다.

V 10.0 에서 IBM MQ 10.0.0, 런타임 시 를 IBM MQ 사용하는 .NET 10 애플리케이션에 대한 IBM MQ classes for .NET 를 지원합니다. 새로운 런타임 또는 컴포넌트에서 실행될 때 기존 바이너리는 로드 또는 실행 실패와 같은 동작상의 중대한 변경 사항을 겪을 수 있으며, 재컴파일이 필요할 수 있습니다. 또는 .NET 8.NET 6 애플리케이션을 사용 중인 경우, 파일을 runtimeconfig 약간 수정하여 값을 targetframeworkversion 로 설정하면 재컴파일 없이도 이 "net10.0" 애플리케이션을 실행할 수 있습니다.

V 10.0 IBM MQ classes for .NET는 IBM MQ IBM MQ 10.0.0 를 사용하는 애플리케이션을 위해 를 대상 프레임워크로 하여 .NET 10 구축된 클라이언트 라이브러리를 .NET 10 제공합니다. "net10.0" 또는 .NET 8 애플리케이션을 .NET 6 사용 중이라면, 파일에서 runtimeconfig 값을 targetframeworkversion 로 설정하도록 약간만 수정하면 재컴파일 없이도 이 애플리케이션을 실행할 수 있습니다.

Windows Windows의 필수조건 및 설치에 대한 자세한 정보는 다음을 참조하십시오.

- 실행하기 위한 필수 요구 사항 IBM MQ classes for .NET IBM MQ classes for .NET 소프트웨어는 에서 확인하십시오.
- 설치 지시사항은 Windows에 IBM MQ 서버 설치 또는 Windows 시스템에 IBM MQ 클라이언트 설치를 참조하십시오.

Linux Linux의 필수조건 및 설치에 대한 자세한 정보는 다음을 참조하십시오.

- 실행하기 위한 필수 요구 사항 IBM MQ classes for .NET IBM MQ classes for .NET 소프트웨어는 에서 확인하십시오.
- rpm 설치 지침은 시스템에 Linux IBM MQ 클라이언트 설치하기를 참조하십시오.
- Linux Ubuntu의 경우, Debian 패키지 사용은 Linux 시스템에 IBM MQ 클라이언트 설치를 참조하십시오.

IBM MQ classes for .NET Standard 라이브러리(amqmdnetstd.dll)는 NuGet 저장소에서 다운로드할 수 있습니다. 자세한 정보는 526 페이지의 『저장소에서 NuGet IBM MQ classes for .NET 다운로드 중』의 내용을 참조하십시오.

amqmdnetstd.dll 라이브러리

다음 위치에서 를 대상으로 하는 IBM MQ 9.4.0 .NET 6 프레임워크로 구축된 amqmdnetstd.dll 라이브러리를 사용할 수 있습니다:

- **Windows** 날짜 Windows: MQ_INSTALLATION_PATH\bin. 샘플 애플리케이션은 MQ_INSTALLATION_PATH\samp\dotnet\samples\cs\core\base에 설치됩니다.

- **Linux** 날짜 Linux: `MQ_INSTALLATION_PATH\lib64`. 샘플들은 `.NET.MQ_INSTALLATION_PATH\samp\dotnet\samples\cs\core\base`에 있습니다.



주의: **Removed** 에서 IBM MQ 9.4.0, 를 대상 IBM MQ.NET 프레임워크로 .NET Standard 2.0 사용하여 빌드된 클라이언트 라이브러리가 제거됩니다. 이 라이브러리들은 에서 사용 IBM MQ 9.3.1 중단되었습니다.

Stabilized .NET Framework의 `amqmdnet.dll` 라이브러리가 여전히 제공되지만 이 라이브러리는 안정화되어 있습니다. 즉, 새 기능이 도입되지 않습니다. 최신 기능을 위해 `amqmdnetstd.dll` 라이브러리로 마이그레이션해야 합니다. 그러나 해당 `amqmdnet.dll` 라이브러리는 버전이나 그 Continuous Delivery IBM MQ 9.1 이후 Long Term Support 버전에서 계속 사용할 수 있습니다.

라이브러리 `netstandard2.0` 제거 후 발생할 수 있는 두 가지 시나리오는 다음과 같습니다:

- 애플리케이션이 라이브러리(예 `netstandard2.0:amqmdnetstd.dll`)를 IBM MQ classes for .NET Framework 사용하여 빌드된 경우, 애플리케이션이 성공적으로 실행되도록 하려면 Microsoft.NET Framework 4.7.2 라이브러리(예: `amqmdnet.dll`)로 애플리케이션을 재빌드해야 합니다. 애플리케이션을 재빌드하지 않으면 "System.IO.Unexceptionable" 메시지가 발생할 수 있습니다:

예외 발생: System.IO.FileLoadException: 파일 또는 어셈블리 'amqmdnetstd, Version=9.3.5.0, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e ' 또는 그 종속성 중 하나를 로드할 수 없습니다. 찾은 어셈블리의 매니페스트 정의가 어셈블리 참조와 일치하지 않습니다. (0x80131040 예외 발생)
파일 이름: 'amqmdnetstd, Version=9.3.5.0, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e '
(SimplePut.SimplePut.PutMessages()
C:\SampleCode\Program 파일의 132번째 줄에 있는 SimplePut.SimplePut (String[] args)에서 오류가 발생했습니다.

- 해당 .NET 6 애플리케이션이 `netstandard2.0` 라이브러리로 빌드된 경우, 애플리케이션 런타임 디렉토리의 폴더에 `bin` 있는 동일한 .NET 6 라이브러리로 교체하기만 하면 됩니다. 재구축이 필요하지 않습니다.

참고: 대체 .NET 6 라이브러리는 항상 대체되는 `netstandard2.0` 라이브러리와 동일하거나 더 높은 수준이어야 합니다.

dspmqver 명령

dspmqver 명령을 사용하여 .NET Core 컴포넌트에 대한 버전 및 빌드 정보를 표시할 수 있습니다.

와 IBM MQ classes for .NET IBM MQ classes for .NET Framework 라이브러리) 간의 기능 비교

다음 표는 의 기능과 IBM MQ classes for .NET (IBM MQ classes for .NET Framework 라이브러리)의 기능을 비교하여 나열합니다

표 77. 와 IBM MQ classes for .NET IBM MQ classes for .NET Framework 간의 차이점 .		
기능	IBM MQ classes for .NET Framework	IBM MQ classes for .NET
클래스 이름(API)	모든 클래스는 각 네트워크에서 동일하게 유지됩니다.	모든 클래스는 각 네트워크에서 동일하게 유지됩니다.
운영 체제	Windows	Windows 도커화된(Dockerized) 컨테이너 Linux macOS

표 77. 와 IBM MQ classes for .NET IBM MQ classes for .NET Framework 간의 차이점 . (계속)

기능	IBM MQ classes for .NET Framework	IBM MQ classes for .NET
app.config 파일(재배포 가능 클라이언트에서 추적을 사용으로 설정하기 위한 구성 파일)	app.config 파일은 재배포 가능 패키지와 독립 IBM MQ.NET 실행형 클라이언트에 대한 추적을 활성화하는 데 사용됩니다. 추적에 사용하는 변수(예: MQTRACEPATH `` 및 <b>MQTRACELEVEL</b> ``)에 대한 자세한 내용은 애플리케이션 구성 파일을 사용하여 클라이언트 IBM MQ classes for .NET Framework 추적하기를 참조하십시오.	app.config은(는) 지원되지 않습니다. 환경 변수를 사용하십시오.
추적	의 전체 IBM MQ클라이언트 설치 시, 명령어를 strmqtrc 사용하여 의 추적을 활성화할 IBM MQ classes for .NET Framework 수 있습니다. 재배포 가능한 클라이언트의 경우, 해당 app.config 파일은 추적 기능을 활성화하는 데에도 사용됩니다. 자세한 내용은 애플리케이션 IBM MQ.NET 추적을 참조하십시오. 에서 IBM MQ 9.4.0, 파일을 mqclient.ini 사용하여 추적을 활성화하거나 비활성화하고 Trace 섹션의 적절한 속성을 설정할 수 있습니다. 추적 기능을 동적으로 활성화 및 비활성화하려면 파일을 mqclient.ini 사용하십시오. 자세한 내용은 을 참조하십시오 애플리케이션 IBM MQ.NET 추적 mqclient.ini.	환경 변수는 재배포 가능 클라이언트에 MQDOTNET_TRACE_ON 대한 추적을 활성화하는 데 사용됩니다. 0 이하의 값은 추적을 활성화하지 않습니다. 값 1은 기본 수준 추적을 활성화합니다. 값이 1보다 클 경우, 상세 추적을 활성화합니다. 이 환경 변수를 문자열과 같은 다른 값으로 설정해도 추적이 활성화되지 않습니다. 환경 변수를 사용하여 애플리케이션 IBM MQ.NET 추적하기를 참조하십시오. 환경 MQDOTNET_TRACE_ON 변수는 추적 IBM MQ 디렉터리가 사용 가능한지 여부를 확인합니다. 추적 디렉터리가 사용 가능한 경우, 추적 파일은 해당 디렉터리에 생성됩니다. 그러나 가 설치되어 있지 않은 IBM MQ 경우, 추적 파일은 현재 작업 디렉터리로 복사됩니다. 다른 환경 변수들(예: MQERRORPATH MQLOGLEVEL , MQSERVER , 등)도 동일한 방식으로 사용 IBM MQ classes for .NET Framework 및 작동할 수 있습니다. 에서 IBM MQ 9.4.0, 파일을 mqclient.ini 사용하여 추적을 활성화하거나 비활성화하고 Trace 섹션의 적절한 속성을 설정할 수 있습니다. 추적 기능을 동적으로 활성화 및 비활성화하려면 파일을 mqclient.ini 사용하십시오. 자세한 내용은 을 참조하십시오 애플리케이션 IBM MQ.NET 추적 mqclient.ini.
전송 모드	관리, 비관리 및 바인딩	관리됨

표 77. 와 IBM MQ classes for .NET IBM MQ classes for .NET Framework 간의 차이점 . (계속)

기능	IBM MQ classes for .NET Framework	IBM MQ classes for .NET
TLS	Windows 키 저장소가 인증서를 저장하는 데 사용됩니다.	Windows Windows에서는 인증서를 저장하는 데 키 저장소를 사용해야 합니다. 허용되는 값은 *USER 또는 *SYSTEM입니다. 입력을 기반으로 IBM MQ .NET 클라이언트에서 시스템 전체 또는 현재 사용자의 Windows 키 저장소를 확인합니다. Linux Linux에서는 X509Store 클래스를 사용하여 인증서를 설치하는 것이 좋으며 .NET Core에서는 ".dotnet/corefx/cryptography/x509stores" 위치에 인증서를 설치합니다. V 10.0.0 TLS 인증서를 저장할 사용자 지정 경로를 추가할 수 있으며 IBM MQ 10.0.0, 인증서에 대한 암호화된 암호를 생성하기 위해 일반 텍스트 암호 또는 Dotnet Password Encryptor(rundotnetcred) 도구를 사용할 수 있습니다. 자세한 내용은 일반 텍스트 또는 암호화된 암호를 사용하는 TLS 연결을 위한 .NET의 인증서 사용자 지정 경로를 참조하십시오.
CCDT	지원됨	지원됨, CCDT 경로 설정은 .NET Framework 클래스와 동일합니다.
클라이언트 자동 다시 연결	지원됨	지원됨
분배 트랜잭션	지원됨	지원되지 않음
글로벌 어셈블리 캐시 (GAC)로 동적 링크 라이브러리(dll)의 설치	Dll은 IBM MQ 설치의 일부로 GAC에 설치됩니다.	Dll은 IBM MQ 설치의 일부로 설치되지 않습니다.

참고: **Windows** Windows SID(Security Identifiers):

IBM MQ classes for .NET (.NET Standard 및 .NET 6 lib) 라이브러리에서는 도메인 수준 인증이 지원되지 않습니다. 로그인 사용자 ID는 인증에 사용됩니다.

macOS에서 IBM MQ .NET Core 애플리케이션 개발

macOS

IBM MQ .NET Core 애플리케이션은 macOS에서 개발할 수 있습니다.

IBM MQ .NET 라이브러리는 macOS 툴킷과 함께 패키징되지 않으므로 Windows 또는 Linux IBM MQ 클라이언트에서 macOS(으)로 복사해야 합니다. 그런 다음 이러한 라이브러리를 사용하여 macOS에서 IBM MQ .NET Core 애플리케이션을 개발할 수 있습니다.

개발이 완료되면 이러한 애플리케이션은 Windows 또는 Linux 환경에서 지원될 수 있습니다.

관련 개념

527 페이지의 [『설치 IBM MQ classes for .NET Framework』](#)

IBM MQ classes for .NET Framework 샘플을 포함하여 설치됩니다 IBM MQ. 에 대한 Microsoft.NET Framework 전제 Windows 조건이 있습니다.

585 페이지의 『설치 IBM MQ classes for XMS .NET』

IBM MQ classes for XMS .NET 샘플을 포함하여, 에 IBM MQ 설치됩니다 WindowsLinux .

Windows Linux 저장소에서 NuGet IBM MQ classes for .NET 다운로드 중

IBM MQ classes for .NET 파일들은 저장소에서 NuGet 다운로드 가능하므로 개발자들이 .NET 쉽게 활용할 수 있습니다.

이 태스크 정보

NuGet 개발 플랫폼을 Microsoft 위한 패키지 .NET 관리자입니다. NuGet 클라이언트 도구는 패키지를 생성하고 이용하는 기능을 제공합니다. NuGet 패키지는 컴파일된 코드(DLL)를 포함한 .nupkg 확장이 있는 단일 압축 파일, 해당 코드에 관련된 기타 파일 및 패키지의 버전 번호와 같은 정보를 포함한 기술적 Manifest입니다.

모든 패키지 작성자 및 이용자가 사용하는 중앙 패키지 저장소인 NuGet 갤러리에서 amqmdnetstd.dll 라이브러리를 포함한 IBMMQDotnetClient NuGet 패키지를 다운로드할 수 있습니다.

참고: 패키지는 NuGet 를 IBM MQ 9.4.0 대상으로 하는 .NET 8 프레임워크로 빌드된 라이브러리를 포함합니다.

Removed 에서 비추천된 로 .NET Standard 2.0 구축된 IBM MQ 9.3.1 클라이언트 IBM MQ.NET 라이브러리는 에서 제품에서 IBM MQ 9.4.0 제거되었습니다.

, 에서 IBM MQ 9.4.0, IBM MQ classes for .NET 를 사용하는 IBM MQ 애플리케이션을 .NET 8 지원합니다. 해당 .NET 6 애플리케이션을 사용 중이라면, 파일을 runtimeconfig 약간 수정하여 값을 targetframeworkversion 로 설정함으로써 재컴파일 없이도 이 "net8.0" 애플리케이션을 실행할 수 있습니다.

IBMMQDotnetClient 패키지를 다운로드하는 세 가지 방법이 있습니다.

- Microsoft Visual Studio를 사용합니다. NuGet은 Microsoft Visual Studio 확장으로서 배포됩니다. Microsoft Visual Studio 2012부터는 NuGet이 기본적으로 사전 설치됩니다.
- 명령행에서 NuGet 패키지 관리자 또는 .NET CLI를 사용합니다.
- 웹 브라우저를 사용합니다.

재배포 가능 패키지의 경우, 환경 변수를 사용하여 추적을 MQDOTNET_TRACE_ON 활성화합니다.

프로시저

- Microsoft Visual Studio에서 Package Manager UI를 사용하여 IBMMQDotnetClient 패키지를 다운로드하려면 다음 단계를 완료하십시오.
 - a) .NET 프로젝트를 마우스 오른쪽 단추로 클릭한 후 **Nuget 패키지 관리**를 클릭하십시오.
 - b) **찾아보기** 탭을 클릭하고, "IBMMQDotnetClient"를 검색하세요.
 - c) 해당 패키지를 선택하고 **설치**를 클릭하십시오.설치 중에, 패키지 관리자는 콘솔에 표시되는 문장의 형태로 진행상태 정보를 제공합니다.
- 명령행에서 IBMMQDotnetClient 패키지를 다운로드하려면 다음 옵션 중 하나를 선택하십시오.
 - NuGet 패키지 관리자에서 다음 명령을 입력하십시오.

```
Install-Package IBMMQDotnetClient -Version 9.1.4.0
```

설치 중에, 패키지 관리자는 콘솔에 표시되는 문장의 형태로 진행상태 정보를 제공합니다. 이 출력은 로그 파일로 경로 재지정할 수 있습니다.

- .NET CLI를 사용하여 다음 명령을 입력하십시오.

```
dotnet add package IBMMQDotnetClient --version 9.1.4
```


- 웹 브라우저를 사용하여 <https://www.nuget.org/packages/IBMQDotnetClient>에서 IBM MQDotnetClient 패키지를 다운로드하십시오.

관련 개념

[IBM MQ 라이선스 정보 .NET 클라이언트](#)

관련 태스크

589 페이지의 『[NuGet 저장소에서 IBM MQ classes for XMS .NET 다운로드](#)』

IBM MQ classes for XMS .NET은(는) NuGet 저장소에서 다운로드가 가능하기 때문에 .NET 개발자가 쉽게 이용할 수 있습니다.

Windows 설치 IBM MQ classes for .NET Framework

IBM MQ classes for .NET Framework 샘플을 포함하여 설치됩니다 IBM MQ. 에 대한 Microsoft.NET Framework 전제 Windows 조건이 있습니다.

최신 버전의 IBM MQ classes for .NET Framework 는 Java 및 .NET 메시징 및 웹 서비스 기능의 표준 IBM MQ 설치 과정의 일부로 기본적으로 설치됩니다. 설치 방법은 또는 [클라이언트 IBM MQ 설치 Windows 시스템을 서버 IBM MQ 설치 Windows](#) 참조하십시오.

V4.7.2 이상 버전을 Microsoft.NET Framework 설치해야 IBM MQ 9.3.0 실행할 IBM MQ classes for .NET Framework 수 있습니다.

Microsoft.NET Framework V3.5로 컴파일된 기존 애플리케이션은 애플리케이션의 app.config 파일에 다음 태그를 추가해서 다시 컴파일하지 않고 실행할 수 있습니다.

```
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2"/>
  </startup>
</configuration>
```

참고: V4.7.2 이상 Microsoft.NET Framework 버전이 설치 전에 설치되지 IBM MQ 애플리케이션의 경우, 제품 IBM MQ 설치는 오류 없이 진행되지만 해당 기능은 IBM MQ classes for .NET 사용할 수 없습니다. IBM MQ을(를) 설치한 후에 .NET Framework이(가) 설치된 경우에는 `WMQInstallDir\bin\amqiRegisterDotNet.cmd` 스크립트를 실행하여 IBM MQ.NET 어셈블리를 등록해야 합니다. 여기서, `WMQInstallDir`은 IBM MQ이(가) 설치된 디렉토리입니다. 이 스크립트는 GAC(Global Assembly Cache)에 필수 어셈블리를 설치합니다. 수행된 조치를 기록하는 `amqi*.log` 파일 세트가 `%TEMP%` 디렉토리에 작성됩니다. 이전 버전(예: .NET V3.5)에서 V4.7.2 이상으로 업그레이드하는 .NET 경우 `amqiRegisterDotNet.cmd` 스크립트를 다시 실행할 필요가 없습니다.

다중 설치 환경에서 이전에 IBM MQ classes for .NET을 지원 팩으로 설치한 경우에는 먼저 지원 팩을 설치 제거해야 IBM MQ를 설치할 수 있습니다. 설치된 기능은 IBM MQ classes for .NET 지원 팩과 IBM MQ 동일한 기능을 포함합니다.

소스 파일을 포함하는 샘플 애플리케이션도 제공됩니다. 528 페이지의 『[.NET용 샘플 애플리케이션](#)』의 내용을 참조하십시오.

.NET에서 Microsoft WCF용 IBM MQ 사용자 정의 채널을 사용하는 방법에 대한 정보는 1247 페이지의 『[개발하는 Microsoft Windows Communication Foundation 애플리케이션 IBM MQ](#)』의 내용을 참조하십시오.

관련 개념

521 페이지의 『[설치 IBM MQ classes for .NET](#)』

IBM MQ classes for .NET 샘플을 포함한 모든 구성 요소가 에 IBM MQ 설치됩니다 Windows . Linux

관련 태스크

[IBM MQ.NET 애플리케이션 추적](#)

IBM MQ classes for .NET를 큐 관리자에 연결하는 옵션

IBM MQ classes for .NET를 큐 관리자에 연결하는 모드는 세 가지가 있습니다. 요구사항에 가장 적합한 연결 유형을 고려하십시오.

클라이언트 바인딩 연결

IBM MQ classes for .NET 를 IBM MQ MQI client로 사용하기 위해 IBM MQ 서버 시스템 또는 별도의 시스템에 IBM MQ MQI client를 사용하여 설치할 수 있습니다. 클라이언트 바인딩 연결에서 XA 또는 비XA 트랜잭션을 사용할 수 있습니다.

서버 바인딩 연결

서버 바인딩 모드에서 사용할 때 IBM MQ classes for .NET에서 네트워크를 통해 통신하지 않고 큐 관리자 API를 사용합니다. 그러면 네트워크 연결을 사용하는 것보다 IBM MQ 애플리케이션의 성능이 향상됩니다.

바인딩 연결을 사용하려면 IBM MQ 서버에 IBM MQ classes for .NET 를 설치해야 합니다.

관리 클라이언트 연결

이 모드에서 설정된 연결은 IBM MQ 클라이언트로 로컬 시스템이나 원격 시스템에서 실행 중인 IBM MQ 서버에 연결합니다.

이 모드로 연결 중인 IBM MQ classes for .NET는 .NET 관리 코드에 그대로 남아 있지만 고유 서비스를 호출하지 않습니다. 관리 코드에 대한 자세한 정보는 Microsoft 문서를 참조하십시오.

관리 클라이언트를 사용하는 데 관한 많은 제한사항이 있습니다. 이에 대한 자세한 정보는 [542 페이지의 『관리 클라이언트 연결』](#)의 내용을 참조하십시오.

.NET용 샘플 애플리케이션

고유 .NET 애플리케이션을 실행하려면 확인 프로그램의 지시사항을 사용하여 샘플 애플리케이션 대신 애플리케이션 이름을 바꿉니다.

다음 샘플 애플리케이션이 제공됩니다.

- 메시지 넣기 애플리케이션
- 메시지 가져오기 애플리케이션
- 'hello world' 애플리케이션
- 발행/구독 애플리케이션
- 메시지 특성을 사용하는 애플리케이션

이러한 샘플 애플리케이션은 모두 C# 언어로 제공되며, 일부는 C++ 및 Visual Basic으로도 제공됩니다. .NET에서 지원되는 언어로 애플리케이션을 작성할 수 있습니다.

"메시지 넣기" 프로그램 SPUT (nmqsput.cs, mmqsput.cpp, vmqsput.vb)

이 프로그램은 이름 지정된 큐에 메시지를 두는 방법을 보여줍니다. 프로그램에는 다음 세 개의 변수가 있습니다.

- 큐의 이름(필수). 예: SYSTEM.DEFAULT.LOCAL.QUEUE
- 큐 관리자의 이름(선택사항)
- 채널의 정의(선택사항). 예: SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

큐 관리자 이름이 지정되지 않으면 큐 관리자가 기본값으로 기본 로컬 큐 관리자가 됩니다. 채널이 정의되면 MQSERVER 환경 변수와 형식이 동일하게 됩니다.

"메시지 가져오기" 프로그램 SGET (nmqsget.cs, mmqsget.cpp, vmqsget.vb)

이 프로그램은 이름 지정된 큐에서 메시지를 가져오는 방법을 보여줍니다. 프로그램에는 다음 세 개의 변수가 있습니다.

- 큐의 이름(필수). 예: SYSTEM.DEFAULT.LOCAL.QUEUE
- 큐 관리자의 이름(선택사항)
- 채널의 정의(선택사항). 예: SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

큐 관리자 이름이 지정되지 않으면 큐 관리자가 기본값으로 기본 로컬 큐 관리자가 됩니다. 채널이 정의되면 MQSERVER 환경 변수와 형식이 동일하게 됩니다.

"Hello World" 프로그램 (nmqwrld.cs, mmqwrld.cpp, vmqwrld.vb)

이 프로그램은 메시지를 넣고 가져오는 방법을 보여줍니다. 프로그램에는 다음 세 개의 변수가 있습니다.

- 큐의 이름(선택사항). 예: SYSTEM.DEFAULT.LOCAL.QUEUE or SYSTEM.DEFAULT.MODEL.QUEUE
- 큐 관리자의 이름(선택사항)
- 채널 정의(선택사항). 예: SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

큐 이름이 지정되지 않으면 이름은 기본값으로 SYSTEM.DEFAULT.LOCAL.QUEUE가 됩니다. 큐 관리자 이름이 지정되지 않으면 큐 관리자가 기본값으로 기본 로컬 큐 관리자가 됩니다.

"발행/구독" 프로그램(MQPubSubSample.cs)

이 프로그램은 IBM MQ 발행/구독 사용 방법을 보여줍니다. 이 프로그램은 C#으로만 제공됩니다. 이 프로그램에는 다음 두 개의 매개변수가 있습니다.

- 큐 관리자의 이름(선택사항)
- 채널 정의(선택사항)

"메시지 특성" 프로그램(MQMessagePropertiesSample.cs)

이 프로그램은 메시지 특성 사용 방법을 보여줍니다. 이 프로그램은 C#으로만 제공됩니다. 이 프로그램에는 다음 두 개의 매개변수가 있습니다.

- 큐 관리자의 이름(선택사항)
- 채널 정의(선택사항)

이러한 애플리케이션을 컴파일하고 실행하여 설치를 확인할 수 있습니다.

설치 위치

샘플 애플리케이션은 작성된 언어에 따라 다음 위치에 설치됩니다. *MQ_INSTALLATION_PATH*은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

C#

```
MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqswrld.cs
MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqspu.cs
MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsgt.cs
MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQPubSubSample.cs
MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQMessagePropertiesSample.cs
```

Managed C++

```
MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcpc\mmqswrld.cpp
MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcpc\mmqspu.cpp
MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcpc\mmqsgt.cpp
```

Visual Basic

```
MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqswrld.vb
MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqspu.vb
MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsgt.vb
MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqswwrld.vb
MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsput.vb
MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsget.vb
```

샘플 애플리케이션 빌드

샘플 애플리케이션을 빌드하기 위해 각 언어의 배치 파일이 제공됩니다.

C#

`MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\bldcssamp.bat`

bldcssamp.bat 파일에는 각 샘플의 행이 하나씩 포함되어 있습니다. 모두 이 샘플 프로그램을 빌드하는 데 필요합니다.

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin
/out:nmqwrld.exe nmqwrld.cs
```

Managed C++

`MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\bldmcp Samp.bat`

bldmcp Samp.bat 파일에는 각 샘플의 행이 하나씩 포함되어 있습니다. 모두 이 샘플 프로그램을 빌드하는 데 필요합니다.

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Microsoft Visual Studio 2003/.NET SDKv1.1에서 이러한 애플리케이션을 컴파일하려는 경우 다음 컴파일 명령을 바꾸십시오.

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

다음 항목으로 바꾸십시오.

```
cl /clr MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Visual Basic

`MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\bldvbsamp.bat`

bldvbsamp.bat 파일에는 각 샘플의 행이 하나씩 포함되어 있습니다. 모두 이 샘플 프로그램을 빌드하는 데 필요합니다.

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:vmqwrld.exe vmqwrld.vb
```

IBM MQ에서 Microsoft .NET Core를 사용하는 샘플

IBM MQ 환경에서의 Windows 응용을 IBM MQ.NET 위한 .NET Core 지지대. 샘플을 포함한 IBM MQ classes for .NET Standard는 기본적으로 IBM MQ 표준 설치의 일부로 설치됩니다.

IBM MQ .NET의 샘플 애플리케이션은 &MQINSTALL_PATH&/smp/dotnet/samples/cs/core/base에 설치됩니다. 샘플을 컴파일하는 데 사용할 수 있는 스크립트도 제공됩니다.

제공된 build.bat 파일을 사용하여 샘플을 빌드할 수 있습니다. Windows의 다음 위치에 각 샘플마다 build.bat이(가) 하나씩 있습니다.

- MQ\tools\dotnet\samples\cs\core\base\SimpleGet
- MQ\tools\dotnet\samples\cs\core\base\SimplePut

 IBM MQ 또한 환경 Linux 내 애플리케이션을 위한 Core를 지원합니다.

사용 IBM MQ 방법에 대한 자세한 Microsoft.NET Core 내용은 을 참조하십시오 [521 페이지의 『설치 IBM MQ classes for .NET』](#).

TCP/IP 클라이언트 연결을 승인하도록 큐 관리자 구성

클라이언트에서 수신되는 연결 요청을 승인하기 위해 큐 관리자를 구성합니다.

이 태스크 정보

이 태스크는 TCP/IP 클라이언트 연결을 승인하기 위해 큐 관리자를 구성하는 기본 단계를 설명합니다. 프로덕션 시스템에서 큐 관리자를 구성할 때 보안에 미치는 영향도 고려해야 합니다.

프로시저

1. 다음과 같이 서버 연결 채널을 정의하십시오.
 - a. 큐 관리자를 시작하십시오.
 - b. NET.CHANNEL이라는 샘플 채널을 정의합니다.

```
DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +  
DESCR('Sample channel for IBM MQ classes for .NET')
```

중요사항: 이 샘플에서는 보안에 미치는 영향을 고려하지 않으므로 샌드박스 환경에서만 사용해야 합니다. 프로덕션 시스템에서는 TLS 또는 보안 엑시트 사용을 고려하십시오. 자세한 정보는 [보안 IBM MQ](#)의 내용을 참조하십시오.

2. 리스너를 시작하십시오.

```
runmqclsr -t tcp [-m qmname] [-p portnum]
```

참고: 꺾쇠 괄호는 선택적 매개변수를 나타냅니다. 기본 큐 관리자에는 *qmname*이 필요하지 않고, 기본값(1414)을 사용하는 경우 포트 번호 *portnum*이 필요하지 않습니다.

.NET의 분산 트랜잭션

분산 트랜잭션 또는 글로벌 트랜잭션을 사용하면 클라이언트 애플리케이션이 하나의 트랜잭션에 두 개 이상의 네트워크 시스템에 있는 여러 다른 데이터 소스를 포함할 수 있습니다.

분산 트랜잭션에서 트랜잭션 관리자가 두 개 이상의 자원 관리자 간에 트랜잭션을 통합하고 관리합니다.

트랜잭션은 1단계 또는 2단계 커밋 프로세스일 수 있습니다. 1단계 커밋은 단 하나의 자원 관리자가 트랜잭션에 참여하는 프로세스이며 2단계 커밋에서는 두 개 이상의 자원 관리자가 트랜잭션에 참여합니다. 2단계 커밋 프로세스에서는 모든 자원 관리자가 커밋할 준비가 되었는지 확인하기 위해 트랜잭션 관리자가 준비 호출을 송신합니다. 모든 자원 관리자로부터 수신확인을 받으면 커밋 호출이 실행됩니다. 그렇지 않으면 전체 트랜잭션에서 롤백이 수행됩니다. 자세한 내용은 [트랜잭션 관리 및 지원](#)을 참조하십시오. 자원 관리자가 트랜잭션에 참여함을 트랜잭션 관리자에게 알려야 합니다. 자원 관리자가 참여함을 트랜잭션 관리자에게 알리면 트랜잭션을 커밋하거나 롤백할 때 자원 관리자가 트랜잭션 관리자에게 콜백을 받습니다.

IBM MQ .NET 클래스는 이미 비관리 서버 바인딩 모드 연결에서 분산 트랜잭션을 지원합니다. 이러한 모드에서 IBM MQ .NET 클래스는 모든 호출을 C 확장 트랜잭션 클라이언트에 위임하므로, 이 클라이언트에서 .NET 대신 트랜잭션 처리를 관리합니다.

IBM MQ.NET 클래스는 이제 IBM MQ .NET 클래스가 분산 트랜잭션 지원을 위해 System.Transactions 네임스페이스를 사용하는 관리 모드에서 분산 트랜잭션을 지원합니다. System.Transactions 인프라를 사용하면 IBM MQ를 포함하여 모든 자원 관리자에서 시작된 트랜잭션을 지원하므로 트랜잭션 프로그래밍이 간소화되고 효율적이게 됩니다. IBM MQ .NET 애플리케이션은 .NET 암시적 트랜잭션 프로그래밍 또는 명확한 트랜잭션 프로그래밍 모델을 사용하여 메시지를 넣고 가져올 수 있습니다. 암시적 트랜잭션에서 트랜잭션 경계는 트랜잭션을 커밋, 롤백(명확한 트랜잭션의 경우) 또는 완료할 시기를 결정하는 애플리케이션 프로그램에서 작성합니다. 명확한 트랜잭션에서는 트랜잭션을 커밋, 롤백 및 완료할지 여부를 명확하게 지정해야 합니다.

IBM MQ.NET에서 MS DTC(Microsoft Distributed Transaction Coordinator)를 트랜잭션 관리자로 사용하여, 여러 자원 관리자 간의 트랜잭션을 통합하고 관리합니다. IBM MQ는 자원 관리자로 사용합니다. XA 트랜잭션과 TLS를 사용할 수 없습니다. CCDT를 사용해야 합니다. 자세한 정보는 [TLS 채널과 확장된 트랜잭션 클라이언트 사용](#)을 참조하십시오.

IBM MQ.NET에서는 X/Open DTP(Distributed Transaction Processing) 모델을 따릅니다. X/Open DTP(Distributed Transaction Processing) 모델은 Open Group, 벤더 컨소시엄에서 제안한 분산 트랜잭션 처리 모델입니다. 이 모델은 트랜잭션 처리 및 데이터베이스 도메인에 있는 대부분의 상업 벤더 간에 표준입니다. 대부분의 상업 트랜잭션 관리 제품에서 X/DTP 모델을 지원합니다.

트랜잭션 모드

- [532 페이지의 『.NET 관리 모드의 분산 트랜잭션』](#)
- [비관리 모드에 대한 분산 트랜잭션](#)

다양한 시나리오의 통합 트랜잭션

- 연결이 여러 트랜잭션에 참여할 수 있지만, 항상 하나의 트랜잭션만 활성입니다.
- 트랜잭션 중에 MQQueueManager.Disconnect 호출을 사용합니다. 이 경우 트랜잭션에 롤백하도록 요청합니다.
- 트랜잭션 중에 MQQueue.Close 또는 MQTopic.Close 호출을 사용합니다. 이 경우 트랜잭션에 롤백하도록 요청합니다.
- 트랜잭션 경계는 트랜잭션을 커밋, 롤백(명확한 트랜잭션의 경우) 또는 완료(암시적 트랜잭션의 경우)할 시기를 결정하는 애플리케이션 프로그램에서 작성합니다.
- 큐 또는 토픽 호출에서 Put 또는 Get 호출을 실행하기 전에 트랜잭션 중에 클라이언트 애플리케이션이 중단되고 예상치 못한 오류가 발생하면 트랜잭션이 롤백되고 MQException이 발생합니다.
- 큐 또는 토픽 호출에서 Put 또는 Get 호출 중에 MQCC_FAILED가 리턴되면 이유 코드와 함께 MQException이 발생하고 트랜잭션이 롤백됩니다. 트랜잭션 관리자가 준비 호출을 이미 실행한 경우 IBM MQ .NET에서 트랜잭션을 강제로 롤백하여 준비 요청을 리턴합니다. 그러면 트랜잭션 관리자 DTC가 현재 주변에서 발생하는 트랜잭션의 모든 자원 관리자와 함께 현재 작업을 롤백합니다.
- 여러 자원 관리자와 관련된 트랜잭션 중에 환경상의 이유로 인해 Put 또는 Get 호출이 무기한 정지되면 트랜잭션 관리자가 지정된 시간까지 기다립니다. 이 시간이 지나고 나면 현재 주변에서 발생하는 트랜잭션의 모든 자원 관리자와 함께 모든 현재 작업을 롤백합니다. 준비 단계 중에 무한 대기가 발생하면 트랜잭션 관리자가 제한 시간을 초과시키거나 자원에서 인다우트(in-doubt)를 발행하여 트랜잭션을 롤백할 수 있습니다.
- 트랜잭션을 사용하는 애플리케이션은 SYNC_POINT의 Put 또는 Get 메시지여야 합니다. SYNC_POINT에 없는 트랜잭션 컨텍스트에서 메시지 Put 또는 Get 호출이 실행되면 호출에 실패하고 MQRC_UNIT_OF_WORK_NOT_STARTED 이유 코드가 표시됩니다.

Microsoft.NET System.Transactions 네임스페이스를 사용하는 관리 및 비관리 클라이언트 트랜잭션 지원 사이의 동작 차이점

중첩 트랜잭션은 다른 TransactionScope에 TransactionScope가 있음

- IBM MQ .NET 전체 관리 클라이언트는 중첩 TransactionScope를 지원합니다.
- IBM MQ .NET 비관리 클라이언트는 중첩 TransactionScope를 지원하지 않습니다.

System.Transactions의 종속 트랜잭션

- IBM MQ .NET 전체 관리 클라이언트는 System.Transactions에서 제공한 종속 트랜잭션 기능을 지원합니다.
- IBM MQ .NET 비관리 클라이언트는 System.Transactions에서 제공하는 종속 트랜잭션 기능을 지원하지 않습니다.

제품 샘플

제품 샘플 SimpleXAPut 및 SimpleXAGet은(는) WebSphere MQ\tools\dotnet\samples\cs\base에서 사용 가능합니다. 샘플은 C# 애플리케이션이며, SystemTransactions 네임스페이스를 사용하여 분산 트랜잭션에서 MQPUT 및 MQGET을 사용하는 방식을 데모합니다. 이러한 샘플에 대한 자세한 정보는 [535 페이지의 『TransactionScope에서 단순 메시지 넣기 및 가져오기 작성』](#)의 내용을 참조하십시오.

.NET 관리 모드의 분산 트랜잭션

IBM MQ .NET 클래스에서는 관리 모드의 분산 트랜잭션을 지원하기 위해 System.Transactions 네임스페이스를 사용합니다. 관리 모드에서 MS DTC가 트랜잭션에 관련된 모든 서버의 분산된 트랜잭션을 통합하고 관리합니다.

IBM MQ .NET 클래스는 System.Transactions.Transaction 클래스를 기반으로 하는 명시적 프로그래밍 모델과 System.Transactions.TransactionScope 클래스를 사용하는 암시적 프로그래밍 모델을 제공합니다. 여기서 트랜잭션은 인프라에서 자동으로 관리합니다.

암시적 트랜잭션

다음 코드는 IBM MQ .NET 애플리케이션이 .NET 암시적 트랜잭션 프로그래밍을 사용하여 메시지를 넣는 방법을 설명합니다.

```
Using (TransactionScope scope = new TransactionScope ())
{
    Q.Put (putMsg,pmo);
    scope.Complete ();
}

Q.close();
qMgr.Disconnect();}
```

암시적 트랜잭션의 코드 플로우에 대한 설명

이 코드는 *TransactionScope*를 작성하고 범위 내에 메시지를 넣습니다. 그런 다음 *Complete*를 호출하여 트랜잭션 통합자에게 트랜잭션 완료를 알립니다. 이제 트랜잭션 조정자가 *prepare* 및 *commit*를 발행하여 트랜잭션을 완료합니다. 문제가 감지되면 *rollback*이 호출됩니다.

명시적 트랜잭션

다음 코드는 IBM MQ .NET 애플리케이션이 application puts messages using .NET 명시적 트랜잭션 프로그래밍 모델을 사용하여 메시지를 넣은 방법을 설명합니다.

```
MQQueueManager qMgr = new MQQueueManager ("MQQM");
MQQueue Q = QMGR.AccessQueue("Q", MQC.MQOO_OUTPUT+MQC.MQOO_INPUT_SHARED);
MQPutMessageOptions pmo = new MQPutMessageOptions();
pmo.Options = MQC.MQPMO_SYNCPOINT;
MQMessage putMsg1 = new MQMessage();
Using(CommittableTransaction tx = new CommittableTransaction()){
    Transaction.Current = tx;
    try
    {
        Q.Put(MSG,pmo);
        tx.commit();
    }
    catch(Exception)
    {tx.rollback();}
}

Q.close();
qMgr.Disconnect();
}
```

명시적 트랜잭션의 코드 플로우에 대한 설명

코드의 일부는 *CommittableTransaction* 클래스를 사용하여 트랜잭션을 작성합니다. 해당 범위에 메시지를 넣은 다음 명시적으로 *commit*를 호출하여 트랜잭션을 완료합니다. 문제가 있으면 *rollback*이 호출됩니다.

.NET 비관리 모드의 분산 트랜잭션

IBM MQ.NET 클래스는 암시적 또는 명확한 트랜잭션 프로그래밍 모델을 사용하여 확장 트랜잭션 클라이언트 및 COM+/MTS를 트랜잭션 통합자로 사용하여 비관리 연결(클라이언트)을 지원합니다. 비관리 모드에서 IBM MQ .NET 클래스는 .NET 대신 트랜잭션 처리를 관리하는 C 확장 트랜잭션 클라이언트에 모든 호출을 위임합니다.

트랜잭션 처리는 외부 트랜잭션 관리자가 제어하며, 트랜잭션 관리자의 API가 제어하는 글로벌 작업 단위를 통합합니다. MQBEGIN, MQCMIT 및 MQBACK 동사는 사용할 수 없습니다. IBM MQ .NET 클래스는 비관리 전송 모드(C 클라이언트)를 통해 이 지원을 표시합니다. [XA 준수 트랜잭션 관리자 구성](#)을 참조하십시오.

MTS는 트랜잭션 처리 (TP) 시스템으로 전개되어 CICS, Tuxedo 및 기타 플랫폼에서 사용 가능한 것과 동일한 기능을 Windows NT 에 제공합니다. MTS가 설치되면 Microsoft Distributed Transaction Coordinator(MSDTC)라는 Windows NT에 별도의 서비스가 추가됩니다. MSDTC은(는) 별도의 데이터 저장소 또는 자원에 걸쳐 있는 트랜잭션을 조정합니다. 이 서비스가 작동하려면 각 데이터 저장소에서 고유 독점 자원 관리자를 구현해야 합니다.

IBM MQ 인터페이스(독점 리소스 관리자 인터페이스)를 구현하여 DTC XA 호출을 IBM MQ(X/Open) 호출로 매핑함으로써 호환성을 MSDTC 확보합니다. IBM MQ는 자원 관리자의 역할을 수행합니다.

COM+ 요청과 같은 컴포넌트가 IBM MQ에 액세스하면 일반적으로 COM이 적당한 MTS 컨텍스트 오브젝트에 트랜잭션이 필요한지 확인합니다. 트랜잭션이 필요하다면 COM에서 DTC에 이를 알리고 이 조작의 필수 IBM MQ 트랜잭션을 자동으로 시작합니다. 그런 다음 COM이 MQMTS 소프트웨어를 통해 데이터에 대한 작업을 수행하여 필요한 대로 메시지를 넣거나 가져옵니다. 데이터에서 모든 조치가 완료되고 나면 COM에서 확보한 오브젝트 인

스턴스가 SetComplete 또는 SetAbort 메소드를 호출합니다. 애플리케이션에서 SetComplete을 발행하면 이 호출에서 DTC에 애플리케이션이 트랜잭션을 완료했으므로 DTC가 2단계 커밋 프로세스를 수행할 수 있음을 알립니다. 그러면 DTC에서 MQMTS를 호출하고, 이 호출을 통해 다시 IBM MQ를 호출하여 트랜잭션을 커밋하거나 롤백합니다.

비관리 클라이언트를 사용하여 IBM MQ .NET 애플리케이션 작성

COM+의 컨텍스트에서 실행하려면 .NET 클래스가 System.EnterpriseServices.ServicedComponent에서 상속해야 합니다. 서비스된 컴포넌트를 사용하는 어셈블리를 작성하기 위한 규칙과 권장사항은 다음과 같습니다.

참고: 다음은 System.EnterpriseServices 모드를 사용하는 경우에만 적절합니다.

- COM+에서 시작하는 클래스와 메소드는 모두 공용이어야 합니다(내부 클래스가 아니며 보호 또는 정적 메소드가 아님).
- 클래스와 메소드 속성: TransactionOption 속성은 클래스의 트랜잭션 레벨을 나타냅니다. 즉, 트랜잭션이 사용 불가능한지, 지원되는지, 아니면 필수인지를 나타냅니다. ExecuteUOW() 메소드의 AutoComplete 속성은 COM+에 핸들링하지 않은 예외가 발생하지 않은 경우 트랜잭션을 커밋하도록 지시합니다.
- 어셈블리에 강력한 이름(Strong Name) 지정: 어셈블리는 강력한 이름이 지정되어야 하며 GAC(Global Assembly Cache)에 있어야 합니다. 어셈블리는 COM+에 명시적으로 등록되거나 GAC에 등록된 후 지연 등록(lazy registration)을 통해 등록합니다.
- COM+에 어셈블리 등록: COM 클라이언트에 공개할 어셈블리를 준비하십시오. 그런 다음 어셈블리 등록 도구인 regasm.exe를 사용하여 유형 라이브러리를 작성하십시오.

```
regasm UnmanagedToManagedXa.dll
```

- 어셈블리를 GAC gacutil /i UnmanagedToManagedXa.dll에 등록하십시오.
- .NET 서비스 설치 프로그램 도구, regsvcs.exe를 사용하여 COM+에 어셈블리를 등록합니다. 다음을 통해 regasm에서 작성한 유형 라이브러리를 확인하십시오.

```
Regsvcs /appname:UnmanagedToManagedXa /tlb:UnmanagedToManagedXa.tlb UnmanagedToManagedXa.dll
```

- 어셈블리가 GAC에 배치되고 나중에 지연 등록을 통해 COM+에 등록됩니다. 코드를 처음 실행한 후 .NET 프레임워크에서 등록을 처리합니다.

System.EnterpriseServices 모델 및 COM+와 System.Transactions를 사용하는 예제 코드 플로우는 다음 섹션에서 설명합니다.

System.EnterpriseServices 모델을 사용하는 예제 코드 플로우

```
using System;
using IBM.WMQ;
using IBM.WMQ.Nmqi;
using System.Transactions;
using System.EnterpriseServices;

namespace UnmanagedToManagedXa
{
    [ComVisible(true)]
    [System.EnterpriseServices.Transaction(System.EnterpriseServices.TransactionOption.Required)]
    public class MyXa : System.EnterpriseServices.ServicedComponent
    {
        public MQQueueManager QMGR = null;
        public MQQueueManager QMGR1 = null;
        public MQQueue QUEUE = null;
        public MQQueue QUEUE1 = null;
        public MQPutMessageOptions pmo = null;
        public MQMessage MSG = null;

        public MyXa()
        {
        }

        [System.EnterpriseServices.AutoComplete()]
        public void ExecuteUOW()
        {
            QMGR = new MQQueueManager("usemq");

            QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                     MQC.MQOO_INPUT_SHARED +
                                     MQC.MQOO_OUTPUT +
```

```

MQC.MQOO_BROWSE);

        pmo = new MQPutMessageOptions();
        pmo.Options = MQC.MQPMO_SYNCPOINT;
        MSG = new MQMessage();
        QUEUE.Put(MSG, pmo);
        QMGR.Disconnect();
    }
}

public void RunNow()
{
    MyXa xa = new MyXa();
    xa.ExecuteUOW();
}

```

COM+와 트랜잭션에 System.Transactions를 사용하는 예제 코드 플로우

```

[STAThread]
public void ExecuteUOW()
{
    Hashtable t1 = new Hashtable();
    t1.Add(MQC.CHANNEL_PROPERTY, "SYSTEM.DEF.SVRCONN");
    t1.Add(MQC.HOST_NAME_PROPERTY, "localhost");
    t1.Add(MQC.PORT_PROPERTY, 1414);
    t1.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_CLIENT);
    TransactionOptions opts = new TransactionOptions();

    using(TransactionScope scope = new TransactionScope(TransactionScopeOption.RequiresNew,
        opts, EnterpriseServicesInteropOption.Full)
    {
        QMGR = new MQQueueManager("usemq", t1);
        QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
            MQC.MQOO_INPUT_SHARED +
            MQC.MQOO_OUTPUT +
            MQC.MQOO_BROWSE);

        pmo = new MQPutMessageOptions();
        pmo.Options = MQC.MQPMO_SYNCPOINT;
        MSG = new MQMessage();
        QUEUE.Put(MSG, pmo);
        scope.Complete();
    }
    QMGR.Disconnect();
}

```

TransactionScope에서 단순 메시지 넣기 및 가져오기 작성

제품 샘플 C# 애플리케이션은 IBM MQ에서 사용할 수 있습니다. 이러한 단순 애플리케이션은 TransactionScope에서 메시지 넣기 및 가져오기를 시연합니다. 태스크 종료 시 큐 또는 토픽에서 메시지를 넣고 가져올 수 있습니다.

시작하기 전에

MSDTC 서비스가 실행 중이며 XA 트랜잭션에 사용되어야 합니다.

이 태스크 정보

예는 단순 애플리케이션, SimpleXAPut 및 SimpleXAGet입니다. SimpleXAPut 및 SimpleXAGet 프로그램은 IBM MQ에서 사용할 수 있는 C# 애플리케이션입니다. SimpleXAPut은 SystemTransactions 네임스페이스를 사용하여 분산 트랜잭션에서 MQPUT을 사용하여 시연합니다. SimpleXAGet은 SystemTransactions 네임스페이스를 사용하여 분산 트랜잭션에서 MQGET을 사용하여 시연합니다.

SimpleXAPut은(는) MQ\tools\dotnet\samples\cs\base에 있음

프로시저

tools\dotnet\samples\cs\base\bin에서 명령행 매개변수로 애플리케이션을 실행할 수 있음

```
SimpleXAPut.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n  
numberOfMsgs]
```

```
SimpleXAGet.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n  
numberOfMsgs]
```

여기서 매개변수는 다음과 같습니다.

-destinationURI

큐 또는 토픽일 수 있습니다. 큐는 `queue://queueName`으로 지정하고 토픽은 `topic://topicName`으로 지정하십시오.

-host

localhost 또는 IP 주소와 같은 호스트 이름일 수 있습니다.

-port

큐 관리자가 실행 중인 포트입니다.

-channel

사용 중인 연결 채널입니다. 기본값은 `SYSTEM.DEF.SVRCONN`입니다.

-transaction

트랜잭션 결과(예: 커밋 또는 롤백)입니다.

-mode

전송 모드(예: 관리 또는 비관리)입니다.

-numberOfMsgs

메시지 수입니다. 기본값은 1입니다.

예

```
SimpleXAPut -d topic://T01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

```
SimpleXAGet -d queue://Q01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

IBM MQ .NET에서 트랜잭션 복구

이 섹션에서는 관리 모드를 사용하여 IBM MQ .NET XA에서 트랜잭션 복구 프로세스를 설명합니다.

이 태스크 정보

분산 트랜잭션 처리에서는, 트랜잭션이 성공적으로 완료될 수 있지만 여러 가지 이유로 트랜잭션이 실패할 수 있는 시나리오가 있을 수 있습니다. 이러한 이유로는 시스템 장애, 하드웨어 장애, 네트워크 오류, 올바르게 않거나 유효하지 않은 데이터, 애플리케이션 오류 또는 자연 재해나 인재가 포함될 수 있습니다. 트랜잭션 실패는 예방할 수 없습니다. 분산 트랜잭션 시스템은 이러한 실패를 핸들링할 수 있어야 합니다. 오류가 발생하는 경우 오류를 감지하고 수정할 수 있어야 합니다. 이 프로세스는 트랜잭션 복구라고 합니다.

분산 트랜잭션 처리의 중요한 요소는 불완전 또는 인다우트 트랜잭션을 복구하는 것입니다. 특정 트랜잭션의 작업 단위 부분은 복구할 때까지 잠긴 상태로 유지되므로 복구를 실행하는 것이 중요합니다. `System.Transactions` 클래스 라이브러리의 `Microsoft.NET`에서 불완전/인다우트 트랜잭션을 복구하는 옵션을 제공합니다. 이 복구 지원에서는 자원 관리자가 트랜잭션 로그를 유지보수하고 필요할 때 복구를 실행해야 합니다.

`Microsoft .NET` 트랜잭션 복구 모델에서 트랜잭션 관리자(`System.Transactions` 또는 `MS DTC(Microsoft Distributed Transaction coordinator)` 또는 둘 다)가 트랜잭션 복구를 시작, 통합 및 제어합니다. `OLE Tx` 프로토콜(`Microsoft XA` 프로토콜) 기반 자원 관리자가 복구를 구동, 통합 및 제어하도록 구성하는 옵션을 제공합니다. 이 작업을 수행하려면 자원 관리자가 고유 인터페이스를 사용하여 `MS DTC`에 `XA_Switch`를 등록해야 합니다.

XA_Switch는 Distributed Transaction Coordinator의 자원 관리자에 XA 함수(예: xa_start, xa_end 및 xa_recover)의 시작점을 제공합니다.

Microsoft DTC(Distributed Transaction Coordinator)를 사용하여 복구:

Microsoft Distributed Transaction Coordinator는 두 가지 유형의 복구 프로세스를 제공합니다.

콜드 복구

콜드 복구는 XA 자원 관리자에 대한 연결이 열린 동안 트랜잭션 관리자 프로세스에 실패하는 경우 수행합니다. 트랜잭션 관리자가 재시작되면 트랜잭션 관리자 로그를 읽고 XA 자원 관리자에 대한 연결을 재설정하는 다음 복구를 시작합니다.

핫 복구

핫 복구는 XA 자원 관리자나 네트워크가 실패하여 트랜잭션 관리자와 XA 자원 관리자 사이의 연결에 실패한 경우 트랜잭션 관리자가 그대로 실행되면 수행합니다. 실패 후에 트랜잭션 관리자가 주기적으로 XA 자원 관리자에 다시 연결하기 위해 시도합니다. 연결을 재설정하면 트랜잭션 관리자가 XA 복구를 시작합니다.

System.Transactions 네임스페이스는 MS DTC를 기반으로 하는 분산 트랜잭션의 구현을 트랜잭션 관리자로 제공합니다. 완전히 관리되는 환경에서 MS DTC의 고유 인터페이스 기능과 비슷한 기능을 제공합니다. 유일한 차이점은 트랜잭션 복구입니다. System.Transactions에서는 자원 관리자가 직접 복구를 구동한 다음 트랜잭션 관리자(MS DTC)와 통합해야 합니다. 자원 관리자가 특정 불완전 트랜잭션의 복구를 요청하고 나면 트랜잭션 관리자가 이 요청을 승인하고 특정 트랜잭션의 실제 결과를 기반으로 통합합니다.

IBM MQ .NET의 트랜잭션 복구 프로세스

이 절에서는 IBM MQ .NET 클래스를 사용하여 분산 트랜잭션을 복구할 수 있는 방법에 대해 설명합니다.

개요

미완료 트랜잭션을 복구하려면 복구 정보가 필요합니다. 트랜잭션 복구 정보는 자원 관리자가 스토리지에 로깅해야 합니다. IBM MQ .NET 클래스는 유사한 경로를 따릅니다. 트랜잭션 복구 정보는 SYSTEM.DOTNET.XARECOVERY.QUEUE라는 시스템 큐에 로깅됩니다.

IBM MQ .NET의 트랜잭션 복구는 다음과 같은 두 단계 프로세스입니다.

1. SYSTEM.DOTNET.XARECOVERY.QUEUE에 트랜잭션 복구 정보 로깅.
2. XA 모니터 애플리케이션 WmqDotnetXAMonitor를 사용하여 트랜잭션 복구.

SYSTEM.DOTNET.XARECOVERY.QUEUE

SYSTEM.DOTNET.XARECOVERY.QUEUE는 미완료 트랜잭션에 대한 트랜잭션 복구 정보를 보유하는 시스템 큐입니다. 이 큐는 큐 관리자가 작성될 때 작성됩니다.

모든 트랜잭션에서 준비 단계 중에 복구 정보를 포함하는 지속 메시지가 SYSTEM.DOTNET.XARECOVERY.QUEUE에 추가됩니다. 커밋 호출에 성공하면 메시지가 삭제됩니다.

참고: SYSTEM.DOTNET.XARECOVERY.QUEUE 큐를 삭제해선 안됩니다.

WMQDotnetXAMonitor 애플리케이션

IBM MQ .NET XA 모니터 애플리케이션, WmqDotnetXAMonitor는 큐 관리자를 모니터링하고 SYSTEM.DOTNET.XARECOVERY.QUEUE의 메시지를 처리하며 미완료 트랜잭션을 복구하는 .NET 관리 애플리케이션입니다.

메시지 채널 에이전트(MCA)는 메시지를 목적지 큐에 넣을 수 없는 경우, 원래 메시지를 포함한 예외 보고서를 생성하고 원래 메시지에 지정된 응답 대상 큐에 전송되도록 이 메시지를 전송 큐에 넣습니다. (응답 대상 큐가 MCA와 같은 큐 관리자에 있으면, 메시지를 전송 큐가 아닌 해당 큐에 바로 넣음).

다음은 미완료 트랜잭션으로 간주되어 복구됩니다.

- 트랜잭션이 준비되었지만 제한시간 내에 COMMIT이 완료되지 않았습니다.
- 트랜잭션이 준비되었지만 IBM MQ 큐 관리자가 작동 중지되었습니다.
- 트랜잭션이 준비된 후 트랜잭션 관리자가 작동 중지되었습니다.

XA 모니터 애플리케이션은 IBM MQ .NET 클라이언트 애플리케이션이 실행 중인 동일한 시스템에서 실행해야 합니다. 여러 시스템에서 실행 중이며 동일한 큐 관리자에 연결하는 애플리케이션이 있는 경우에는 모든 시스템에서 WmqDotnetXAMonitor 애플리케이션을 실행해야 합니다. 각 클라이언트 시스템마다 애플리케이션을 복구하기 위해 실행되는 XA 모니터 애플리케이션의 인스턴스가 있지만, 각 XA 모니터 인스턴스는 현재 XA 모니터의 로컬 MS DTC가 조정 중인 트랜잭션에 해당하는 메시지를 식별할 수 있어야 트랜잭션을 다시 등록하고 완료할 수 있습니다.

관련 개념

538 페이지의 『IBM MQ .NET의 트랜잭션 복구 유스 케이스』

트랜잭션을 복구해야 하는 여러 유스 케이스가 있습니다.

관련 태스크

539 페이지의 『WMQDotnetXAMonitor 애플리케이션 사용』

IBM MQ .NET 클라이언트는 불완전한 분산 트랜잭션을 복구하는 데 사용할 수 있는 XA 모니터 애플리케이션 WmqDotnetXAMonitor를 제공합니다. WmqDotnetXAMonitor 애플리케이션은 트랜잭션이 인다우트 상태인 큐 관리자에 대한 연결을 설정한 후 사용자가 설정한 매개변수를 기반으로 트랜잭션을 해결합니다.

IBM MQ .NET의 트랜잭션 복구 유스 케이스

트랜잭션을 복구해야 하는 여러 유스 케이스가 있습니다.

- **IBM MQ 단일 DTC 및 단일 큐 관리자 인스턴스를 사용하는 애플리케이션:** 이 경우 큐 관리자에 연결하고 트랜잭션에서 작업 단위(UoW)를 실행할 때, 트랜잭션이 실패하고 미완료 상태가 되면 XA 모니터 애플리케이션이 트랜잭션을 복구하고 완료합니다.

이 유스 케이스에서는, 단일 큐 관리자가 트랜잭션과 연관되어 있기 때문에 XA 모니터 애플리케이션의 단일 인스턴스가 실행됩니다.

- **단일 DTC 및 단일 큐 관리자 인스턴스를 사용하는 다중 IBM MQ 애플리케이션:** 이 유스 케이스에서는, 단일 DTC에 둘 이상의 IBM MQ 애플리케이션이 있고 모두 동일한 큐 관리자에 연결되어 있으며 트랜잭션에서 UoW를 실행합니다.

트랜잭션이 실패하고 미완료 상태가 되면 XA 모니터 애플리케이션은 이를 복구하고 모든 애플리케이션에 관련된 트랜잭션을 완료합니다.

이 유스 케이스에서는, 하나의 큐 관리자가 트랜잭션에서 사용되기 때문에 XA 모니터 애플리케이션의 단일 인스턴스가 실행됩니다.

- **다중 IBM MQ 애플리케이션, 다중 DTC, 여러 다른 큐 관리자 인스턴스:** 이 유스 케이스에서는, 여러 다른 DTC에 IBM MQ 애플리케이션이 둘 이상 있으며(즉, 각 애플리케이션이 다른 시스템에서 실행 중임) 여러 다른 큐 관리자에 연결되어 있습니다.

실패가 발생하고 트랜잭션이 미완료 상태가 되면 모니터 애플리케이션에서 메시지의 TransactionManagerWhereabouts를 확인하여 DTC 주소를 판별합니다. TransactionManagerWhereabouts 값이 모니터를 실행 중인 DTC 주소와 일치하면 복구를 완료하고, 그렇지 않으면 해당 DTC에 해당하는 메시지를 찾을 때까지 계속 검색합니다.

이 유스 케이스에서는, 각 클라이언트마다 트랜잭션에 사용되는 자체 큐 관리자가 있기 때문에 클라이언트(사용자 또는 컴퓨터)당 실행 중인 XA 모니터 애플리케이션의 인스턴스가 하나만 있습니다.

- **다중 IBM MQ 애플리케이션, 다중 DTC, 동일한 다중 큐 관리자 인스턴스:** 이 유스 케이스에서는, 여러 다른 DTC에 IBM MQ 애플리케이션이 둘 이상 있으며(즉, 각 애플리케이션이 다른 시스템에서 실행 중임) 모두 동일한 큐 관리자에 연결되어 있습니다.

실패가 발생하고 트랜잭션이 미완료 상태가 되면 모니터 애플리케이션이 메시지의 TransactionManagerWhereabouts를 확인하여 DTC 주소와 값이 모니터를 실행 중인 DTC와 일치하는지 확인합니다. 두 값이 모두 일치하면 복구를 완료하고, 그렇지 않으면 DTC에 해당하는 메시지를 찾을 때까지 계속 검색합니다.

이 유스 케이스에서는, 각 클라이언트마다 트랜잭션에 사용되는 자체 큐 관리자 연관이 있기 때문에 클라이언트(사용자 또는 컴퓨터)당 실행 중인 XA 모니터 애플리케이션의 인스턴스가 하나만 있습니다.

- **다중 IBM MQ 애플리케이션, 단일 DTC, 여러 다른 큐 관리자 인스턴스:** 이 유스 케이스에서는, 단일 DTC에 둘 이상의 IBM MQ 애플리케이션이 있으며(즉, 컴퓨터에 둘 이상의 IBM MQ 애플리케이션이 실행 중임) 여러 다른 큐 관리자에 연결되어 있습니다.

트랜잭션에 실패하고 미완료 상태가 되면 모니터 애플리케이션이 트랜잭션을 복구합니다.

이 유스 케이스에서는 각 애플리케이션이 트랜잭션에서 고유 큐 관리자를 사용하며 각 큐 관리자를 복구해야 하므로, 연결된 큐 관리자 수만큼 많은 모니터 애플리케이션 인스턴스가 실행 중입니다.

참고: XA 모니터 애플리케이션이 백그라운드에서 실행되고 있지 않으면 이를 시작할 수 있습니다.

관련 개념

537 페이지의 『IBM MQ .NET의 트랜잭션 복구 프로세스』

이 절에서는 IBM MQ .NET 클래스를 사용하여 분산 트랜잭션을 복구할 수 있는 방법에 대해 설명합니다.

관련 태스크

539 페이지의 『WMQDotnetXAMonitor 애플리케이션 사용』

IBM MQ .NET 클라이언트는 불완전한 분산 트랜잭션을 복구하는 데 사용할 수 있는 XA 모니터 애플리케이션 WmqDotnetXAMonitor를 제공합니다. WmqDotnetXAMonitor 애플리케이션은 트랜잭션이 인다우트 상태인 큐 관리자에 대한 연결을 설정한 후 사용자가 설정한 매개변수를 기반으로 트랜잭션을 해결합니다.

WMQDotnetXAMonitor 애플리케이션 사용

IBM MQ .NET 클라이언트는 불완전한 분산 트랜잭션을 복구하는 데 사용할 수 있는 XA 모니터 애플리케이션 WmqDotnetXAMonitor를 제공합니다. WmqDotnetXAMonitor 애플리케이션은 트랜잭션이 인다우트 상태인 큐 관리자에 대한 연결을 설정한 후 사용자가 설정한 매개변수를 기반으로 트랜잭션을 해결합니다.

이 태스크 정보

WMQDotnetXAMonitor 애플리케이션은 수동으로 실행해야 합니다. 언제든지 시작할 수 있습니다.

SYSTEM.DOTNET.XARECOVERY.QUEUE에 메시지가 표시되면 애플리케이션을 시작하거나, IBM MQ .NET 클래스를 사용하여 작성된 애플리케이션에 대한 트랜잭션 작업을 수행하기 전에 백그라운드에서 애플리케이션을 계속 실행할 수 있습니다.

명령행을 통해서 또는 애플리케이션 구성 파일을 사용하여 WMQDotnetXAMonitor의 매개변수 값을 설정할 수 있습니다. 애플리케이션 구성 파일을 통해 제공된 값은 명령행을 통해 설정된 값보다 우선합니다.

WMQDotnetXAMonitor 가 설정하는 IBM MQ 9.3.0연결은 보안되지 않은 연결입니다.

IBM MQ 9.3.0부터는, WMQDotnetXAMonitor의 추가 매개변수를 설정하여 큐 관리자에 대한 보안 연결을 설정하는 옵션이 제공됩니다.

프로시저

- 애플리케이션 구성 파일을 사용하여 WmqDotNETXAMonitor에 입력을 제공하려면 [541 페이지의 『WmqDotNETXAMonitor 애플리케이션 구성 파일 설정』](#)의 내용을 참조하십시오.
- 명령행에서 WMQDotnetXAMonitor 애플리케이션을 시작하려면 필요한 매개변수와 함께 다음 명령을 사용하십시오.

이전 IBM MQ 9.3.0:

```
WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i
```

IBM MQ 9.3.0부터:

```
WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i -k SSL Key Repository -s Cipher Spec
```

지정할 수 있는 매개변수는 다음과 같습니다.

-m QueueManagerName

큐 관리자의 이름.

선택사항

-n ConnectionName

호스트(포트) 형식의 연결 이름입니다. ConnectionName은 둘 이상의 연결 이름을 포함할 수 있습니다. 여러 연결 이름은 쉼표로 구분된 목록으로 제공해야 합니다(예: localhost (1414) ,

localhost (1415), localhost (1416)). WMQDotnetXAMonitor 애플리케이션은 실패로 구분된 목록에 지정된 각 연결 이름에 대해 복구를 실행합니다.

-c ChannelName

채널 이름입니다.

-i

경험적 분기 완료.

선택사항

-k SSL Key Repository

SSL 키 저장소의 이름입니다. 지원되는 값은 다음과 같습니다.

- *SYSTEM(기본값)

- *사용자

선택사항

-s Cipher Spec

설정된 CipherSpec은 지원되는 버전에 대한 CipherSpec 중 하나여야 하며, 가급적 Windows 그룹 정책에 지정된 것과 동일할 수 있습니다. 자세한 정보는 [562 페이지의 『관리 대상 .NET 클라이언트의 CipherSpec 지원』](#)의 내용을 참조하십시오.

큐 관리자에 대한 보안 연결을 설정하려는 경우 필수입니다.

-dn SSLPeer Name

피어 큐 관리자로부터 인증서의 구별 이름(DN)을 검사하는 데 사용되는 SSL 피어 이름입니다.

선택사항

-cl Certificate Label

인증서를 식별하는 레이블 이름입니다.

선택사항

-sn OutboundSNI

SNI(Server Name Indication)를 TLS 연결을 시작할 때 원격 시스템에 대한 대상 IBM MQ 채널 이름으로 또는 호스트 이름으로 설정해야 하는지 여부입니다. 이 옵션의 지원되는 값은 다음과 같습니다.

- CHANNEL(기본값)

- 호스트 이름

- *

값을 설정하지 않으면 기본값인 CHANNEL이 사용됩니다.

선택사항

-cr Certificate Revocation Check

인증 취소 검사를 수행할지 여부입니다. 이 옵션의 지원되는 값은 다음과 같습니다.

- true

- false(기본값)

선택사항

-kr KeyResetCount

암호화에 사용되는 비밀 키가 재조정되기 전에 채널에서 송신 및 수신된 암호화되지 않은 총 바이트 수입니다.

기본값 0은 비밀 키가 재조정되지 않음을 나타냅니다.

선택사항

WMQDotnetXAMonitor 애플리케이션은 다음과 같은 조치를 수행합니다.

1. 100초 간격으로 SYSTEM.DOTNET.XARECOVERY.QUEUE의 큐 용량을 확인합니다.
2. 큐 용량이 0보다 크면 큐에서 메시지를 찾아보고 메시지가 불완전 트랜잭션 기준을 충족하는지 확인합니다.
3. 메시지가 불완전 트랜잭션 기준을 충족하면 메시지를 제거하고 트랜잭션 복구 정보를 검색합니다.

4. 복구 정보가 로컬 MS DTC(Microsoft Distributed Transaction Coordinator)와 관련되어 있는지 여부를 판별합니다. 관련된 경우에는 WMQDotnetXAMonitor가 계속해서 트랜잭션을 복구하고, 그렇지 않으면 돌아가서 다음 메시지를 찾습니다.
5. 불완전 트랜잭션을 복구하기 위해 큐 관리자를 호출합니다.

WmqDotNETXAMonitor 애플리케이션 구성 파일 설정

애플리케이션 구성 파일을 사용하여 IBM MQ .NET XA Monitor 애플리케이션, WmqDotNETXAMonitor에 대한 입력을 제공할 수 있습니다. 샘플 애플리케이션 구성 파일은 IBM MQ .NET과 함께 제공됩니다. 요구사항에 따라 이 샘플 파일을 수정할 수 있습니다.

애플리케이션 구성 파일을 통해 제공된 입력 값이 우선순위가 가장 높습니다. 539 페이지의

『WMQDotnetXAMonitor 애플리케이션 사용』 및 애플리케이션 구성 파일에 설명된 대로 명령행에서 두 입력 값을 모두 제공하는 경우에는 애플리케이션 구성 파일의 값이 우선합니다.

. IBM MQ 9.3.0이전에 대한 샘플 애플리케이션 구성 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value="" />
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
</dnetxa>
</dnetxa>
</configuration>
```

IBM MQ 9.3.0의 샘플 애플리케이션 구성 파일.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value="" />
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
<add key="SSLKeyRepository" value="" />
<add key="SSLCipherSpec" value="" />
<add key="SSLPeerName" value="" />
<add key="SSLKeyResetCount" value="" />
<add key="SSLCertRevocationCheck" value="" />
<add key="CertificateLabel" value="" />
<add key="OutboundSNI" value="" />
</dnetxa>
</dnetxa>
</configuration>
```

WmqDotNetXAMonitor 애플리케이션 로그

Monitor 애플리케이션은 Monitor의 진행 상태와 트랜잭션 복구 상태를 로깅하기 위한 애플리케이션 디렉토리에 로그 파일을 작성합니다. 로깅은 연결 이름과 채널 세부 사항으로 시작하며 복구를 실행 중인 현재 큐 관리자를 보여줍니다.

복구가 시작되면 트랜잭션 복구 메시지의 MessageId, 미완료 트랜잭션의 TransactionId 및 Transaction Manager Coordination당 트랜잭션의 실제 결과가 로깅됩니다.

샘플 로그 파일:

```

Time|ProcessId|ThreadId|WMQ .NET XA Recovery Monitor, Running now for
ConnectionName:xxxx, Time|ProcessId|ThreadId|Channel=xxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId|Recovery Completed for TransactionId= xxxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId| Recovery Completed for TransactionId= xxxxx

```

IBM MQ .NET 프로그램 작성 및 배치

큐에 IBM MQ 접근하려면 IBM MQ classes for .NET, 가 지원하는 어떤 언어로든 프로그램을 작성하여 큐에 .NET 메시지를 넣고, 큐에서 메시지를 가져오는 IBM MQ 호출을 포함시킵니다.

시작하기 전에

V 9.4.0 **V 9.4.0** **Deprecated** 부터 IBM MQ 9.4.0, 에 IBM MQ classes for .NET, 데이터 직렬화 및 역직렬화에 사용되는 메서드 WriteObject(), ReadObject(), CreateObjectMessage(), 그리고 클래스 ObjectMessage 및 XmsObjectMessageImpl 는 더 이상 사용되지 않습니다.

Removed 에서 비추천된 로 .NET Standard 2.0 구축된 IBM MQ 9.3.1클라이언트 IBM MQ.NET 라이브러리는 에서 제품에서 IBM MQ 9.4.0제거되었습니다.

이 태스크 정보

IBM MQ 문서에는 C#, C++ 및 Visual Basic 언어에 대한 정보만 포함되어 있습니다.

이 섹션의 주제는 시스템과 IBM MQ 상호작용하는 애플리케이션 작성에 도움이 되는 정보를 제공합니다. 개별 클래스에 대한 세부사항은 [IBM MQ .NET 클래스 및 인터페이스](#)를 참조하십시오.

연결 차이점

IBM MQ.NET용으로 프로그래밍하는 방식은 사용할 연결 모드에 따라 일부 달라집니다.

일부 기능은 관리 클라이언트에 사용할 수 없으므로 IBM MQ classes for .NET를 관리 클라이언트로 사용하는 경우 표준 IBM MQ MQI client와 여러 차이점이 있습니다.

IBM MQ.NET은 연결 이름, 채널 이름, 사용자 정의 값 NMQ_MQ_LIB 및 MQC.TRANSPORT_PROPERTY 특성의 설정에서 사용할 연결 유형을 판별합니다.

관리 클라이언트 연결

IBM MQ classes for .NET가 관리 클라이언트로 사용되는 경우 표준 IBM MQ MQI client와 여러 차이점이 있습니다.

다음 기능은 관리 클라이언트에서 사용할 수 없습니다.

- 채널 압축
- 채널 엑시트 체인
- Advanced Message Security (AMS)

관리 클라이언트에서 이러한 기능을 사용하려고 하면 MQException이 리턴됩니다. 연결의 클라이언트 측에서 오류가 감지되면 MQRC_ENVIRONMENT_ERROR 이유 코드를 사용합니다. 서버 측에서 감지되면 서버에서 리턴한 이유 코드가 사용됩니다.

비관리 클라이언트용으로 작성된 채널 엑시트는 작동하지 않습니다. 특별히 관리 클라이언트의 새 엑시트를 작성해야 합니다. 클라이언트 채널 정의 테이블(CCDT)에 지정된 올바르지 않은 채널 엑시트가 없는지 확인하십시오.

관리 채널 엑시트의 이름은 최대 999자일 수 있습니다. 그러나 CCDT를 사용하여 채널 엑시트 이름을 지정하는 경우 128자로 제한됩니다.

통신은 TCP/IP를 통해서만 지원됩니다.

endmqm 명령을 사용하여 큐 관리자를 중지하면 .NET 관리 클라이언트에 대한 서버 연결 채널이 다른 클라이언트에 대한 서버 연결 채널보다 닫는 데 오래 걸립니다.

관리 형 IBM MQ 문제 진단을 사용하기 위해 *NMQ_MQ_LIB*를 managed로 설정했다면, 명령어의 **stimqtrc** 다음 매개변수들은 지원되지 않습니다: -i, -p, -s, -b, -c.

XA 트랜잭션을 사용하는 관리 .NET 애플리케이션은 z/OS 큐 관리자에서 작동하지 않습니다. 관리되는 .NET 클라이언트가 큐 z/OS 관리자에 연결을 시도할 때 MQOPEN 호출 시 MQRC_UOW_ENLISTMENT_ERROR(mqrc=2354) 오류가 발생합니다. 그러나 X 트랜잭션을 사용하는 관리 .NET 애플리케이션이 분산 큐 관리자에서 작동합니다.

관련 정보

[AMS의 알려진 제한사항](#)

사용할 연결 유형 정의

연결 유형은 연결 이름, 채널 이름, 사용자 정의 값 NMQ_MQ_LIB 및 특성 MQC.TRANSPORT_PROPERTY의 설정을 통해 판별합니다.

다음과 같이 연결 이름을 지정할 수 있습니다.

- MQQueueManager 구성자에서 명확하게 지정합니다.

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- MQC.HOST_NAME_PROPERTY를 설정하고 선택적으로 MQQueueManager 구성자의 해시 테이블 입력 항목에서 MQC.PORT_PROPERTY를 설정하여 지정합니다.

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- 명확한 MQEnvironment 값으로 지정합니다.

```
MQEnvironment.Hostname
```

MQEnvironment.Port(선택사항).

- MQC.HOST_NAME_PROPERTY 특성을 설정하고 선택적으로 MQEnvironment.properties 해시 테이블에 MQC.PORT_PROPERTY를 설정하여 지정합니다.

다음과 같이 채널 이름을 지정할 수 있습니다.

- MQQueueManager 구성자에서 명확하게 지정합니다.

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- MQQueueManager 구성자의 해시 테이블 입력 항목에 MQC.CHANNEL_PROPERTY 특성을 설정하여 지정합니다.

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- 명확한 MQEnvironment 값으로 지정합니다.

```
MQEnvironment.Channel
```

- MQEnvironment.properties 해시 테이블에서 MQC.CHANNEL_PROPERTY 특성을 설정하여 지정합니다.

다음과 같이 전송 특성을 지정할 수 있습니다.

- MQQueueManager 구성자의 해시 테이블 입력 항목에 MQC.TRANSPORT_PROPERTY 특성을 설정하여 지정합니다.

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- MQEnvironment.properties 해시 테이블에서 MQC.TRANSPORT_PROPERTY 특성을 설정하여 지정합니다.

다음 값 중 하나를 사용하여 필요한 연결 유형을 선택하십시오.

MQC.TRANSPORT_MQSERIES_BINDINGS - 서버로 연결
 MQC.TRANSPORT_MQSERIES_CLIENT - 비XA 클라이언트로 연결
 MQC.TRANSPORT_MQSERIES_XACLIENT - XA 클라이언트로 연결
 MQC.TRANSPORT_MQSERIES_MANAGED - 비XA 관리 클라이언트로 연결

다음 테이블에 표시된 대로 명확하게 연결 유형을 선택하도록 사용자 정의 값 NMQ_MQ_LIB를 설정할 수 있습니다.

NMQ_MQ_LIB 값	연결 유형
mqic.dll	비XA 클라이언트로 연결
mqicxa.dll	XA 클라이언트로 연결
mqm.dll	서버 또는 비XA 클라이언트로 연결
관리됨	비XA 관리 클라이언트로 연결

참고: 이전 릴리스와의 호환성을 위해 mqic32.dll 및 mqic32xa.dll의 값은 mqic.dll 및 mqicxa.dll의 동의어로 허용됩니다. 그러나 mqm.dll과 mqm.pdb는 IBM WebSphere MQ 7.1 이상 클라이언트의 일부일 뿐입니다.

환경에서 사용할 수 없는 연결 유형을 선택하는 경우, 예를 들어 mqic32xa.dll을 지정하고 XA 지원이 없는 경우 IBM MQ.NET에서 예외가 발생합니다.

NMQ_MQ_LIB를 "managed"로 설정하면 클라이언트가 관리 IBM MQ 문제점 진단 테스트, .NET 데이터 변환 및 기타 관리 하위 레벨 IBM MQ 함수를 사용하게 됩니다.

NMQ_MQ_LIB의 기타 모든 값은 .NET 프로세스에서 비관리 IBM MQ 문제점 진단 테스트와 데이터 변환 및 기타 비관리 하위 레벨 IBM MQ 함수를 사용하게 합니다(시스템에 IBM MQ MQI client 또는 서버가 설치되었다고 가정).

IBM MQ.NET에서는 다음과 같이 연결 유형을 선택합니다.

1. MQC.TRANSPORT_PROPERTY가 지정된 경우 MQC.TRANSPORT_PROPERTY의 값에 따라 연결합니다.

그러나 MQC.TRANSPORT_PROPERTY를 MQC.TRANSPORT_MQSERIES_MANAGED로 설정해도 클라이언트 프로세스가 반드시 관리된 상태로 실행되지는 않습니다. 이 설정을 사용해도 다음과 같은 경우 클라이언트가 관리되지 않습니다.

- MQC.TRANSPORT_PROPERTY가 MQC.TRANSPORT_MQSERIES_MANAGED 이외의 것으로 설정되어 프로세스의 다른 스레드가 연결된 경우.
- NMQ_MQ_LIB가 "managed"로 설정되지 않은 경우 문제점 진단 테스트, 데이터 변환 및 기타 하위 레벨 함수가 완전히 관리되지 않습니다(시스템에 IBM MQ MQI client 또는 서버가 설치되었다고 가정).

2. 연결 이름이 채널 이름 없이 지정되었거나 채널 이름이 연결 이름 없이 지정된 경우 오류가 발생합니다.

3. 연결 이름과 채널 이름이 모두 지정된 경우 다음과 같습니다.

- NMQ_MQ_LIB가 mqic32xa.dll로 설정된 경우 XA 클라이언트로 연결됩니다.
- NMQ_MQ_LIB가 managed로 설정된 경우 관리된 클라이언트로 연결됩니다.
- 그렇지 않으면 비XA 클라이언트로 연결됩니다.

4. NMQ_MQ_LIB가 지정된 경우 NMQ_MQ_LIB의 값에 따라 연결됩니다.

5. IBM MQ 서버가 설치된 경우 서버로 연결됩니다.

6. IBM MQ MQI client가 설치된 경우 비XA 클라이언트로 연결됩니다.

7. 그렇지 않으면 관리된 클라이언트로 연결됩니다.

Windows IBM MQ .NET 프로젝트 템플릿 사용

IBM MQ .NET 클라이언트는 .NET Core 애플리케이션 개발을 지원하는 프로젝트 템플릿을 사용할 수 있는 기능을 제공합니다.

시작하기 전에

Microsoft Visual Studio 2017 이상 및 .NET Core 2.1이 시스템에 있어야 합니다.

.NET 템플릿을 복사해야 합니다.

```
&MQ_INSTALL_ROOT\tools\dotnet\samples\cs\core\base\ProjectTemplates\IBMMQ.NETClientApp.zip
```

디렉토리에서

```
&USER_HOME_DIRECTORY\Documents\&Visual_Studio_Version\Templates\ProjectTemplates
```

디렉토리로 복사합니다. 여기서,

- `&MQ_INSTALL_ROOT`는 설치의 루트 디렉토리입니다.
- `&USER_HOME_DIRECTORY`는 홈 디렉토리입니다.

템플릿을 선택하려면 Microsoft Visual Studio를 중지한 후 재시작해야 합니다.

이 태스크 정보

.NET 프로젝트 템플릿은 애플리케이션을 개발하는 데 사용할 수 있는 몇 가지 공통 코드를 포함합니다. 내장 코드를 통해 IBM MQ 큐 관리자에 연결하고, 내장 코드에서 특성을 수정하여 넣기 및 가져오기 조작을 수행할 수 있습니다.

프로시저

1. Microsoft Visual Studio를 여십시오.
2. 파일을 클릭하고 새로 작성, 프로젝트를 연속해서 클릭하십시오.
3. 새 프로젝트 만들기 창에서 IBM MQ .NET Client App (.NET Core) 선택하고 다음을 클릭합니다.
4. 새 프로젝트 구성 창에서 원하는 경우 프로젝트의 프로젝트 이름을 변경하고 작성을 클릭하여 .NET 프로젝트를 작성하십시오.

`MQDotnetApp.cs`은(는) 프로젝트 파일과 함께 작성되는 파일입니다. 이 파일은 큐 관리자에 연결되는 코드를 포함하여, 넣기 및 가져오기 조작을 수행합니다.

연결 속성은 기본값으로 설정됩니다.

- `MQC.CONNECTION_NAME_PROPERTY`는 `localhost(1414)`로 설정됨
- `MQC.CHANNEL_PROPERTY`는 `DOTNET.SVRCONN`으로 설정됨

큐는 `Q1`로 설정되며, 이 속성을 적절히 수정할 수 있습니다.

5. 애플리케이션을 컴파일하고 실행하십시오.

관련 개념

[IBM MQ 컴포넌트 및 기능](#)

[.NET 애플리케이션 런타임 - Windows만 해당](#)

IBM MQ classes for .NET의 구성 파일

.NET 클라이언트 애플리케이션은 IBM MQ MQI client 구성 파일을 사용할 수 있고, 관리 연결 유형을 사용하는 경우 .NET 애플리케이션 구성 파일을 사용할 수 있습니다. 애플리케이션 구성 파일의 설정은 우선순위가 있습니다.

클라이언트 구성 파일

IBM MQ classes for .NET 클라이언트 애플리케이션은 다른 모든 IBM MQ MQI client와 동일한 방식으로 클라이언트 구성 파일을 사용할 수 있습니다. 이 파일은 일반적으로 mqclient.ini이지만 다른 파일 이름을 지정할 수 있습니다. 클라이언트 구성 파일에 대한 자세한 내용은 [을 참조하십시오 IBM MQ MQI client 구성 파일, mqclient.ini.](#)

IBM MQ MQI client 구성 파일의 다음 속성만 IBM MQ classes for .NET와 관련됩니다. 다른 속성을 지정하는 경우 영향을 미치지 않습니다.

표 78. IBM MQ classes for .NET와 관련된 클라이언트 구성 파일 속성	
스탠자	속성
채널	CCSID
채널	ChannelDefinitionDirectory
채널	ChannelDefinitionFile
채널	ReconDelay
채널	DefRecon
채널	MQReconnectTimeout
채널	ServerConnectionParms
채널	Put1DefaultAlwaysSync
채널	PasswordProtection
ClientExitPath	ExitsDefaultPath
ClientExitPath	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage
보안	MQIInitialKeyFile
SSL	SSLKeyRepository
SSL	SSLKeyRepositoryPassword
TCP	ClnRcvBufSize
TCP	ClnSndBufSize
TCP	IPAddressVersion
TCP	KeepAlive

적절한 환경 변수를 사용하여 이러한 속성을 대체할 수 있습니다.

애플리케이션 구성 파일

관리형 연결 유형으로 실행 중인 경우, 애플리케이션 구성 .NET 파일을 사용하여 클라이언트 구성 IBM MQ 파일과 해당 환경 변수를 재정의할 수도 있습니다.

.NET 애플리케이션 구성 파일 설정은 관리 연결 유형과 실행할 때만 작동하며 다른 연결 유형에는 무시됩니다.

애플리케이션 .NET 구성 파일과 그 형식은 프레임워크 .NET 내에서 일반적인 사용을 위해 Microsoft 정의되지만, 본 문서에서 언급된 특정 IBM MQ 섹션 이름, 키 및 값은 예 특화된 것입니다.

.NET 애플리케이션 구성 파일의 형식은 섹션 수입니다. 각 섹션에는 하나 이상의 키가 있으며 각 키에는 연관된 값이 있습니다. 다음 예에서는 TCP/IP KeepAlive 특성을 제어하기 위해 .NET 애플리케이션 구성 파일에 사용된 섹션, 키 및 값을 보여줍니다.

```
<configuration>
  <configSections>
    <section name="TCP" type="System.Configuration.NameValueSectionHandler"/>
  </configSections>
  <TCP>
    <add key="KeepAlive" value="true"/></add>
  </TCP>
</configuration>
```

.NET 애플리케이션 구성 파일 섹션 이름과 키에서 사용한 키워드는 클라이언트 구성 파일에 정의된 스탠자 및 속성의 키워드와 정확히 일치합니다.

<configSections> 섹션은 <configuration> 요소의 첫 번째 하위 요소여야 합니다.

자세한 정보는 Microsoft 문서를 참조하십시오.

.NET에서 사용할 C# 코드 단편 예

애플리케이션이 큐 관리자에 연결하여 큐에 메시지를 넣고 응답을 받는 과정을 보여주는 C# 코드 단편입니다.

다음 C# 코드 단편은 세 개의 조치를 수행하는 애플리케이션을 보여줍니다.

1. 큐 관리자에 연결
2. SYSTEM.DEFAULT.LOCAL.QUEUE에 메시지 넣기
3. 다시 메시지 가져오기

연결 유형을 변경하는 방법도 보여줍니다.

```
// =====
// Licensed Materials - Property of IBM
// 5724-H72
// (c) Copyright IBM Corp. 2003, 2026
// =====
using System;
using System.Collections;

using IBM.WMQ;

class MQSample
{
    // The type of connection to use, this can be:-
    // MQC.TRANSPORT_MQSERIES_BINDINGS for a server connection.
    // MQC.TRANSPORT_MQSERIES_CLIENT for a non-XA client connection
    // MQC.TRANSPORT_MQSERIES_XAClient for an XA client connection
    // MQC.TRANSPORT_MQSERIES_MANAGED for a managed client connection
    const String connectionType = MQC.TRANSPORT_MQSERIES_CLIENT;

    // Define the name of the queue manager to use (applies to all connections)
    const String qManager = "your_Q_manager";

    // Define the name of your host connection (applies to client connections only)
    const String hostName = "your_hostname";

    // Define the name of the channel to use (applies to client connections only)
    const String channel = "your_channelname";

    /// <summary>
    /// Initialise the connection properties for the connection type requested
    /// </summary>
    /// <param name="connectionType">One of the MQC.TRANSPORT_MQSERIES_ values</param>
    static Hashtable init(String connectionType)
    {
        Hashtable connectionProperties = new Hashtable();

        // Add the connection type
        connectionProperties.Add(MQC.TRANSPORT_PROPERTY, connectionType);

        // Set up the rest of the connection properties, based on the
        // connection type requested
    }
}
```

```

switch(connectionType)
{
    case MQC.TRANSPORT_MQSERIES_BINDINGS:
        break;
    case MQC.TRANSPORT_MQSERIES_CLIENT:
    case MQC.TRANSPORT_MQSERIES_XACLIENT:
    case MQC.TRANSPORT_MQSERIES_MANAGED:
        connectionProperties.Add(MQC.HOST_NAME_PROPERTY, hostName);
        connectionProperties.Add(MQC.CHANNEL_PROPERTY, channel);
        break;
}

return connectionProperties;
}
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static int Main(string[] args)
{
    try
    {
        Hashtable connectionProperties = init(connectionType);

        // Create a connection to the queue manager using the connection
        // properties just defined
        MQQueueManager qMgr = new MQQueueManager(qManager, connectionProperties);

        // Set up the options on the queue we want to open
        int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;

        // Now specify the queue that we want to open, and the open options
        MQQueue system_default_local_queue =
            qMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", openOptions);

        // Define an IBM MQ message, writing some text in UTF format
        MQMessage hello_world = new MQMessage();
        hello_world.WriteUTF("Hello World!");

        // Specify the message options
        MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the defaults,
                                                             // same as MQPMO_DEFAULT

        // Put the message on the queue
        system_default_local_queue.Put(hello_world, pmo);

        // Get the message back again

        // First define an IBM MQ message buffer to receive the message
        MQMessage retrievedMessage = new MQMessage();
        retrievedMessage.MessageId = hello_world.MessageId;

        // Set the get message options
        MQGetMessageOptions gmo = new MQGetMessageOptions(); //accept the defaults
                                                             //same as MQGMO_DEFAULT

        // Get the message off the queue
        system_default_local_queue.Get(retrievedMessage, gmo);

        // Prove we have the message by displaying the UTF message text
        String msgText = retrievedMessage.ReadUTF();
        Console.WriteLine("The message is: {0}", msgText);

        // Close the queue
        system_default_local_queue.Close();

        // Disconnect from the queue manager
        qMgr.Disconnect();
    }

    //If an error has occurred, try to identify what went wrong.

    //Was it an IBM MQ error?
    catch (MQException ex)
    {
        Console.WriteLine("An IBM MQ error occurred: {0}", ex.ToString());
    }

    catch (System.Exception ex)
    {

```

```

        Console.WriteLine("A System error occurred: {0}", ex.ToString());
    }

    return 0;
} //end of start
} //end of sample

```

IBM MQ 환경 설정

클라이언트 연결을 사용하여 큐 관리자에 연결하기 전에 IBM MQ 환경을 설정해야 합니다.

참고: 서버 바인딩 모드에서 IBM MQ classes for .NET를 사용할 때 이 단계는 필요하지 않습니다.

.NET 프로그래밍 인터페이스를 사용하면 NMQ_MQ_LIB 사용자 정의 값을 사용할 수 있지만 MQEnvironment 클래스도 포함되어 있습니다. 이 클래스를 사용하면 연결 중에 사용할 세부 사항(예: 다음 목록에 지정된 사항)을 지정할 수 있습니다.

- 채널 이름
- 호스트 이름
- 포트 번호
- 채널 엑시트
- SSL 매개변수
- 사용자 ID 및 비밀번호

MQEnvironment 클래스에 대한 전체 정보는 [MQEnvironment.NET 클래스](#) 를 참조하십시오.

채널 이름과 호스트 이름을 지정하려면 다음 코드를 사용하십시오.

```

MQEnvironment.Hostname = "host.domain.com";
MQEnvironment.Channel = "client.channel";

```

기본적으로 클라이언트가 1414 포트에서 IBM MQ 리스너에 연결을 시도합니다. 다른 포트를 지정하려면 다음 코드를 사용하십시오.

```

MQEnvironment.Port = nnnn;

```

큐 관리자에 연결 및 큐 관리자에서 연결 끊기

IBM MQ 환경을 구성한 경우 큐 관리자에 연결할 준비가 됩니다.

큐 관리자에 연결하려면 MQQueueManager 클래스의 새 인스턴스를 작성하십시오.

```

MQQueueManager queueManager = new MQQueueManager("qMgrName");

```

큐 관리자에서 연결을 끊으려면 다음과 같이 큐 관리자에서 Disconnect 메소드를 호출하십시오.

```

queueManager.Disconnect();

```

큐 관리자에 연결을 시도할 때 큐 관리자에 대한 조회(inq) 권한이 있어야 합니다. 조회 권한이 없으면 연결에 실패합니다.

Disconnect 메소드를 호출하는 경우 큐 관리자를 통해 액세스한 열린 모든 큐와 프로세스가 닫힙니다. 그러나 프로그래밍 시 이러한 자원의 사용을 완료한 후 명시적으로 닫는 것이 좋습니다. 자원을 닫으려면 각 자원과 연관된 오브젝트에서 Close 메소드를 사용하십시오.

큐 관리자의 Commit 및 Backout 메소드는 프로시저에 따른 인터페이스와 함께 사용하는 MQCMIT 및 MQBACK 호출을 대체합니다.

큐와 토픽에 액세스

MQQueueManager 메소드 또는 적절한 구성자를 사용하여 큐와 토픽에 액세스할 수 있습니다.

큐에 액세스하려면 MQQueueManager 클래스의 메소드를 사용하십시오. MQOD(오브젝트 디스크립터 구조)는 이러한 메소드의 매개변수로 구분됩니다. 예를 들어 queueManager라는 MQQueueManager 오브젝트로 표시된 큐 관리자에서 큐를 열려면 다음 코드를 사용하십시오.

```
MQQueue queue = queueManager.AccessQueue("qName",
                                           MQC.MQOO_OUTPUT,
                                           "qMgrName",
                                           "dynamicQName",
                                           "altUserId");
```

options 매개변수는 MQOPEN 호출의 Options 매개변수와 동일합니다.

AccessQueue 메소드는 MQQueue 클래스의 새 오브젝트를 리턴합니다.

큐 사용이 완료되면 다음 예에서와 같이 Close() 메소드를 사용하여 닫으십시오.

```
queue.Close();
```

IBM MQ .NET에서 MQQueue 구성자를 사용하여 큐도 작성할 수 있습니다. 매개변수는 accessQueue 메소드와 똑같으며, 사용할 인스턴스화된 MQQueueManager 오브젝트를 지정하는 큐 관리자 매개변수가 추가됩니다. 예를 들면, 다음과 같습니다.

```
MQQueue queue = new MQQueue(queueManager,
                              "qName",
                              MQC.MQOO_OUTPUT,
                              "qMgrName",
                              "dynamicQName",
                              "altUserId");
```

이 방식으로 큐 오브젝트를 구성하면 MQQueue의 고유 서브클래스를 작성할 수 있습니다.

마찬가지로 MQQueueManager 클래스의 메소드를 사용하여 토픽에도 액세스할 수 있습니다. AccessTopic() 메소드를 사용하여 토픽을 여십시오. 그러면 MQTopic의 새 오브젝트가 리턴됩니다. 토픽 사용이 완료되면 MQTopic의 Close() 메소드를 사용하여 닫으십시오.

MQTopic 구성자를 사용하여 토픽도 작성할 수 있습니다. 토픽의 구성자는 여러 개가 있습니다. 자세한 정보는 [MQTopic.NET 클래스](#)를 참조하십시오.

메시지 핸들링

메시지는 큐 또는 토픽 클래스의 메소드를 사용하여 핸들링합니다. 새 메시지를 빌드하려면 새 MQMessageobject를 작성하십시오.

MQQueue 또는 MQTopic 클래스의 Put() 메소드를 사용하여 큐나 토픽에 메시지를 넣습니다. MQQueue 또는 MQTopic 클래스의 Get() 메소드를 사용하여 큐나 토픽에서 메시지를 가져옵니다. MQPUT과 MQGET에서 바이트 배열을 넣고 가져오는 프로시저 인터페이스와 달리 IBM MQ classes for .NET는 MQMessage 클래스의 인스턴스를 넣고 가져옵니다. MQMessage 클래스는 실제 메시지 데이터를 포함하는 데이터 버퍼와 해당 메시지를 설명하는 모든 MQMD(메시지 디스크립터) 매개변수를 함께 캡슐화합니다.

새 메시지를 빌드하려면 MQMessage 클래스의 새 인스턴스를 작성하고 WriteXXX 메소드를 사용하여 메시지 버퍼에 데이터를 넣으십시오.

새 메시지 인스턴스를 작성하면 MQMD의 초기값 및 언어 선언에 설명된 대로 모든 MQMD 매개변수가 자동으로 기본값으로 설정됩니다. MQQueue의 Put() 메소드는 MQPutMessageOptions 클래스의 인스턴스도 매개변수로 받습니다. 이 클래스는 MQPMO 구조를 나타냅니다. 다음 예는 메시지를 작성하여 큐에 넣습니다.

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.WriteInt(25);

String name = "Charlie Jordan";
myMessage.WriteUTF(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message
!queue.Put(myMessage, pmo);
```


MQQueue의 Get() 메소드는 큐에서 방금 받은 메시지를 나타내는 MQMessage의 새 인스턴스를 리턴합니다. 또한 MQGetMessageOptions 클래스의 인스턴스도 매개변수로 받습니다. 이 클래스는 MQGMO 구조를 나타냅니다.

수신 메시지에 맞게 Get() 메소드가 내부 버퍼의 크기를 자동으로 조정하므로 최대 메시지 크기를 지정하지 않아도 됩니다. MQMessage 클래스의 ReadXXX 메소드를 사용하여 리턴된 메시지의 데이터에 액세스하십시오.

다음 예는 큐에서 메시지를 가져오는 방법을 보여줍니다.

```
// Get a message from the queue
MQMessage theMessage = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.Get(theMessage,gmo); // has default values

// Extract the message data
int age = theMessage.ReadInt();
String name1 = theMessage.ReadUTF();
```

encoding 멤버 변수를 설정하여 읽기 및 쓰기 메소드에서 사용하는 번호 형식을 대체할 수 있습니다.

characterSet 멤버 변수를 설정하여 문자열을 읽고 쓰는 데 사용하는 문자 세트를 대체할 수 있습니다.

자세한 정보는 [MQMessage.NET](#) 클래스를 참조하십시오.

참고: MQMessage의 WriteUTF() 메소드는 포함된 유니코드 바이트 외에도 문자열 길이를 자동으로 인코딩합니다. 다른 .NET 프로그램에서 메시지를 읽을 때(readUTF() 사용) 이 방법을 사용하여 가장 간단하게 문자열 정보를 송신할 수 있습니다.

메시지 특성 핸들링

메시지 특성을 사용하면 메시지를 선택하거나, 헤더에 액세스하지 않고 메시지에 대한 정보를 검색할 수 있습니다. MQMessage 클래스는 특성을 가져오고 설정하는 메소드를 포함합니다.

애플리케이션이 처리할 메시지를 선택하거나 MQMD 또는 MQRFH2 헤더에 액세스하지 않고 메시지에 대한 정보를 검색할 수 있도록 메시지 특성을 사용할 수 있습니다. 또한 IBM MQ 애플리케이션과 JMS 애플리케이션 사이의 통신을 용이하게 합니다. IBM MQ의 메시지 특성에 대한 자세한 정보는 [메시지 특성](#)을 참조하십시오.

MQMessage 클래스는 특성의 데이터 유형에 따라 특성을 가져오고 설정하는 여러 메소드를 제공합니다. get 메소드에는 Get*Property 형식의 이름이 있고 set 메소드에는 Set*Property 형식의 이름이 있습니다. 여기서 별표(*)는 다음 문자열 중 하나를 표시합니다.

- 부울
- Byte
- 바이트
- Double
- Float
- Int
- Int2
- Int4
- Int8
- Long
- 오브젝트
- 짧음
- 문자열

예를 들어 IBM MQ 특성 myproperty(문자열)를 가져오려면

message.GetStringProperty('myproperty') 호출을 사용하십시오. IBM MQ에서 완료할 특성 디스크립터를 선택적으로 전달할 수 있습니다.

오류 핸들링

'IBM MQ classes for .NET'에서 발생하는 오류는 'try' 및 'catch' 블록을 사용하여 처리합니다.

.NET 인터페이스의 메소드는 완료 코드와 이유 코드를 리턴하지 않습니다. 대신 IBM MQ 호출을 통해 생성된 완료 코드와 이유 코드가 둘 다 0이 아닌 경우 예외 처리를 합니다. 그러면 IBM MQ를 각각 호출한 후에 리턴 코드를 확인할 필요가 없도록 프로그램 로직이 간소화됩니다. 프로그램에서 실패의 가능성을 처리할 지점을 결정할 수 있습니다. 다음 예와 같이 이러한 지점의 코드를 try와 catch 블록으로 묶습니다.

```
try
{
    myQueue.Put(messageA,PutMessageOptionsA);
    myQueue.Put(messageB,PutMessageOptionsB);
}
catch (MQException ex)
{
    // This block of code is only executed if one of
    // the two put methods gave rise to a non-zero
    // completion code or reason code.
    Console.WriteLine("An error occurred during the put operation:" +
        "CC = " + ex.CompletionCode +
        "RC = " + ex.ReasonCode);
    Console.WriteLine("Cause exception:" + ex );
}
```

속성 값 가져오기 및 설정

MQManagedObject, MQQueue 및 MQQueueManager 클래스에는 속성 값을 가져오고 설정할 수 있는 메소드가 포함되어 있습니다. MQQueue의 경우 큐를 열 때 적절한 inquire 및 set 플래그를 지정하는 경우에만 메소드가 작동합니다.

공통 속성의 경우 MQQueueManager와 MQQueue 클래스는 MQManagedObject라는 클래스에서 상속합니다. 이 클래스는 Inquire() 및 Set() 인터페이스를 정의합니다.

new 연산자를 사용하여 새 큐 관리자를 작성하면 조회를 위해 자동으로 열립니다. AccessQueue() 메소드를 사용하여 큐 오브젝트에 액세스하면 해당 오브젝트가 조회나 설정 조작을 위해 자동으로 열리지 않으므로 일부 리모트 큐 유형에 문제를 초래할 수 있습니다. 큐에서 Inquire 및 Set 메소드를 사용하고 특성을 설정하려면 AccessQueue() 메소드의 openOptions 매개변수에 적절한 inquire 및 set 플래그를 지정해야 합니다.

inquire 및 set 메소드는 다음 세 개의 매개변수를 사용합니다.

- selectors 배열
- intAttrs 배열
- charAttrs 배열

배열 길이는 항상 공개되므로 MQINQ에 있는 SelectorCount, IntAttrCount 및 CharAttrLength 매개변수가 필요하지 않습니다. 다음 예는 큐에서 조회하는 방법을 보여줍니다.

```
//inquire on a queue
int [ ] selectors = new int [2] ;
int [ ] intAttrs = new int [1] ;
byte [ ] charAttrs = new byte [MQC.MQ_Q_DESC_LENGTH];
selectors [0] = MQC.MQIA_DEF_PRIORITY;
selectors [1] = MQC.MQCA_Q_DESC;
queue.Inquire(selectors,intAttrs,charAttrs);
ASCIIEncoding enc = new ASCIIEncoding();
String s1 = "";
s1 = enc.GetString(charAttrs);
```

이러한 오브젝트의 모든 속성을 조회할 수 있습니다. 속성의 서브세트는 오브젝트의 특성으로 공개됩니다. 오브젝트 속성 목록은 [오브젝트 속성](#)을 참조하십시오. 오브젝트 특성은 적절한 클래스 설명을 참조하십시오.

멀티스레드 프로그램

.NET 런타임 환경은 본질적으로 멀티스레드입니다. IBM MQ classes for .NET를 사용하면 여러 스레드에서 큐 관리자 오브젝트를 공유할 수 있지만 대상 큐 관리자에 대한 모든 액세스를 동기화되는지 확인하십시오.

큐 관리자에 연결하고 시동 시 큐를 여는 단순 프로그램을 고려하십시오. 프로그램이 화면에 단일 단추를 표시합니다. 사용자가 해당 단추를 클릭하면 프로그램이 큐에서 메시지를 폐치합니다. 이 경우 애플리케이션 초기화가 한 스레드에서 수행되고, 단추 누르기에 응답하여 실행되는 코드는 별도 스레드(사용자 인터페이스 스레드)에서 실행됩니다.

IBM MQ .NET를 구현하면 특정 연결(MQQueueManager 오브젝트 인스턴스)에서 대상 IBM MQ 큐 관리자에 대한 모든 액세스가 동기화됩니다. 기본 동작은 해당 연결을 위해 진행 중인 다른 모든 호출이 완료될 때까지 큐 관리자에 대한 호출을 실행할 스레드가 차단됩니다. 프로그램의 여러 스레드에서 동일한 큐 관리자에 동시에 액세스해야 하는 경우 동기 액세스해야 하는 각 스레드의 새 MQQueueManager 오브젝트를 작성하십시오 (각 스레드에 개별 MQCONN 호출을 실행하는 것과 동등).

MQC.MQCNO_HANDLE_SHARE_NONE 또는 MQC.MQCNO_SHARE_NO_BLOCK으로 기본 연결 옵션을 대체하면 큐 관리자가 더 이상 동기화되지 않습니다.

.NET에서 클라이언트 채널 정의 테이블 사용

클라이언트 채널 정의 테이블(CCDT)을 사용할 수 IBM MQ classes for .NET 있습니다. 사용 중인 연결(관리 또는 비관리)에 따라 여러 다른 방식으로 CCDT의 위치를 지정합니다.

비XA 또는 XA 관리되지 않은 클라이언트 연결 유형

비관리 연결 유형을 사용하여 다음 두 방법으로 CCDT의 위치를 지정할 수 있습니다.

- 환경 변수 MQCHLLIB를 사용하여 테이블이 위치한 디렉토리를 지정하고, MQCHLTAB를 사용하여 테이블의 파일 이름을 지정합니다.
- 클라이언트 구성 파일 사용. CHANNELS 스탠자에서, 테이블이 위치한 디렉토리를 지정하려면 **ChannelDefinitionDirectory** 속성을 사용하고, 파일 이름을 **ChannelDefinitionFile** 지정하려면 속성을 사용하십시오.

위치가 클라이언트 구성 파일에 지정되고 동시에 환경 변수를 사용하여 지정된 경우, 환경 변수가 우선순위를 가집니다. 이 기능을 사용하여 클라이언트 구성 파일에 표준 위치를 지정하고 필요하면 환경 변수를 사용하여 이를 대체할 수 있습니다.

관리 클라이언트 연결 유형

관리 연결 유형을 사용하여 다음 세 방식으로 CCDT의 위치를 지정할 수 있습니다.

- .NET 애플리케이션 구성 파일 사용. CHANNELS 섹션에서 키를 사용하여 테이블이 **ChannelDefinitionDirectory** 위치한 디렉토리를 지정하고, 키를 **ChannelDefinitionFile** 사용하여 파일 이름을 지정합니다.
- 환경 변수 MQCHLLIB를 사용하여 테이블이 위치한 디렉토리를 지정하고, MQCHLTAB를 사용하여 테이블의 파일 이름을 지정합니다.
- 클라이언트 구성 파일 사용. CHANNELS 스탠자에서, 테이블이 위치한 디렉토리를 지정하려면 **ChannelDefinitionDirectory** 속성을 사용하고, 파일 이름을 **ChannelDefinitionFile** 지정하려면 속성을 사용하십시오.

두 가지 이상의 이러한 방법으로 위치를 지정하면 환경 변수가 클라이언트 구성 파일보다 우선하며 .NET 애플리케이션 구성 파일이 다른 방법보다 우선합니다. 이 기능을 사용하여 클라이언트 구성 파일에 표준 위치를 지정하고 필요하면 환경 변수 또는 애플리케이션 구성 파일을 사용하여 이를 대체할 수 있습니다.

클라이언트는 .NET C 및 Java 클라이언트와 동일한 IBM MQ 9.3.0 방식으로 동작하며, CCDT를 사용하여 큐 관리자 그룹핑을 할 때 MQRC_Q_MGR_NAME_ERROR를 반환합니다.

.NET 애플리케이션을 통해 사용할 채널 정의를 판별하는 방법

IBM MQ .NET 클라이언트 환경에서 사용할 채널 정의는 여러 다른 방식으로 지정할 수 있습니다. 채널 정의의 스펙은 여러가지가 있을 수 있습니다. 애플리케이션은 하나 이상의 소스에서 채널 정의를 도출합니다.

채널 정의가 두 개 이상 있으면 다음 우선순위에 따라 사용되는 정의를 선택합니다.

1. 특성 해시 테이블에 *MQC.CHANNEL_PROPERTY*를 포함하거나 명시적으로 *MQQueueManager* 구성자에 지정된 특성
2. *MQEnvironment.properties* 해시 테이블의 *MQC.CHANNEL_PROPERTY* 특성
3. *MQEnvironment*의 *Channel* 특성
4. .NET 애플리케이션 구성 파일, 섹션 이름 CHANNELS, 키 *ServerConnectionParms*(관리 연결에만 적용)
5. *MQSERVER* 환경 변수
6. 클라이언트 구성 파일, 스탠자 CHANNELS, Attribute *ServerConnectionParms*
7. 클라이언트 채널 정의 테이블(CCDT). CCDT의 위치는 .NET 애플리케이션 구성 파일에 지정됨(관리 연결에만 적용됨)
8. 클라이언트 채널 정의 테이블(CCDT). CCDT의 위치는 환경 변수 *MQCHLLIB* 및 *MQCHLTAB*을 사용하여 지정합니다
9. 클라이언트 채널 정의 테이블(CCDT). CCDT의 위치는 클라이언트 구성 파일을 사용하여 지정합니다.

항목 1-3의 경우 애플리케이션에서 제공한 값을 통해 필드별로 채널 정의를 빌드합니다. 이러한 값은 여러 다른 인터페이스를 사용하여 제공하며 각각에 값이 여러 개일 수 있습니다. 지정된 우선순위에 따라 채널 정의에 필드 값을 추가합니다.

1. *MQQueueManager* 구성자의 *connName* 값
 2. *MQQueueManager.properties* 해시 테이블의 특성 값
 3. *MQEnvironment.properties* 해시 테이블의 특성 값
 4. *MQEnvironment* 필드로 설정된 값(예: *MQEnvironment.Hostname*, *MQEnvironment.Port*)
- 4-6 항목의 경우 전체 채널 정의가 값으로 제공됩니다. 채널 정의에서 지정되지 않은 필드는 시스템 기본값을 사용합니다. 채널과 해당 필드를 정의하는 다른 메소드의 값은 이러한 스펙과 병합되지 않습니다.
- 7-9 항목의 경우 CCDT에서 전체 채널 정의를 가져옵니다. 채널을 정의할 때 명시적으로 지정되지 않은 필드는 시스템 기본값을 사용합니다. 채널과 해당 필드를 정의하는 다른 메소드의 값은 이러한 스펙과 병합되지 않습니다.

IBM MQ .NET의 채널 엑시트 사용

클라이언트 바인딩을 사용하는 경우 다른 모든 클라이언트 연결에 채널 엑시트를 사용할 수 있습니다. 관리 바인딩을 사용하는 경우 적절한 인터페이스를 구현하는 엑시트 프로그램을 작성해야 합니다.

클라이언트 바인딩

클라이언트 바인딩을 사용하는 경우 채널 엑시트에 설명된 대로 채널 엑시트를 사용할 수 있습니다. 관리 바인딩용으로 작성된 채널 엑시트를 사용할 수 없습니다.

관리 바인딩

관리 연결을 사용하여 엑시트를 구현하는 경우 적절한 인터페이스를 구현하는 새 .NET 클래스를 정의합니다. 다음 세 개의 엑시트 인터페이스가 IBM MQ 패키지에 정의되어 있습니다.

- *MQSendExit*
- *MQReceiveExit*
- *MQSecurityExit*

참고: 이러한 인터페이스를 사용하여 작성된 사용자 엑시트는 비관리 환경에서 채널 엑시트로 지원되지 않습니다.

다음 샘플은 세 인스턴스를 모두 구현하는 클래스를 정의합니다.

```
class MyMQExits : MQSendExit, MQReceiveExit, MQSecurityExit
{
    // This method comes from the send exit
    byte[] SendExit(MQChannelExit channelExitParms,
                    MQChannelDefinition channelDefinition,
                    byte[] dataBuffer,
                    ref int dataOffset)
```

```

        ref int          dataLength
        ref int          dataMaxLength)
    {
        // complete the body of the send exit here
    }

    // This method comes from the receive exit
    byte[] ReceiveExit(MQChannelExit channelExitParms,
                       MQChannelDefinition channelDefinition,
                       byte[] dataBuffer,
                       ref int dataOffset,
                       ref int dataLength,
                       ref int dataMaxLength)
    {
        // complete the body of the receive exit here
    }

    // This method comes from the security exit
    byte[] SecurityExit(MQChannelExit channelExitParms,
                       MQChannelDefinition channelDefParms,
                       byte[] dataBuffer,
                       ref int dataOffset,
                       ref int dataLength,
                       ref int dataMaxLength)
    {
        // complete the body of the security exit here
    }
}

```

각 엑시트에 MQChannelExit 및 MQChannelDefinition 오브젝트 인스턴스가 전달됩니다. 이러한 오브젝트는 절차적 인터페이스에 정의된 MQCXP 및 MQCD 구조를 표시합니다.

송신 엑시트가 송신할 데이터 및 보안 또는 수신 엑시트에서 수신할 데이터는 엑시트의 매개변수를 사용하여 지정합니다.

입력 시 *dataBuffer* 바이트 배열의 *dataOffset* 오프셋에 있는 길이가 *dataLength*인 데이터가 송신 엑시트에서 송신할 데이터이고 보안 또는 수신 엑시트에서 받은 데이터입니다. *dataMaxLength* 매개변수는 *dataBuffer*의 엑시트에서 사용할 수 있는 최대 길이(*dataOffset*에서)를 제공합니다. 참고: 보안 엑시트에서 처음으로 엑시트를 호출하거나 파트너 측에서 데이터를 송신하지 않도록 선택한 경우 *dataBuffer*는 널일 수 있습니다.

리턴 시, *dataOffset* 및 *dataLength* 값은 .NET에서 사용해야 하는 리턴된 바이트 배열에 포함된 오프셋과 길이를 가리키도록 설정해야 합니다. 송신 엑시트의 경우에는 송신해야 하는 데이터를 나타내고 보안 또는 수신 엑시트의 경우에는 해석해야 하는 데이터를 표시합니다. 일반적으로 엑시트는 바이트 배열을 리턴합니다. 단, 데이터를 송신하지 않도록 선택할 수 있는 보안 엑시트와 INIT 또는 TERM 이유로 호출된 엑시트는 예외입니다. 따라서 작성할 수 있는 가장 간단한 양식의 엑시트는 *dataBuffer*를 리턴하는 기능만 수행하는 엑시트입니다.

가장 간단한 엑시트 본문은 다음과 같습니다.

```

{
    return dataBuffer;
}

```

MQChannelDefinition 클래스

관리 .NET 클라이언트 애플리케이션으로 지정하는 사용자 ID 및 비밀번호는 클라이언트 보안 엑시트에 전달되는 IBM MQ .NET MQChannelDefinition 클래스에 설정됩니다. 보안 엑시트는 사용자 ID 및 비밀번호를 MQCD.RemoteUserIdentifier 및 MQCD.RemotePassword 필드에 복사합니다([721 페이지의 『보안 엑시트 작성』 참조](#)).

채널 엑시트 지정(관리 클라이언트)

MQQueueManager 오브젝트를 작성할 때(MQEnvironment 또는 MQQueueManager 구성자에서) 채널 이름과 연결 이름을 지정하는 경우 두 가지 방법으로 채널 엑시트를 지정할 수 있습니다.

우선순위 순으로 다음과 같습니다.

1. MQQueueManager 구성자에서 MQC.SECURITY_EXIT_PROPERTY, MQC.SEND_EXIT_PROPERTY 또는 MQC.RECEIVE_EXIT_PROPERTY 해시 테이블 특성 전달.

2. MQEnvironment SecurityExit, SendExit 또는 ReceiveExit 특성 설정.

채널 이름과 연결 이름을 지정하지 않으면 사용할 채널 엑시트는 클라이언트 채널 정의 테이블(CCDT)에서 수집한 채널 정의에서 가져옵니다. 채널 정의에 저장된 값을 대체할 수 없습니다. 채널 정의 테이블에 대한 자세한 정보는 [클라이언트 채널 정의 테이블 및 553 페이지의 『.NET에서 클라이언트 채널 정의 테이블 사용』](#)의 내용을 참조하십시오.

각 경우에 스펙은 다음 형식의 문자열 양식을 사용합니다.

```
Assembly_name(Class_name)
```

*Class_name*은 IBM.WMQ.MQSecurityExit, IBM.WMQ.MQSendExit 또는 IBM.WMQ.MQReceiveExit 인터페이스를 적절하게 구현하는 .NET 클래스의 완전한 이름입니다(네임스페이스 스펙 포함). *Assembly_name*은 클래스가 있는 어셈블리의 완전한 위치입니다(파일 확장자 포함). MQEnvironment 또는 MQQueueManager의 특성을 사용하는 경우 문자열의 길이는 999자로 제한됩니다. 그러나 채널 엑시트 이름이 CCDT에 지정된 경우 128자로 제한됩니다. 필요한 경우 .NET 클라이언트 코드가 문자열 스펙을 구문 분석하여 지정된 클래스의 인스턴스를 로드하고 작성합니다.

채널 엑시트 사용자 데이터 지정(관리 클라이언트)

채널 엑시트에는 사용자 데이터가 연관될 수 있습니다. MQQueueManager 오브젝트를 작성할 때 (MQEnvironment 또는 MQQueueManager 구성자에서) 채널 이름과 연결 이름을 지정하는 경우 두 가지 방법으로 사용자 데이터를 지정할 수 있습니다.

우선순위 순으로 다음과 같습니다.

1. MQQueueManager 구성자에서 MQC.SECURITY_USERDATA_PROPERTY, MQC.SEND_USERDATA_PROPERTY 또는 MQC.RECEIVE_USERDATA_PROPERTY 해시 테이블 특성 전달.
2. MQEnvironment SecurityUserData, SendUserData 또는 ReceiveUserData 특성 설정.

채널 이름과 연결 이름을 지정하지 않으면 사용할 엑시트 사용자 데이터 값은 클라이언트 채널 정의 테이블(CCDT)에서 수집한 채널 정의에서 가져옵니다. 채널 정의에 저장된 값을 대체할 수 없습니다. 채널 정의 테이블에 대한 자세한 정보는 [클라이언트 채널 정의 테이블 및 553 페이지의 『.NET에서 클라이언트 채널 정의 테이블 사용』](#)의 내용을 참조하십시오.

각각의 경우 스펙은 32자로 제한되는 문자열입니다.

.NET에서 자동 클라이언트 다시 연결

예상치 못한 연결 중단 시 클라이언트가 자동으로 큐 관리자에 다시 연결하게 할 수 있습니다.

예를 들어 큐 관리자가 중지하거나 네트워크 또는 서버가 실패하면 예상치 못하게 큐 관리자에서 클라이언트의 연결이 끊깁니다.

자동 클라이언트 다시 연결을 사용하지 않으면 연결 실패 시 오류가 생성됩니다. 오류 코드를 사용하여 연결을 재설정할 수 있습니다.

자동 클라이언트 다시 연결 기능을 사용하는 클라이언트는 다시 연결 가능한 클라이언트라고 합니다. 다시 연결 가능한 클라이언트를 작성하려면 큐 관리자에 연결하는 동안 다시 연결 옵션이라는 특정 옵션을 지정하십시오.

클라이언트 애플리케이션이 IBM MQ .NET 클라이언트이면 MQQueueManager 클래스를 사용하여 큐 관리자를 작성할 때 CONNECT_OPTIONS_PROPERTY의 적절한 값을 지정하여 자동으로 클라이언트를 다시 연결하게 선택할 수 있습니다. CONNECT_OPTIONS_PROPERTY 값의 자세한 내용은 [다시 연결 옵션](#)을 참조하십시오.

클라이언트 애플리케이션이 항상 연결하고 다시 연결하는 대상(동일한 이름의 큐 관리자, 동일한 큐 관리자 또는 클라이언트 연결 테이블에 동일한 QMNAME으로 정의된 큐 관리자 세트)을 선택할 수 있습니다(자세한 내용은 [CCDT의 큐 관리자 그룹](#) 참조).

TLS (TLS) 지원 활성화 .NET

IBM MQ classes for .NET 클라이언트 애플리케이션은 TLS(Transport Layer Security) 암호화를 지원합니다. TLS 프로토콜은 인터넷 상에서 통신 보안을 제공하며, 클라이언트/서버 애플리케이션이 기밀성과 신뢰성을 갖춘 방식으로 통신할 수 있도록 합니다.

이 태스크 정보

참고:  SSL3.0, TLS 및 1.0 프로토콜은 부터 더 이상 IBM MQ.NET IBM MQ 10.0.0 사용되지 않습니다. MQEnvironment의 `MQC.SSL\_CIPHER\_SPEC\_PROPERTY` 또는 `SSLCipherSpec` 속성 아래에 정의된 Cipherspecs 중 `SSL`, `3.0`, `TLS` 또는 `1.0` 프로토콜을 사용하는 것은 보안 취약점으로 인해 사용을 자제해야 하며, 향후 릴리스에서 지원이 중단될 예정입니다. 이 두 프로토콜 중 하나를 사용할 경우 클라이언트 애플리케이션의 오류 로그에 경고 메시지가 기록되며, TLS 1.2 프로토콜 이상을 사용할 것을 권장합니다.

IBM MQ.NET 완전 관리 클라이언트는 Microsoft.NET SSLStreams 킷을 기반으로 하는 TLS(Transport Layer Security) 지원을 제공합니다. 이는 다른 IBM MQ 클라이언트들과는 다릅니다. 다른 클라이언트들은 IBM Global Security Kit (GSKit)을 기반으로 합니다.

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 래퍼 클래스로 구성됩니다.

관리 IBM MQ.NET 클라이언트는 Microsoft.NET Framework 라이브러리를 사용하여 TLS 보안 소켓 프로토콜을 구현합니다. System.NET.Security.SSLStream 클래스는 .NET IBM MQ.NET 프레임워크에서 보안 (TLSMicrosoft)을 구현하는 데 사용됩니다.

비관리형 IBM MQ.NET 클라이언트 모드는 이미 C MQI (및 GSKit)을 기반으로 하는 TLS 기능을 지원합니다. 즉, TLS 조작은 C MQI에서 처리합니다. 이 경우, GSKit TLS 보안 소켓 프로토콜을 구현합니다.

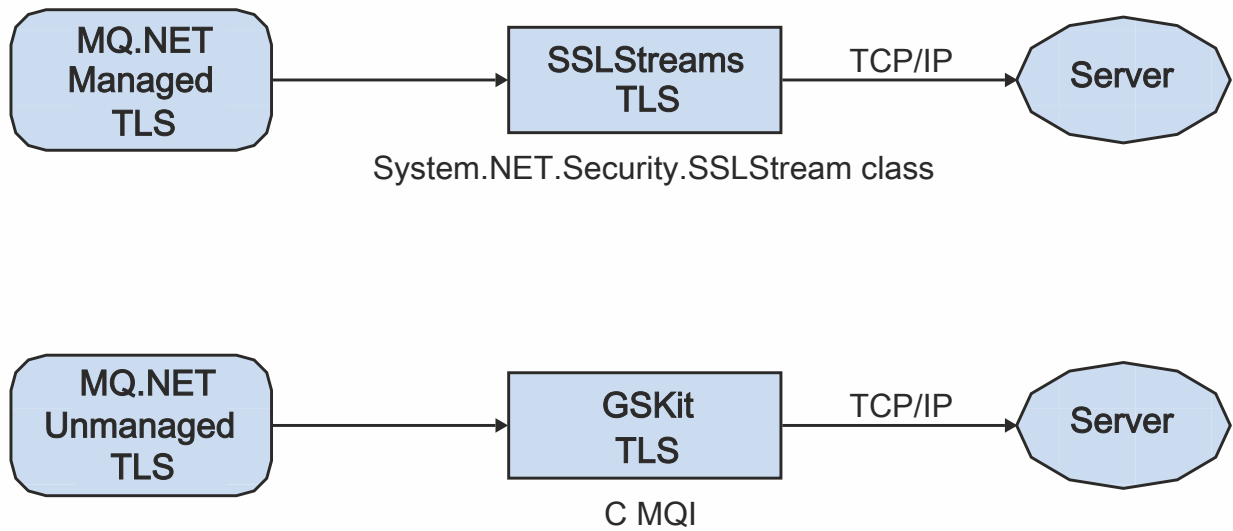


그림 67. IBM MQ.NET 관리 및 비관리 TLS 비교

다음 표는 관리와 비관리 구현 사이의 차이점을 요약합니다.

표 79. 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 그리고 C- MQI	TLS 보안 소켓 프로토콜

참고: [v 10.0.0](#) TLS 인증서의 사용자 지정 경로를 지정하고 애플리케이션에서 .NET 사용하는 IBM MQ 9.4.4 인증서의 암호를 안전하게 암호화할 수 있습니다. 인증서를 검색하고 애플리케이션 실행 시 TLS 핸드셰이크에 이 사용자 지정 경로를 사용할 수 있습니다. 자세한 내용은 일반 텍스트 또는 암호화된 암호를 사용하는 TLS 연결을 위한 .NET의 인증서 사용자 지정 경로를 참조하십시오.

관련 개념

암호화 보안 프로토콜: TLS

비관리형 .NET 클라이언트에 대한 TLS 지원 활성화

TLS 비관리형 .NET 클라이언트에 대한 지원은 C++ MQI와 IBM Global Security Kit (GSKit).NET을 기반으로 합니다. C MQI는 TLS의 작업을 처리하며, GSKit TLS 보안 소켓 프로토콜을 구현합니다.

이 태스크 정보

TLS는 클라이언트 연결에만 지원됩니다. TLS를 사용하려면 큐 관리자와 통신할 때 사용할 CipherSpec을 지정해야 하며, 이 CipherSpec은 대상 채널에 설정된 CipherSpec과 일치해야 합니다.

큐 관리자는 식별 이름(DN)을 포함하는 TLS 인증서를 사용하여 식별됩니다. IBM MQ .NET 클라이언트 애플리케이션은 이 DN을 사용하여 올바른 큐 관리자와 통신 중인지 확인할 수 있습니다.

TLS를 사용하여 큐 관리자에 연결할 때 IBM MQ classes for .NET에서 다음 이유 코드가 발행될 수 있습니다.

MQRC_SSL_NOT_ALLOWED

SSLCipherSpec 특성이 설정되어 있지만 바인딩 연결이 사용되었습니다. 클라이언트 연결에서만 TLS를 지원합니다.

MQRC_SSL_PEER_NAME_MISMATCH

SSLPeerName 특성에 지정된 DN 패턴이 큐 관리자에서 제공하는 DN과 일치하지 않습니다.

MQRC_SSL_PEER_NAME_ERROR

SSLPeerName 특성에 지정된 DN 패턴이 올바르지 않습니다.

MQRC_KEY_REPOSITORY_ERROR

키 저장소의 위치가 지정되지 않았거나, 유효하지 않거나, 접근할 수 없습니다.

프로시저

- TLS를 사용하려면 MQEnvironment의 SSLCipherSpec 정적 멤버 변수를 사용하여 CipherSpec을 지정하십시오.

다음 예는 CipherSpec이 TLS_RSA_WITH_AES_128_CBC_SHA인 TLS가 필요하도록 설정된 SECURE.SVRCONN.CHANNEL이라는 SVRCONN 채널에 첨부합니다.

```
MQEnvironment.Hostname = "your_hostname";
MQEnvironment.Channel = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "C:\mqm\key.kdb";
MQQueueManager qmgr = new MQQueueManager("your_q_manager");
```

CipherSpec 목록은 [CipherSpec 지정](#)을 참조하십시오.

SSLCipherSpec 특성은 연결 특성의 해시 테이블에서 MQC.SSL_CIPHER_SPEC_PROPERTY도 사용하여 설정할 수 있습니다.

TLS를 사용하여 성공적으로 연결하려면 큐 관리자가 제공한 인증서를 인증할 수 있는 인증 기관 루트 인증서 체인으로 클라이언트 키 저장소를 설정해야 합니다. 마찬가지로 SVRCONN 채널의 SSLClientAuth가 MQSSL_CLIENT_AUTH_REQUIRED로 설정된 경우 클라이언트 키 저장소에는 큐 관리자가 신뢰하는 식별 인증서가 있어야 합니다.

- DN 패턴을 지정하려면 MQEnvironment의 `sslPeerName` 변수를 사용하십시오.

예를 들어,

```
MQEnvironment.SSLPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPPHERE";
```

큐 관리자가 QMGR.으로 시작하는 공통 이름(Common Name)을 가진 인증서를 제시하는 경우에만 연결이 성공하도록 허용합니다 그리고 최소 두 개의 조직 단위 이름이 있어야 하며, 첫 번째는 이고 두 IBM 번째는 여야 WEBSPPHERE 합니다.

SSLPeerName 특성은 연결 특성 해시 테이블의 MQC.SSL_PEER_NAME_PROPERTY도 사용하여 설정할 수 있습니다. 구별명 및 피어 이름 설정 규칙에 대한 자세한 내용은 [구별명을 참조하십시오](#).

SSLPeerName이 설정된 경우, 올바른 패턴으로 설정되고 큐 관리자가 일치하는 인증서를 제공하는 경우에만 연결에 성공합니다.

관리 대상 .NET 클라이언트에 대한 TLS 지원 활성화

관리 대상 .NET 클라이언트는 Microsoft.NET Framework 라이브러리를 사용하여 TLS 보안 소켓 프로토콜을 구현합니다. Microsoft System.Net.SecuritySslStream 클래스는 연결된 TCP 소켓에서 스트림으로서 작동하며, 해당 소켓 연결에서 데이터를 송신하고 수신합니다.

이 태스크 정보

최소 요구 .NET Framework 수준은 v3.5.NET Framework 입니다. 암호화 알고리즘 지원 수준은 애플리케이션이 사용하는 수준에 .NET Framework 기반합니다.

- 3.5 및 4.0 레벨을 .NET Framework 기반으로 하는 애플리케이션의 경우, 사용 가능한 보안 소켓 프로토콜은 다음과 같습니다: SSL 3.0 TLS 1.0
- 4.5 레벨을 .NET Framework 기반으로 하는 애플리케이션의 경우, 사용 가능한 보안 소켓 프로토콜은 다음과 같습니다: SSL 3.0 TLS 1.1 TLS 1.2

TLS 프로토콜의 고급 지원 기능을 기대하는 애플리케이션은 .NET FrameworkNET 프레임워크의 보안 Microsoft 지원에 정의된 대로 프레임워크의 최신 버전으로 이동해야 할 수 있습니다.

관리 대상 .NET 클라이언트에 대한 TLS 지원의 기본 기능은 다음과 같습니다.

TLS 프로토콜 지원

.NET 관리 대상 클라이언트에 대한 TLS 지원은 .NET SSLStream 클래스를 통해 정의되며, 애플리케이션이 사용 중인 .NET Framework에 의존합니다. 자세한 정보는 [562 페이지의 『관리 대상 .NET 클라이언트의 TLS 프로토콜 지원』](#)의 내용을 참조하십시오.

CipherSpec 지원

.NET 관리 대상 클라이언트의 TLS 설정은 Microsoft.NET TLS 스트림에 대한 설정입니다. 자세한 정보는 [562 페이지의 『관리 대상 .NET 클라이언트의 CipherSpec 지원』](#) 및 [564 페이지의 『관리 대상 .NET 클라이언트의 CipherSpec 맵핑』](#)의 내용을 참조하십시오.

키 저장소

클라이언트 측의 키 저장소는 또는 WindowsLinux 키스토어여야 하며 PKCS #12 형식이어야 합니다. 서버 측 저장소는 CMS(Cryptographic Message Syntax) 유형의 저장소입니다. 자세한 정보는 [566 페이지의 『관리 대상 .NET 클라이언트의 키 저장소』](#)의 내용을 참조하십시오.

V 3000 TLS 키 저장소 암호 속성이 추가로 있습니다IBM MQ 9.4.4 자세한 정보는 [573 페이지의 『TLS 연결 시 .NET 평문 또는 암호화된 비밀번호 사용을 위한 인증서 사용자 지정 경로』](#)의 내용을 참조하십시오.

인증서

자체 서명된 TLS 인증서를 사용하여 클라이언트 및 큐 관리자 간의 상호 인증을 구현할 수 있습니다. 자세한 정보는 [567 페이지의 『관리 대상 .NET 클라이언트의 인증서 사용』](#)의 내용을 참조하십시오.

SSLPEERNAME

.NET에서 애플리케이션은 선택적 SSLPEERNAME 속성을 사용하여 식별 이름(DN) 패턴을 지정할 수 있습니다. 자세한 정보는 [567 페이지의 『SSLPEERNAME』](#)의 내용을 참조하십시오.

FIPS 규제 준수

프로그래밍 방식으로 FIPS 활성화하는 기능은 보안 Microsoft.NET 라이브러리에서 지원되지 않습니다. FIPS 활성화는 그룹 정책 Windows 설정으로 제어됩니다.

NSA 스위트 B 준수

IBM MQ는 RFC 6460을 구현합니다. NSA 스위트 B의 Microsoft.NET 구현은 5430입니다. 이는 .NET Framework 3.5 이상에서 지원됩니다.

비밀 키 재설정 또는 재협상

SSLStream 클래스는 비밀 키 재설정 또는 재협상을 지원하지 않지만, 다른 IBM MQ 클라이언트와의 일관성을 위해 관리형 .NET 클라이언트는 애플리케이션이 설정을 허용합니다. SSLKeyResetCount. 자세한 정보는 [568 페이지의 『관리 .NET 대상 클라이언트의 비밀 키 재설정 또는 재협상』](#)의 내용을 참조하십시오.

폐기 확인

SSLStream 클래스는 인증서 폐기 확인을 지원하며, 이는 인증서 체인 엔진에 의해 자동으로 수행됩니다. 자세한 정보는 [568 페이지의 『폐기 확인』](#)의 내용을 참조하십시오.

IBM MQ 보안 엑시트 지원

해당 SSLStream 클래스는 보안 IBM MQ 종료에 대한 제한된 지원을 제공합니다. 로컬 및 원격 인증서를 조회하여 (주체 DN) 및 SSLRemCertIssNamePtr (SSLPeerNamePtr발행자 DN)을 가져오는 것이 가능합니다. 이는 Microsoft.NET 에서 지원되기 때문입니다. 그러나 DNQ, UNSTRUCTUREDNAME 및 UNSTRUCTUREDADDRESS 등의 속성 가져오기는 지원되지 않으므로, 이러한 값은 엑시트를 사용하여 검색될 수 없습니다.

암호화 하드웨어 지원

암호화 하드웨어는 관리 대상 .NET 클라이언트에 대해 지원되지 않습니다.

관련 개념

[564 페이지의 『관리 대상 .NET 클라이언트의 CipherSpec 매핑』](#)

IBM MQ.NET 인터페이스는 관리 대상 클라이언트가 큐 관리자와의 보안 연결을 설정하는 데 사용하기 위해 필요한 TLS 프로토콜의 버전을 판별하는 데 사용되는 IBM MQ 대 Microsoft.NET 매핑 테이블을 유지보수합니다.

관리형 .NET 클라이언트에서 TLS1.3 지원

TLS1.3 는 운영 체제가 TLS1.3IBM MQ 9.4.0을 지원하는 경우,, IBM MQ.NET 및 XMS .NET 클라이언트 지원을 제공합니다.

관리 대상 .NET 클라이언트는 Microsoft.NET Framework 라이브러리를 사용하여 TLS 보안 소켓 프로토콜을 구현합니다. 해당 MicrosoftSystem.Net.SecuritySslStream 클래스는 연결된 TCP 소켓을 통해 스트림으로 작동하며, 해당 소켓 연결을 통해 데이터를 송수신합니다.

에서 WindowsCHANNEL을 사용하고 .NET ,에서는 SSL 통신을 Linux.NET 위해 OpenSSL 을 사용합니다.

클라이언트 IBM MQ.NET 애플리케이션이 실행되는 Windows

Windows

Microsoft 11을 Windows 발표했으며 기본적으로 TLS1.3Windows Server 2022 암호화를 지원합니다.

TLS_AES_128_GCM_SHA256 TLS_AES_256_GCM_SHA384 암호 모음은 두 버전의 모두에서 기본적으로 Windows활성화되어 있습니다.



주의:

- TLS_CHACHA20_POLY1305_SHA256 암호 모음은 기본적으로 활성화되어 있지 않지만 지원됩니다.
- TLS1.3 이 활성화된 클라이언트가 IBM MQ.NET 큐 관리자에 성공적으로 연결하려면, 큐 관리자 IBM Global Security Kit (GSKit) 측에서 최소한 8.0.55.29 버전이 필요합니다.

클라이언트 IBM MQ.NET 애플리케이션이 실행되는 Linux

Linux

OpenSSL 를 .NET 사용하여 SSL 통신 기능을 Linux 이용하려면, TLS1.3, OpenSSL, v1.1.1 가 최소 요구 사항입니다.

또한, 에서 LinuxOpenSSL 를 .NET 사용하므로, OpenSSL 에서 지원하는 모든 암호화 알고리즘은 에서도 .NET 작동해야 합니다.

OpenSSL 다음 CipherSpecs 를 지원합니다: TLS1.3:

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_AES_128_CCM_SHA256

관련 개념

[564 페이지의 『관리 대상 .NET 클라이언트의 CipherSpec 매핑』](#)

IBM MQ.NET 인터페이스는 관리 대상 클라이언트가 큐 관리자와의 보안 연결을 설정하는 데 사용하기 위해 필요한 TLS 프로토콜의 버전을 판별하는 데 사용되는 IBM MQ 대 Microsoft.NET 매핑 테이블을 유지보수합니다.

관리 대상 .NET 클라이언트의 TLS 프로토콜 지원

IBM MQ.NET TLS 지원은 .NET SSLStream 클래스를 기반으로 합니다.

참고: 관리 대상 .NET 클라이언트에 대한 TLS 프로토콜 지원은 애플리케이션이 사용 중인 .NET Framework 레벨에 의존합니다. 자세한 정보는 [560 페이지의 『관리 대상 .NET 클라이언트에 대한 TLS 지원 활성화』](#)의 내용을 참조하십시오.

Microsoft.NET SSLStream 클래스가 TLS를 초기화하고 큐 관리자와 데이터 교환을 수행할 수 있도록 하기 위해 사용자가 설정해야 하는 필수 매개변수 중 하나는 **SSLProtocol**입니다. 여기서 사용자는 TLS 버전 번호를 지정해야 하며, 이는 다음 값 중 하나여야 합니다.

- SSL3.0
- TLS1.0
- TLS1.2

참고:  SSL3.0, TLS 및 1.0 프로토콜은 부터 더 이상 IBM MQ.NETIBM MQ 10.0.0 사용되지 않습니다. MQEnvironment의 `MQC.SSL\_CIPHER\_SPEC\_PROPERTY` 또는 `SSLCipherSpec` 속성 아래에 정의된 Cipherspecs 중 `SSL`, `3.0`, `TLS` 또는 `1.0` 프로토콜을 사용하는 것은 보안 취약점으로 인해 사용을 자제해야 하며, 향후 릴리스에서 지원이 중단될 예정입니다. 이 두 프로토콜 중 하나를 사용할 경우 클라이언트 애플리케이션의 오류 로그에 경고 메시지가 기록되며, TLS 1.2 프로토콜 이상을 사용할 것을 권장합니다.

매개변수 **SSLProtocol** 값은 선호되는 포트(CipherSpec)가 속한 프로토콜 패밀리(protocol family)와 밀접하게 결합되어 있습니다. SSLStream이 서버(큐 관리자)와 TLS 데이터 교환을 시작할 때 이는 **SSLProtocol**에 지정된 TLS 버전을 사용하여 협상에 사용되는 CipherSpec의 목록을 식별합니다.

IBM MQ.NET은 이 값을 설정하기 위해 애플리케이션이 사용할 수 있는 특성을 작성하지 않습니다. 대신 IBM MQ는 매핑 테이블을 사용하여 CipherSpec 세트를 프로토콜 패밀리로 내부적으로 매핑하며 사용될 SSLProtocol 버전을 식별합니다. 이 테이블은 Microsoft.NET 및 IBM MQ 간에 지원되는 각각의 CipherSpec의 매핑 및 이들이 속하는 프로토콜 버전을 표시합니다. 추가 정보는 [564 페이지의 『관리 대상 .NET 클라이언트의 CipherSpec 매핑』](#)의 내용을 참조하십시오.

관리 대상 .NET 클라이언트의 CipherSpec 지원

애플리케이션의 CipherSpec 설정은 서버와의 데이터 교환 중에 사용됩니다.

IBM MQ 클라이언트를 사용하면 큐 관리자와의 데이터 교환 중에 사용되는 CipherSpec 값을 설정할 수 있습니다. IBM MQ 클라이언트는 보안 연결이 설정될 수 있도록 반드시 올바른 CipherSpec을 설정해야 합니다(가급적이면 Windows 그룹 정책에 지정된 CipherSpec). 이 필드를 공백으로 두면 소켓에 보안이 없는 일반 텍스트 채널을 표시합니다.

IBM MQ.NET 관리 대상 클라이언트의 경우, TLS 설정은 Microsoft.NET SSLStream 클래스용입니다.

SSLStream의 경우, CipherSpec 또는 CipherSpec의 환경 설정 목록은 컴퓨터 전체 설정인 Windows 그룹 정책에서만 설정될 수 있습니다. 그리고 SSLStream은 서버와의 데이터 교환 중에 지정된 CipherSpec 또는 환경 설

정 목록을 사용합니다. 기타 IBM MQ 클라이언트의 경우, CipherSpec 특성은 IBM MQ 채널 정의의 애플리케이션에서 설정될 수 있으며 동일한 설정이 TLS 협상에 사용됩니다. 이 제한사항의 결과, TLS 데이터 교환은 IBM MQ 채널 구성에 지정된 것과 무관하게 지원되는 CipherSpec을 협상할 수 있습니다. 따라서 이의 결과로 큐 관리자에서 AMQ9631 오류가 발생할 수 있습니다. 이 오류를 방지하려면 Windows 그룹 정책에서 TLS 구성으로서 애플리케이션에서 설정된 것과 동일한 CipherSpec을 설정하십시오.

새 IBM MQ.NET TLS 클라이언트 코드는 올바른 프로토콜 버전이 협상되지만 확인합니다. TLS 프로토콜 버전은 애플리케이션이 설정하는 CipherSpec에서 도출되며, 서버(큐 관리자)와의 TLS 데이터 교환에 사용됩니다. 따라서 IBM MQ.NET 관리 대상 클라이언트 애플리케이션에서 CipherSpec을 설정하는 것이 디자인상 필요합니다. IBM MQ 클라이언트에서 설정한 CipherSpec이 SSL 3.0, TLS 1.0, TLS 1.2 프로토콜의 CipherSpec이 아닌 경우, IBM MQ 관리 대상 .NET 클라이언트는 기본적으로 SSL 3.0 또는 TLS 1.0 프로토콜의 CipherSpec 중 하나로 결정하며, 이는 오류를 보고하지 않습니다.

참고:  SSL (3.0 또는 TLS)을 사용하는 경우 IBM MQ 10.0.0, 1.0 은 더 이상 권장되지 않으며 클라이언트 애플리케이션의 오류 로그에 경고 메시지가 표시됩니다. 따라서 TLS, 1.2 또는 그 이상의 버전의 프로토콜을 사용하는 것이 좋습니다.

참고: 애플리케이션이 제공하는 CipherSpec 값이 IBM MQ에 알려진 CipherSpec이 아닌 경우, IBM MQ 관리 .NET 클라이언트는 이를 무시하고 Windows 시스템의 그룹 정책을 기반으로 연결을 협상합니다.

CipherSpec 설정

CipherSpec을 설정하는 세 가지 방법이 있습니다.

MQEnvironment .NET 클래스

다음 예제는 MQEnvironment 클래스로 CipherSpec을 설정하는 방법을 표시합니다.

```
MQEnvironment.SSLKeyRepository = "*USER";
MQEnvironment.ConnectionName = connectionName;
MQEnvironment.Channel = channelName;
MQEnvironment.properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA";
```

TLS CipherSpec 특성

다음 예제는 MQQueueManager 구성자에 해시 테이블 매개변수를 추가하여 CipherSpec을 설정하는 방법을 표시합니다.

```
properties = new Hashtable();
properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
properties.Add(MQC.PORT_PROPERTY, port);
properties.Add(MQC.CHANNEL_PROPERTY, channelName);
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
queueManager = new MQQueueManager(queueManagerName, properties);
```

Windows 그룹 정책

Windows 그룹 정책 관리 콘솔을 통해 Cipher Suite를 구성한 경우 SVRCONN 채널 정의에서는 일치하는 CipherSpec을 지정해야 합니다. 일치하는 CipherSpec은 "ANY_TLS12_OR_HIGHER"와 같은 일반 값이거나 정렬된 목록에서 협상되는 가장 높은 Cipher Suite에 매핑되는 특정 값일 수 있습니다. 일반 CipherSpec 값은 클라이언트 목록 순서가 변경된 경우 SVRCONN CipherSpec 구성을 변경해야 하는 상황을 방지하기 위해 .NET 클라이언트와 함께 사용하는 것이 좋습니다.

CCDT 사용법

IBM MQ.NET은 로컬 컴퓨터에 있는 CCDT(Client Channel Definition Table)(.TAB 파일)만 지원합니다. CipherSpec 값이 설정된 기존 CCDT 파일은 IBM MQ.NET 연결에 사용될 수 있습니다. 그러나 클라이언트 연결 채널에 설정된 CipherSpec 값은 TLS 프로토콜 버전을 판별하며, Windows 그룹 정책에 설정된 CipherSpec과도 일치해야 합니다.

관련 개념

549 페이지의 『IBM MQ 환경 설정』

클라이언트 연결을 사용하여 큐 관리자에 연결하기 전에 IBM MQ 환경을 설정해야 합니다.

관련 태스크

560 페이지의 『관리 대상 .NET 클라이언트에 대한 TLS 지원 활성화』

관리 대상 .NET 클라이언트는 Microsoft.NET Framework 라이브러리를 사용하여 TLS 보안 소켓 프로토콜을 구현합니다. Microsoft System.Net.SecuritySslStream 클래스는 연결된 TCP 소켓에서 스트림으로서 작동하며, 해당 소켓 연결에서 데이터를 송신하고 수신합니다.

CipherSpec 지정

관련 참조

[MQEnvironment.NET 클래스](#)

관리 대상 .NET 클라이언트의 CipherSpec 맵핑

IBM MQ.NET 인터페이스는 관리 대상 클라이언트가 큐 관리자와의 보안 연결을 설정하는 데 사용하기 위해 필요한 TLS 프로토콜의 버전을 판별하는 데 사용되는 IBM MQ 대 Microsoft.NET 맵핑 테이블을 유지보수합니다.

참고:  SSL (3.0 또는 TLS)을 사용하는 경우 IBM MQ 10.0.0, 1.0 은 더 이상 권장되지 않으며 클라이언트 애플리케이션의 오류 로그에 경고 메시지가 표시됩니다. 따라서 TLS, 1.2 또는 그 이상의 버전의 프로토콜을 사용하는 것이 좋습니다.

CipherSpec이 SVRCONN 채널에서 지정된 경우, TLS 데이터 교환이 완료된 후에 큐 관리자는 CipherSpec을 클라이언트 애플리케이션이 사용 중인 협상된 CipherSpec과 일치시키려고 시도합니다. 큐 관리자가 일치하는 CipherSpec을 찾을 수 없는 경우에는 통신이 AMQ9631 오류로 실패합니다.

IBM MQ.NET 인터페이스는 IBM MQ 대 Microsoft.NET CipherSpec 맵핑 테이블을 유지보수합니다. 이 테이블은 클라이언트가 큐 관리자와의 보안 소켓 연결을 설정하는 데 사용하고자 하는 TLS 프로토콜 버전을 판별하는 데 사용됩니다. SSLCipherSpec 값을 기반으로, SSLProtocol 버전은 사용 중인 Microsoft.NET Framework의 버전에 따라 TLS 1.0 또는 TLS 1.2일 수 있습니다.

SSLCipherSpec 값을 올바르게 지정해야 합니다. 잘못된 값을 지정할 경우 SSL 3.0 또는 TLS 1.0 프로토콜이 사용될 수 있습니다.

표 80. IBM MQ 및 Microsoft.NET 맵핑 테이블		
IBM MQ CipherSpec	Microsoft.NET CipherSpec	TLS 프로토콜
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0
TLS_RSA_WITH_3DES_EDE_CBC_SHA ¹	TLS_RSA_WITH_3DES_EDE_CBC_SHA ¹	TLS 1.0
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2

표 80. IBM MQ 및 Microsoft.NET 맵핑 테이블 (계속)

IBM MQ CipherSpec	Microsoft.NET CipherSpec	TLS 프로토콜
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P521	TLS 1.2
ECDHE_RSA_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2
ECDHE_RSA_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P521	TLS 1.2
TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS 1.3
TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3
TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS 1.3
TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS 1.3
TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS 1.3
알리어스 CipherSpec		
임의 566 페이지의 『4』	*ANY	다중
ANY_TLS13566 페이지의 『4』	*TLS13	TLS 1.3
ANY_TLS12_OR_HIGHER 566 페이지의 『4』	*TLS12ORHIGHER	TLS 1.2 이상
ANY_TLS13_OR_HIGHER 566 페이지의 『4』	*TLS13ORHIGHER	TLS 1.3 이상

참고:

1. **Deprecated** 이 CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA는 더 이상 사용되지 않습니다. 그러나 연결이 오류로 종료되기 전까지 최대 32GB의 AMQ9288 데이터를 전송하는 데 여전히 사용할 수 있습니다. 이 오류를 방지하려면 트리플 DES 사용을 피하거나, 이 CipherSpec 를 사용할 때 비밀 키 재설정을 활성화하십시오.
2. 응용 프로그램에 설정된 보안 계층 프로토콜(CipherSpec)은 관리되는 클라이언트가 큐 관리자(queue manager)와 보안 연결을 설정하는 데 사용하는 보안 계층 프로토콜(TLS) 버전을 결정하는 데 사용됩니다. 그러나 TLS 핸드셰이크는 OS 수준에서 설정된 CipherSpec 을 사용하여 완료되며, 그룹 정책의 암호

Windows 모음 순서에 따라야 합니다. 또한 애플리케이션에 설정된 CipherSpec 및 큐 매니저에 설정된 암호 모음과도 일치해야 합니다. 자세한 내용은 [관리형 .NET 클라이언트에 대한 CipherSpec 지원을 참조하십시오](#).

3. .NET IBM MQ 암호 모음 순서(선호도)를 결정하지 않습니다. Windows 그룹 정책에 아무것도 설정되지 않은 경우, 핸드셰이크에 Microsoft 기본 암호화 사양(cipherspec)이 사용됩니다. 자세한 내용은 [CipherSpec](#) 를 참조하십시오. TLS 핸드셰이크에서 순서를 참조하십시오.
4. TLS 1.3 암호화를 사용하려면, 애플리케이션을 실행하는 운영 체제(JRE)가 Java runtime environment TLS 1.3 를 지원해야 합니다.

관련 개념

[CipherSpec 관리형 .NET 클라이언트 지원](#)

애플리케이션의 CipherSpec 설정은 서버와의 데이터 교환 중에 사용됩니다.

관련 태스크

560 페이지의 [『관리 대상 .NET 클라이언트에 대한 TLS 지원 활성화』](#)

관리 대상 .NET 클라이언트는 Microsoft.NET Framework 라이브러리를 사용하여 TLS 보안 소켓 프로토콜을 구현합니다. Microsoft System.Net.SecuritySslStream 클래스는 연결된 TCP 소켓에서 스트림으로서 작동하며, 해당 소켓 연결에서 데이터를 송신하고 수신합니다.

[관리 IBM MQ .NET에 대한 TLS 구성](#)

IBM MQ .NET의 TLS 구성은 서명자 인증서 작성 및 서버 측, 클라이언트 측과 애플리케이션 프로그램 구성으로 구성되어 있습니다.

관리 대상 .NET 클라이언트의 키 저장소

관리 .NET 대상 클라이언트가 사용하는 키 저장소는 및 WindowsLinux 키스토어이며, 반드시 PKCS #12 형식이어야 합니다. TLS 핸드셰이크 중 클라이언트 애플리케이션이 인증서 및 개인 키를 ID 및 신뢰에 모두 사용할 수 있게 하려면 사용자 또는 시스템 키 저장소에서 인증서 및 개인 키가 사용 가능해야 합니다.

클라이언트 측

애플리케이션에서는 키 저장소에 대해 다음 값 중 하나를 설정할 수 있습니다.

- `"*USER"`: IBM MQ.NET이(가) 클라이언트 인증서를 검색하기 위해 현재 사용자의 인증서 저장소에 액세스합니다.
- `"*SYSTEM"`: IBM MQ.NET이(가) 인증서를 검색하기 위해 로컬 컴퓨터 계정에 액세스합니다.

클라이언트의 인증서는 사용자 또는 컴퓨터 계정의 내 인증서 저장소에 저장해야 합니다. 모든 서버(CA) 인증서는 인증서 저장소의 루트 디렉토리에 저장되어야 합니다.

참고: 다음 형식으로 단일 파일에 둘 이상의 인증서를 저장할 수 있습니다.

- 개인 정보 교환 - PKCS #12(.PFX, .P12)
- 암호 메시지 구문 표준 - PKCS #7 인증서(.P7B)
- Microsoft 직렬화 인증서 저장소(.SST)

TLS 연결용 별도의 인증서 경로

V 10.0.0

IBM MQ 10.0.0부터, .NET 애플리케이션은 사용자 지정 경로를 지정하여 TLS 연결에 별도의 인증서 경로를 사용할 수 있습니다.

또한 Dotnet Password Encryptor(**rundotnetcred**)라는 암호화 도구를 .NET 사용하여 암호화된 암호를 생성할 수 있습니다. 이 도구는 제공된 암호화 키와 함께 일반 텍스트 형식의 키스토어 액세스 암호를 처리하여 암호화된 암호를 생성하며, 이를 통해 TLS 연결의 보안을 강화할 수 있습니다. 동일한 이름의 파일이 bin(용도: Windows) 및 lib64(용도: Linux)에 있습니다.

자세한 내용은 [일반 텍스트 또는 암호화된 암호를 사용하는 TLS 연결을 위한 .NET의 인증서 사용자 지정 경로를 참조하십시오](#).

관리 대상 .NET 클라이언트의 인증서 사용

클라이언트 인증서의 경우, 관리되는 .NET IBM MQ 클라이언트는 키스토어 Windows 또는 Linux 키스토어에 접근하여 인증서 레이블 또는 문자열로 일치하는 모든 클라이언트 인증서를 로드합니다.

사용할 인증서를 선택할 때 IBM MQ 관리 대상 .NET 클라이언트는 항상 SSLStream TLS 데이터 교환을 위해 첫 번째 일치하는 인증서를 사용합니다.

참고: IBM MQ 10.0.0에서, TLS 인증서를 생성하고 보관하는 파일 시스템상의 키스토어에 대한 사용자 지정 경로를 사용할 수 있습니다. 이 사용자 지정 경로를 사용하면 애플리케이션 실행 시 인증서를 검색하여 TLS 핸드셰이크에 사용할 수 있습니다. TLS 인증서의 암호를 클라이언트 .NET 애플리케이션에 전달하기 전에 안전하게 암호화할 수도 있습니다. 자세한 정보는 573 페이지의 『TLS 연결 시 .NET 평문 또는 암호화된 비밀번호 사용을 위한 인증서 사용자 지정 경로』의 내용을 참조하십시오.

인증서 레이블로 인증서 일치

인증서 레이블을 설정하면 관리형 .NET IBM MQ 클라이언트는 지정된 레이블 이름으로 인증서 저장소 Windows 또는 인증서 Linux 저장소를 검색하여 클라이언트 인증서를 식별합니다. 이는 모든 일치하는 인증서를 로드하고 목록에서 첫 번째 인증서를 사용합니다. 인증서 레이블을 설정하는 두 가지 옵션이 있습니다.

- 인증서 레이블은 MQEnvironment.CertificateLabel에 액세스하는 MQEnvironment 클래스에서 설정될 수 있습니다.
- 다음 예제에서 표시된 대로, 인증서 레이블은 MQQueueManager 구성자에서 매개변수로서 제공되어 해시 테이블 특성에 설정될 수도 있습니다.

```
Hashtable properties = new Hashtable();  
properties.Add("CertificateLabel", "mycert");
```

이름("CertificateLabel") 및 값은 대소문자를 구분합니다.

문자열로 인증서 일치

인증서 레이블이 설정되지 않은 경우, 문자열 "ibmwebspheremq" 및 현재 로그인한 사용자(소문자)와 일치하는 인증서가 검색되고 사용됩니다.

관련 태스크

[클라이언트를 큐 관리자에 안전하게 연결](#)

관련 참조

[MQEnvironment.NET 클래스](#)

SSLPEERNAME

SSLPEERNAME 속성을 사용하여 피어 큐 관리자에서 인증서의 식별 이름(DN)을 확인할 수 있습니다.

IBM MQ.NET에서, 애플리케이션은 SSLPEERNAME을 사용하여 다음 예제에서 표시된 대로 식별 이름 패턴을 지정할 수 있습니다.

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSphere)
```

기타 IBM MQ 클라이언트의 경우처럼 SSLPEERNAME은 선택적 매개변수입니다.

SSLPEERNAME 값이 설정되지 않은 경우, IBM MQ.NET 관리 대상 클라이언트는 원격(서버) 인증서 유효성 검증을 수행하지 않으며 관리 대상 클라이언트는 단지 원격(/서버) 인증서만 있는 그대로 허용합니다.

SSLPEERNAME을 설정하는 방법은 사용 중인 IBM MQ 스택 오퍼링에 따라 다릅니다.

IBM MQ classes for .NET

다음과 같이 세 가지 옵션이 있습니다.

- MQEnvironment 클래스에서 MQEnvironment.SSLPeerName을 설정하십시오.
- MQEnvironment.properties.Add(MQC.SSL_PEER_NAME_PROPERTY, value)
- 큐 관리자 생성자 MQQueueManager (String queueManagerName, Hashtable properties)를 사용하십시오. 옵션 2의 경우처럼 Hashtable properties에서 SSLPEERNAME을 제공하십시오.

XMS.NET

연결 팩토리에서 SSL 피어 이름을 설정하십시오.

```
ConnectionFactory.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, value);
```

WCF

URI에서 세미콜론으로 분리된 필드로서 `SslPeerName`을 포함하십시오.

관련 참조

[MQEnvironment.NET 클래스](#)

관리 *.NET* 대상 클라이언트의 비밀 키 재설정 또는 재협상

`SSLStream` 클래스는 비밀 키 재설정/재협상을 지원하지 않습니다. 그러나 다른 IBM MQ 클라이언트와의 일관성을 유지하기 위해 관리형 *.NET* IBM MQ 클라이언트는 애플리케이션이 이를 설정할 수 있도록

SSLKeyResetCount 허용합니다.

한계에 도달하는 경우, IBM MQ.NET은 큐 관리자와의 연결이 끊어지며 애플리케이션은 이유 코드로서 `MQRC_CONNECTION_BROKEN`의 예외로 이에 대한 알림을 받습니다. 애플리케이션은 예외를 핸들링하고 연결을 재설정하거나 IBM MQ.NET의 `MQCNO_RECONNECT` 옵션을 사용하여 큐 관리자에 자동 재연결하도록 선택할 수 있습니다.

자동 클라이언트 다시 연결 기능의 사용은 키 재설정 계수에 도달할 때 모든 기존 연결이 다운되며 IBM MQ.NET 클라이언트가 모드 연결을 새로 재작성함을 의미합니다. 자동 클라이언트 다시 연결에 대한 자세한 정보는 [자동 클라이언트 다시 연결](#)을 참조하십시오.

관련 개념

[SSL 및 TLS 비밀 키 재설정](#)

폐기 확인

`SSLStream` 클래스는 인증서 폐기 확인을 지원합니다.

폐기 확인은 인증서 체인 엔진에 의해 자동으로 수행됩니다. 이는 OCSP(Online Certificate Status Protocol) 및 CRL(Certificate Revocation List) 모두에 적용됩니다. `SSLStream` 클래스는 인증서에 지정된 서버만 사용하는 인증서 폐기를 사용합니다(즉 서버가 인증서 자체의 지시를 받음). HTTP CDP 확장 및 OCSP HTTP 요청이 HTTP 프록시 서버를 통해 프록시하는 것이 가능합니다.

폐기 확인을 설정하는 방법은 사용 중인 IBM MQ 스택 오퍼링에 따라 다릅니다.

IBM MQ.NET

폐기 확인은 `MQEnvironment.cs` 클래스 파일에서 **`MQEnvironment.SSLCertRevocationCheck`** 특성에 액세스하여 설정될 수 있습니다.

XMS.NET

폐기 확인은 다음 예제에서 표시된 대로 연결 팩토리 특성 컨텍스트에서 설정될 수 있습니다.

```
ConnectionFactory.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

WCF

폐기 확인은 다음 이름 지정 규칙을 사용하여 URI에서 설정될 수 있습니다.

```
"SslCertRevocationCheck=true"
```

관리 대상 IBM MQ .NET의 TLS 구성

IBM MQ .NET의 TLS 구성은 서명자 인증서 작성 및 서버 측, 클라이언트 측과 애플리케이션 프로그램 구성으로 구성되어 있습니다.

이 태스크 정보

참고:  SSL (3.0 또는 TLS)을 사용하는 경우 IBM MQ 10.0.0, 1.0 은 더 이상 권장되지 않으며 클라이언트 애플리케이션의 오류 로그에 경고 메시지가 표시됩니다. 따라서 TLS, 1.2 또는 그 이상의 버전의 프로토콜을 사용하는 것이 좋습니다.

TLS를 구성하려면 우선 적절한 서명자 인증서를 작성해야 합니다. 서명자 인증서는 자체 서명되거나 인증 기관이 제공한 인증서일 수 있습니다. 자체 서명 인증서가 개발, 테스트 및 사전 프로덕션 시스템에서 사용될 수 있지만, 이를 프로덕션 시스템에서 사용하지 마십시오. 프로덕션 시스템에서는 신뢰 외부 인증 기관(CA)에서 확보한 인증서를 사용하십시오.

V 10.0.0에서 IBM MQ 10.0.0 ` TLS ` 인증서에 대한 사용자 지정 경로를 구성하여 인증서 저장소 관리를 유연하게 수행하고, 애플리케이션에 .NET 대해 IBM MQ 시스템 키 저장소 외부의 경로를 지정할 수 있습니다. 이를 위해 Dotnet Password Encryptor (**rundotnetcred**) 도구를 사용할 수 있으며, 이 도구는 비밀번호의 안전한 암호화 및 암호화된 비밀번호 데이터 관리를 지원합니다. 자세한 정보는 573 페이지의 『TLS 연결 시 .NET 평문 또는 암호화된 비밀번호 사용을 위한 인증서 사용자 지정 경로』의 내용을 참조하십시오.

프로시저

1. 서명자 인증서를 작성하십시오.

- 자체 서명 인증서를 생성하려면 또는 **runmqktool runmqakm** 명령을 사용하십시오.

자세한 정보는 [AIX, Linux, and Windows에서 자체 서명된 개인 인증서 작성](#)을 참조하십시오.

- 인증 기관(CA)에서 큐 관리자 및 클라이언트에 대한 인증서를 확보하려면 [인증 기관에서 개인 인증서 확보](#)의 지시사항을 따르십시오.

2. 서버 측을 구성하십시오.

- 큐 관리자에 대한 [보안 연결](#) 방법에 설명된 대로, IBM Global Security Kit (GSKit)TLS 를 큐 관리자에 구성하십시오.

- SVRCONN 채널 TLS 속성을 설정하십시오.

- REQUIRED 또는 **SSLCAUTH** OPTIONAL 로 설정하십시오.

- SSLCIPH**를 적절한 CipherSpec으로 설정하십시오.

자세한 정보는 559 페이지의 『[비관리형 .NET 클라이언트에 대한 TLS 지원 활성화](#)』의 내용을 참조하십시오.

3. 클라이언트 측을 구성하십시오.

- Windows 인증서 저장소(사용자/컴퓨터 계정 아래의)로 클라이언트 인증서를 가져오십시오.

IBM MQ .NET은 Windows 인증서 저장소에서 클라이언트 인증서에 액세스합니다. 따라서 사용자는 Windows 인증서 저장소에 인증서를 가져와서 IBM MQ에 대한 보안 소켓 연결을 설정해야 합니다.

Windows 키 저장소에 액세스하고 클라이언트 측 인증서를 가져오는 방법에 대한 자세한 정보는 [인증서 및 개인 키 가져오기 또는 내보내기](#)를 참조하십시오.

- [안전하게 큐 관리자에 클라이언트 연결](#)에서 설명된 대로 CertificateLabel을 제공하십시오.

- 필요하면 Windows 그룹 정책을 편집하여 CipherSpec을 설정한 후에 Windows 그룹 정책 업데이트가 적용될 수 있도록 컴퓨터를 다시 시작하십시오.

4. 애플리케이션 프로그램을 구성하십시오.

- 보안 연결로서 연결을 표시하도록 MQEnvironment 또는 SSLCipherSpec 값을 설정하십시오.

사용자가 지정하는 값은 사용 중인 프로토콜(TLS)을 식별하는 데 사용됩니다. CipherSpec 설정은 지원되는 SSLProtocol 버전의 CipherSpec 중 하나여야 하며, 이는 Windows 그룹 정책에 지정된 CipherSpec과 동일할 수 있으며 가급적이면 이를 권장합니다. (지원되는 SSLProtocol 버전은 사용되는 .NET 프레임워크에 따라 다릅니다. SSLProtocol 버전은 사용 중인 Microsoft .NET Framework의 버전에 따라 TLS 1.0 또는 TLS 1.2일 수 있습니다.)

참고: 애플리케이션이 제공하는 CipherSpec 값이 IBM MQ에 알려진 CipherSpec이 아닌 경우, IBM MQ 관리 .NET 클라이언트는 이를 무시하고 Windows 시스템의 그룹 정책을 기반으로 연결을 협상합니다.

- SSLKeyRepository 특성을 "*SYSTEM" 또는 "*USER"로 설정하십시오.

- 옵션: SSLPEERNAME을 서버 인증서의 식별 이름(DN)으로 설정하십시오.

- [안전하게 큐 관리자에 클라이언트 연결](#)에서 설명된 대로 CertificateLabel을 제공하십시오.

- KeyResetCount, CertificationRevocationCheck 등의 필요한 추가 선택적 매개변수를 설정하고 FIPS를 사용하십시오.

TLS 프로토콜 및 TLS 키 저장소를 설정하는 방법의 예제

기본 .NET의 경우, 다음 예제에서 표시된 대로 MQEnvironment 클래스를 통해 TLS 프로토콜 및 TLS 키 저장소를 설정할 수 있습니다.

```
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "*USER";

MQEnvironment.properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

또는 다음 예제에서 표시된 대로 MQQueueManager 구성자의 일부로서 해시 테이블을 제공하여 TLS 프로토콜 및 TLS 키 저장소를 설정할 수 있습니다.

```
Hashtable properties = new Hashtable();
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

다음에 수행할 작업

570 페이지의 [『단순 애플리케이션 쓰기』](#) 관리형 .NET TLS 애플리케이션 개발을 IBM MQ 시작하는 방법에 대한 자세한 내용은 다음을 참조하세요.

관련 참조

[MQEnvironment.NET 클래스](#)

[KeyResetCount\(MQLONG\)](#)

[Federal Information Processing Standards \(FIPS\)에 대해 AIX, Linux, and Windows](#)

단순 애플리케이션 쓰기

연결 팩토리의 SSL 특성 설정, 큐 관리자 인스턴스, 연결, 세션 및 목적지 작성, 그리고 테스트 메시지 전송에 대한 예제를 포함하여 단순 IBM MQ 관리 대상 .NET TLS 애플리케이션을 작성하는 팁입니다.

시작하기 전에

568 페이지의 [『관리 대상 IBM MQ .NET의 TLS 구성』](#) 먼저, 설명된 대로 관리형 IBM MQ.NET 서비스용 TLS 을 구성해야 합니다.

기본 .NET의 애플리케이션 프로그램 구성에 대해, MQEnvironment 클래스를 사용하거나 MQQueueManager 구성자의 일부로서 해시 테이블을 제공하여 SSL 특성을 설정하십시오.

XMS .NET의 애플리케이션 프로그램 구성에 대해, 연결 팩토리의 특성 컨텍스트에서 SSL 특성을 설정하십시오.

프로시저

1. 다음 예제에서 표시된 대로, 연결 팩토리에 대해 SSL 특성을 설정하십시오.

IBM MQ.NET의 예제

```
properties = new Hashtable();
properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
properties.Add(MQC.PORT_PROPERTY, port);
properties.Add(MQC.CHANNEL_PROPERTY, channelName);
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
properties.Add("CertificateLabel", "ibmwebspheremq");
MQEnvironment.SSLCertRevocationCheck = sslCertRevocationCheck;
```

XMS .NET의 예제

```
cf.SetStringProperty(XMSC.WMQ_SSL_KEY_REPOSITORY, "sslKeyRepository");
cf.SetStringProperty(XMSC.WMQ_SSL_CIPHER_SPEC, cipherSpec);
cf.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, sslPeerName);
cf.SetIntProperty(XMSC.WMQ_SSL_KEY_RESETCOUNT, keyResetCount);
cf.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

2. 다음 예제에서 표시된 대로 큐 관리자 인스턴스, 연결, 세션 및 목적지를 작성하십시오.

MQ .NET의 예제

```
queueManager = new MQQueueManager(queueManagerName, properties);
Console.WriteLine("done");

// accessing queue
Console.WriteLine("Accessing queue " + queueName + "..");
queue = queueManager.AccessQueue(queueName, MQC.MQOO_OUTPUT +
MQC.MQOO_FAIL_IF_QUIESCING);
Console.WriteLine("done");
```

XMS .NET의 예제

```
connectionWMQ = cf.CreateConnection();
// Create session
sessionWMQ = connectionWMQ.CreateSession(false, AcknowledgeMode.AutoAcknowledge);

// Create destination
destination = sessionWMQ.CreateQueue(destinationName);

// Create producer
producer = sessionWMQ.CreateProducer(destination);
```

3. 다음 예제에서 표시된 대로 메시지를 송신하십시오.

MQ .NET의 예제

```
// creating a message object
message = new MQMessage();
message.WriteString(messageString);

// putting messages continuously
for (int i = 1; i <= numberOfMsgs; i++)
{
    Console.WriteLine("Message " + i + " <" + messageString + ">..");
    queue.Put(message);
    Console.WriteLine("put");
}
```

XMS .NET의 예제

```
textMessage = sessionWMQ.CreateTextMessage();
textMessage.Text = simpleMessage;
producer.Send(textMessage);
```

4. TLS 연결을 확인하십시오.

채널 상태를 확인하여 TLS 연결이 설정되었으며 올바르게 작동하는지 확인하십시오.

SSLStream의 추적 구성

SSLStream 클래스와 연관된 추적 이벤트 및 메시지를 캡처하려면 애플리케이션의 애플리케이션 구성 파일에 시스템 진단에 대한 구성 섹션을 추가해야 합니다.

이 태스크 정보

참고:

이 태스크는 IBM MQ classes for .NET Framework에만 적용됩니다. 응용 프로그램 구성 파일은 IBM MQ classes for .NET (.NET Standard 및 .NET 6 라이브러리)에서 지원되지 않습니다.

애플리케이션 구성 파일에 시스템 진단에 대한 구성 섹션을 추가하지 않는 경우, IBM MQ 관리 대상 .NET 클라이언트는 TLS 및 SSLStream 클래스와 연관된 이벤트, 추적 또는 디버깅 지점을 캡처하지 않습니다.

참고: 사용하여 추적을 IBM MQ 시작하는 것은 필요한 모든 TLSstrmqtrc 추적을 포착하지 못합니다.

프로시저

1. 애플리케이션 프로젝트에 대한 애플리케이션 구성(App.Config) 파일을 작성하십시오.
2. 다음 예제에서 표시된 대로 시스템 진단 구성 섹션을 추가하십시오.

```

<system.diagnostics>
  <sources>
    <source name="System.Net" tracemode="includehex">
      <listeners>
        <add name="ExternalSourceTrace"/>
      </listeners>
    </source>
    <source name="System.Net.Sockets">
      <listeners>
        <add name="ExternalSourceTrace"/>
      </listeners>
    </source>
    <source name="System.Net.Cache">
      <listeners>
        <add name="ExternalSourceTrace"/>
      </listeners>
    </source>
    <source name="System.Net.Security">
      <listeners>
        <add name="ExternalSourceTrace"/>
      </listeners>
    </source>
    <source name="System.Security">
      <listeners>
        <add name="ExternalSourceTrace"/>
      </listeners>
    </source>
  </sources>
  <switches>
    <add name="System.Net" value="Verbose"/>
    <add name="System.Net.Sockets" value="Verbose"/>
    <add name="System.Net.Cache" value="Verbose"/>
    <add name="System.Security" value="Verbose"/>
    <add name="System.Net.Security" value="Verbose"/>
  </switches>

  <sharedListeners>
    <add name="ExternalSourceTrace" type="IBM.WMQ.ExternalSourceTrace,
amqmdnet, Version=n.n.n.n, Culture=neutral, PublicKeyToken=dd3cb1c9aae9ec97" />
  </sharedListeners>
  <trace autoflush="true"/>
</system.diagnostics>

```



주의: add name 항목의 Version 필드는 사용 중인 .net amqmdnet.dll 파일의 버전이어야 합니다.

관련 태스크

[애플리케이션 구성 파일을 사용하여 클라이언트 IBM MQ classes for .NET Framework 추적](#)

관리 대상 .NET에서 TLS 구현을 위한 샘플 애플리케이션

IBM MQ classes for .NET TLS의 구현 예시를 보여주기 위해, .NET 관리형 .NET 환경용 샘플 애플리케이션 (XMS.NET)과 IBM MQ WCF용 사용자 정의 채널이 제공됩니다.

다음 표는 샘플 애플리케이션의 위치를 표시합니다. MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

표 81. 관리 대상 .NET에서 TLS 구현을 위한 샘플 애플리케이션의 위치	
IBM MQ.NET 스택 오퍼링	샘플의 위치
기본 .NET	MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\base\SimplePut\SimplePut.cs MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\base\SimpleGet\SimpleGet.cs
XMS.NET	MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleProducer\SimpleProducer.cs MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleConsumer\SimpleConsumer.cs

표 81. 관리 대상 .NET에서 TLS 구현을 위한 샘플 애플리케이션의 위치 (계속)	
IBM MQ.NET 스택 오퍼링	샘플의 위치
WCF용 IBM MQ 사용자 정의 채널	<code>MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\MQMessagingOneWayService.cs</code>

V 10.0.0 TLS 연결 시 .NET 평문 또는 암호화된 비밀번호 사용을 위한 인증서 사용자 지정 경로 애플리케이션에서 IBM MQ 10.0.0 IBM MQ.NET, 키스토어에 접근하기 위해 평문 또는 암호화된 암호를 사용하여 TLS 연결을 수행하기 위한 인증서를 .NET 가져오도록 파일 시스템에서 사용자 지정 경로를 설정할 수 있습니다.

TLS 구성에 사용되는 속성

IBM MQ 10.0.0 설정하기 전에, 다음 예시와 같이 속성을 **MQC.SSL_CERT_STORE_PROPERTY** 사용하여 키 저장소 위치를 또는 *USER*SYSTEM 로 설정할 수 있습니다:

```
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, "*SYSTEM")
```

또는

```
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, "*USER")
```

*SYSTEM은 해당 위치가 로컬 컴퓨터임을 나타내고, *USER는 해당 위치가 현재 사용자임을 나타냅니다.

TLS 연결을 설정하기 위한 인증서의 사용자 지정 경로를 지정하려면 속성과

MQC.SSL_CERT_STORE_PROPERTY 함께 다음 두 속성을 추가로 사용할 수 있습니다 IBM MQ 10.0.0:

- 키스토어(.p12 파일)의 경로를. **MQC.SSL_CERT_STORE_PROPERTY**을 사용하여 지정합니다.
- 키스토어 접근 비밀번호를 사용하여 **MQC.SSL_CERT_STORE_PASSWORD_PROPERTY**, 평문 또는 암호화된 형태로.
- 암호화된 비밀번호를 **MQC.ENCRYPTION_KEY_PROPERTY** 복호화하기 위해 선택적으로 사용할 수 있는 암호화 키 파일 경로.

다음 예시는 사용자 지정 경로 기능을 사용할 때 및 **MQC.SSL_CERT_STORE_PASSWORD_PROPERTY**

MQC.SSL_CERT_STORE_PROPERTY 속성을 사용하여 일반 텍스트 비밀번호를 설정하는 방법을 보여줍니다:

-  Windows의 경우:

```
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, "C:\\CustomPath_Certificates\\myapp.p12")
properties.Add(MQC.SSL_CERT_STORE_PASSWORD_PROPERTY, "123456")
```

-  Linux의 경우:

```
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, "/root/Certs/myapp.p12");
properties.Add(MQC.SSL_CERT_STORE_PASSWORD_PROPERTY, "123456");
```

where *myapp.p12* 는 인증서를 포함하는 키 저장소의 이름입니다. 지정하는 경로에 이 키스토어 이름을 포함해야 합니다.

TLS 연결에 암호화된 암호를 사용하려면, 클라이언트 애플리케이션에 .NET 암호화 키를 지정하기 위해 추가 속성인 **MQC.ENCRYPTION_KEY_PROPERTY**를 제공해야 합니다. 다음 예제는 사용자 지정 경로 기능을 사용할 때, **MQC.SSL_CERT_STORE_PASSWORD_PROPERTY** 및 **MQC.ENCRYPTION_KEY_PROPERTY** 속성을 사용하여 암호화된 암호를 설정하는 방법을 보여줍니다:

-  Windows의 경우:

```
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, @"C:\CustomPath_Certificates\myapp.p12");
properties.Add(MQC.SSL_CERT_STORE_PASSWORD_PROPERTY, "<DOTNET>2!KJL4YP2sZ/Xu8cf6DgW0F1h+inu8nnYUzo8K1KRJH4U=!YMSX1qtjEtcinz8P")
```

```
properties.Add(MQC.ENCRYPTION_KEY_PROPERTY, "C:\\CustomPath_Certificates\\EncryptionKey\\mykey.txt");
```

• Linux의 경우:

```
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, "/root/Certs/myapp.p12");
properties.Add(MQC.SSL_CERT_STORE_PASSWORD_PROPERTY, "<DOTNET>2!KJL4YP2sZ/Xu8cf6DgW0F1h+inu8nnYUzo8KlKRJH4U=!YMSX1qtjEtcinz8P");
properties.Add(MQC.ENCRYPTION_KEY_PROPERTY, "/root/Certs/EncryptionKey/mykey.txt");
```

설명:

- 첫 번째 코드 줄은 키스토어가 포함된 위치를 필요로 하는 현재 재사용되는 속성입니다. 두 번째 코드 줄은 새로운 비밀번호 속성입니다. 세 번째 코드 줄은 키가 포함된 위치를 필요로 하는 새로운 암호화 키 속성입니다.
- *myapp.p12* 경로에 존재하는 키스토어 이름(인증서를 포함)이며, 경로와 함께 제공되어야 합니다.

참고: 클라이언트 .NET 애플리케이션에서 암호화 키 속성이 설정되지 않은 경우, 라이브러리는 비밀번호 속성에 설정된 값이 평문인지 또는 Base64 인코딩 형식인지 확인한 후 애플리케이션은 TLS 연결을 진행합니다.

- 값이 일반 텍스트로 되어 있다면, 이를 일반 텍스트 비밀번호로 처리합니다.
- 값이 Base64 로 인코딩된 경우, 기본 키를 사용하여 해당 값을 복호화한 후 TLS 연결을 구축합니다.

닷넷 암호화 도구 (rundotnetcred)

이 **rundotnetcred** 도구를 사용하면 제공된 암호화 키로 일반 텍스트 형식의 키스토어 비밀번호를 처리하여 암호화된 비밀번호를 생성할 수 있습니다. 이 암호화된 비밀번호는 TLS 연결의 보안을 강화하는 데 사용할 수 있습니다.



주의: 이 **rundotnetcred** 도구는 평문 비밀번호를 암호화하기 위해 고급 암호화 표준(AES) 알고리즘을 사용하므로, 128비트 또는 256비트 키를 입력으로 제공해야만 AES-128-CBC 또는 AES-256-CBC 암호화를 활성화할 수 있습니다.

명령줄을 사용하여 도구를 **rundotnetcred** 실행하십시오:

```
<Program-name> -p <PT-pass> -sf <Encryptionkey-filepath>
```

여기서,

- **-p** 일반 텍스트 비밀번호입니다.
- **-sf** 암호화 키의 파일 경로입니다.

참고: **-sf** 필수 매개변수가 아니며, 설정되지 않은 경우 도구는 평문 비밀번호를 암호화하기 위해 기본 암호화 키를 사용합니다. 그러나 자체 암호화 키를 사용하는 것이 권장됩니다. 애플리케이션의 .NET 'encryption' **ENCRYPTION_KEY_PROPERTY.NET** 속성에 (에서 사용된 것과 **rundotnetcred** 동일한) 암호화 키를 반드시 사용하십시오.

암호화된 문자열이 의 값이 **SSL_CERT_STORE_PASSWORD_PROPERTY**되며, 키 파일 경로는 에 설정됩니다 **ENCRYPTION_KEY_PROPERTY**. 암호화 키 값은 주로 평문 비밀번호를 암호화하는 데 필요한 암호화 키를 포함하는 파일의 위치입니다.

다음 예제는 도구를 **rundotnetcred** 사용하는 방법을 보여줍니다:

```
rundotnetcred -p APlaintextPassword -sf C:\temp\Key\EncryptionKey.txt
```

설명:

- **rundotnetcred** 응용 프로그램의 이름입니다.
- *APlaintextPassword* TLS 연결을 위해 암호화해야 하는 일반 텍스트 비밀번호입니다.
- *C:\temp\Key\EncryptionKey.txt* AES 암호화에 사용되는 암호화 키의 경로입니다.

이 유틸리티의 출력 형식은 다음과 같습니다:

```
<DOTNET>2!iS9ccKTjzS35SbqHQzQ3yKfodnNeId5UnWreJ99v+/0=!SGzPyIJnWaFacu5h
```

다양한 문자열의 조합입니다:

- 눈길을 끄는 것
- 알고리즘 유형
- 분리자
- Base64-sfncoded 암호화된 비밀번호
- 분리자
- 벡터 (IV)

오류 처리 및 추적

다음 MQRC 오류 코드는 사용자 지정 경로 기능과 관련된 모든 오류를 처리하는 데 사용됩니다:

- MQRC 2393 - MQRC_SSL_초기화_오류
- MQRC 2381 - MQRC_SSL_KEYREPOSITORY_ERROR
- MQRC 2392 - MQRC_SSL_CONFIG_ERROR

자세한 내용은 IBM MQ 의 [.NET 클라이언트 라이브러리에서 발생하는 일반적인 SSL 오류 코드를 참조하십시오](#).

다음과 같은 시나리오에서 예외가 반환됩니다:

- 암호화 키 파일 경로에 암호화 키가 포함된 파일이 존재하지 않으면, 예외(`FileNotFoundException`)가 반환됩니다.
- 암호화 키가 포함된 파일에 null 값이 존재하는 경우, `ArgumentNullException` 가 반환됩니다.

유틸리티에 암호화 키 속성을 제공하지 않을 경우, 암호화를 수행하기 위해 미리 정의된 기본 키가 대신 사용됩니다. 이 기본 키는 내부적으로 정의되며 128비트 또는 256비트를 사용합니다. 오류가 발생하지 않습니다.

Windows .NET Monitor 사용

.NET Monitor는 IBM MQ 트리거 모니터와 비슷한 애플리케이션입니다.

중요사항: 중요한 정보는 [\[여기\]\(링크\)](#)의 Windows ‘기본 설치 환경에서만 사용할 수 있는 기능’을 참조하십시오.

모니터된 큐에서 메시지를 수신할 때마다 인스턴스화한 다음 해당 메시지를 처리하는 .NET 컴포넌트를 작성할 수 있습니다. .NET Monitor는 `runmqdnm` 명령을 사용하여 시작하고 `endmqdnm` 명령을 사용하여 중지합니다. 이러한 명령에 대한 자세한 내용은 `runmqdnm` 및 `endmqdnm`을 참조하십시오.

.NET Monitor를 사용하려면 `amqmdnm.dll`에 정의된 `IMQObjectTrigger` 인터페이스를 구현하는 컴포넌트를 작성합니다.

컴포넌트는 트랜잭션 또는 비트랜잭션일 수 있습니다. 트랜잭션 컴포넌트는 `System.EnterpriseServices.ServicedComponent`에서 상속하며 `RequiresTransaction` 또는 `SupportsTransaction`으로 등록되어야 합니다. .NET Monitor가 이미 트랜잭션을 시작했으므로 `RequiresNew`로 등록하지 않아야 합니다.

컴포넌트가 `runmqdnm`에서 `MQQueueManager`, `MQQueue` 및 `MQMessage` 오브젝트를 수신합니다. `runmqdnm`이 시작된 경우 `-u` 명령행 옵션을 사용하여 사용자 매개변수 문자열(지정된 경우)도 수신할 수 있습니다. 사용자의 컴포넌트가 `MQMessage` 오브젝트에서 모니터링된 큐에 도착한 메시지의 콘텐츠를 수신한다는 점에 유의하십시오. 큐 관리자에 연결하거나 큐를 열거나 메시지 자체를 가져올 필요가 없습니다. 그런 다음 컴포넌트에서 메시지를 적절하게 처리한 다음 .NET Monitor에 제어를 리턴합니다.

컴포넌트를 트랜잭션 컴포넌트로 작성한 경우 `System.EnterpriseServices.ServicedComponent`에서 제공한 기능을 사용하여 트랜잭션을 롤백하거나 커밋하기 위해 컴포넌트를 등록합니다.

컴포넌트에서 메시지 외에도 `MQQueueManager`와 `MQQueue` 오브젝트를 수신하므로 해당 메시지에 대한 전체 컨텍스트 정보가 있으며, 별도로 IBM MQ에 연결하지 않아도 동일한 큐 관리자에서 다른 큐를 여는 등의 작업을 수행할 수 있습니다.

Windows 예제 코드 단편

이 주제에는 .NET Monitor에서 메시지를 확보하여 인쇄하는 두 예가 포함되어 있습니다. 한 예에서는 트랜잭션 처리를 사용하고 다른 예에서는 비트랜잭션 처리를 사용합니다. 세 번째 예는 처음 두 예에 모두 적용되는 공용 유틸리티 루틴을 보여줍니다. 모든 예는 C#으로 작성됩니다.

예 1: 트랜잭션 처리

```
/*
*****
/* Licensed materials, property of IBM
/* 63H9336
/* (C) Copyright IBM Corp. 2005, 2026.
*****
using System;
using System.EnterpriseServices;

using IBM.WMQ;
using IBM.WMQMonitor;

[assembly: ApplicationName("dnmsamp")]

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll TranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdmn -m QMNAME -q QNAME -a dnmsamp.dll -c Tran

namespace dnmsamp
{
    [TransactionAttribute(TransactionOption.Required)]
    public class Tran : ServicedComponent, IMQObjectTrigger
    {
        Util util = null;

        [AutoComplete(true)]
        public void Execute(MQQueueManager qmgr, MQQueue queue,
            MQMessage message, string param)
        {
            util = new Util("Tran");

            if (param != null)
                util.Print("PARAM: '" + param.ToString() + "'");

            util.PrintMessage(message);

            //System.Console.WriteLine("SETTING ABORT");
            //ContextUtil.MyTransactionVote = TransactionVote.Abort;

            System.Console.WriteLine("SETTING COMMIT");
            ContextUtil.SetComplete();
            //ContextUtil.MyTransactionVote = TransactionVote.Commit;
        }
    }
}
```

예 2: 비트랜잭션 처리

```
/*
*****
/* Licensed materials, property of IBM
/* 63H9336
/* (C) Copyright IBM Corp. 2005, 2026.
*****
using System;

using IBM.WMQ;
using IBM.WMQMonitor;

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll NonTranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdmn -m QMNAME -q QNAME -a dnmsamp.dll -c NonTran

namespace dnmsamp
{
    public class NonTran : IMQObjectTrigger
    {
        Util util = null;

        public void Execute(MQQueueManager qmgr, MQQueue queue,
            MQMessage message, string param)
        {
        }
    }
}
```

```

    {
        util = new Util("NonTran");

        try
        {
            util.PrintMessage(message);
        }

        catch (Exception ex)
        {
            System.Console.WriteLine(">>> NonTran\n{0}", ex.ToString());
        }
    }
}
}

```

예 3: 공통 루틴

```

/*****
/* Licensed materials, property of IBM */
/* 63H9336 */
/* (C) Copyright IBM Corp. 2005, 2026. */
*****/

using System;

using IBM.WMQ;

namespace dnmsamp
{
    /// <summary>
    /// Summary description for Util.
    /// </summary>
    public class Util
    {
        /* ----- */
        /* Default prefix string of the namespace. */
        /* ----- */
        private string prefixText = "dnmsamp";

        /* ----- */
        /* Constructor that takes the replacement prefix string to use. */
        /* ----- */
        public Util(String text)
        {
            prefixText = text;
        }

        /* ----- */
        /* Display an arbitrary string to the console. */
        /* ----- */
        public void Print(String text)
        {
            System.Console.WriteLine("{0} {1}\n", prefixText, text);
        }

        /* ----- */
        /* Display the content of the message passed to the console. */
        /* ----- */
        public void PrintMessage(MQMessage message)
        {
            if (message.Format.CompareTo(MQC.QMFMT_STRING) == 0)
            {
                try
                {
                    string messageText = message.ReadString(message.MessageLength);

                    Print(messageText);
                }

                catch (Exception ex)
                {
                    Print(ex.ToString());
                }
            }
            else
            {

```

```

        Print("UNRECOGNISED FORMAT");
    }
}

/* ----- */
/* Convert the byte array into a hex string.          */
/* ----- */
static public string ToHexString(byte[] byteArray)
{
    string hex = "0123456789ABCDEF";

    string retString = "";

    for(int i = 0; i < byteArray.Length; i++)
    {
        int h = (byteArray[i] & 0xF0)>>4;
        int l = (byteArray[i] & 0x0F);

        retString += hex.Substring(h,1) + hex.Substring(l,1);
    }

    return retString;
}
}
}

```

IBM MQ .NET 프로그램 컴파일

다양한 언어로 작성된 .NET 애플리케이션을 컴파일하는 기본 명령

*MQ_INSTALLATION_PATH*은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

IBM MQ classes for .NET를 사용하여 C# 애플리케이션을 빌드하려면 다음 명령을 사용하십시오.

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin /out:MyProg.exe
MyProg.cs
```

IBM MQ classes for .NET를 사용하여 Visual Basic 애플리케이션을 빌드하려면 다음 명령을 사용하십시오.

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:MyProg.exe MyProg.vb
```

IBM MQ classes for .NET를 사용하여 Managed C++ 애플리케이션을 빌드하려면 다음 명령을 사용하십시오.

```
cl /clr MQ_INSTALLATION_PATH\bin Myprog.cpp
```

다른 언어는 언어 벤더가 제공한 문서를 참조하십시오.

독립형 IBM MQ .NET 클라이언트 사용

클라이언트는 IBM MQ.NET 애플리케이션 실행을 위해 프로덕션 시스템에 전체 IBM MQ 클라이언트 설치 없이도 IBM MQ.NET 어셈블리를 패키징하고 배포할 수 있는 기능을 제공합니다.

시작하기 전에

기본 위치에 설치된 클라이언트 IBM MQ 9.4.0amqmdnetstd.dll 라이브러리는 를 기반으로 합니다 .NET 6.

, 에서 IBM MQ 9.4.0, IBM MQ classes for .NET 를 사용하는 IBM MQ 애플리케이션을 .NET 8 지원합니다. 해당 .NET 6 애플리케이션을 사용 중이라면, 파일을 runtimeconfig 약간 수정하여 값을 targetframeworkversion 로 설정함으로써 재컴파일 없이도 이 "net8.0"애플리케이션을 실행할 수 있습니다.

Removed 에서 비추천된 로 .NET Standard 2.0 구축된 클라이언트 IBM MQ.NETIBM MQ 9.3.1 라이브러리가 제품에서 제거되었습니다 IBM MQ 9.4.0.

Stabilized amqmdnet.dll 라이브러리는 여전히 제공되지만, 이 라이브러리는 안정화되었습니다. 즉, 새로운 기능이 소개되지 않습니다. 최신 기능을 위해 amqmdnetstd.dll 라이브러리로 마이그레이션해야 합니다. 그러나 또는 Continuous Delivery IBM MQ 9.1Long Term Support 버전에서 amqmdnet.dll 라이브러리를 계속 사용할 수 있습니다.

이 태스크 정보

전체 IBM MQ 클라이언트가 설치된 머신에서 IBM MQ.NET 애플리케이션을 빌드한 후, 어셈블리(즉 IBM MQ.NET, amqmdnetstd.dll)를 애플리케이션과 함께 패키징하여 프로덕션 시스템에 배포할 수 있습니다.

구축 및 배포하는 애플리케이션은 기존 .NET 애플리케이션, 서비스 또는 Microsoft Azure 웹/워커 애플리케이션 일 수 있습니다

이러한 배치에서는 IBM MQ .NET 클라이언트가 큐 관리자에 대한 연결성의 관리 모드만 지원합니다. 이러한 두 개의 모드가 전체 IBM MQ 클라이언트 설치를 필요로 하므로 서버 바인딩 및 관리되지 않는 클라이언트 모드 연결성을 사용할 수 없습니다. 이러한 다른 두 개의 모드를 사용하기 위한 시도도 결과적으로 애플리케이션 예외가 됩니다.

프로시저

애플리케이션에서 IBM MQ .NET 클라이언트 어셈블리 참조

- 이전 릴리스에서 수행한 동일한 방식으로 애플리케이션에서 amqmdnetstd.dll 어셈블리를 참조하십시오. 어셈블리의 amqmdnetstd.dll 속성을 **CopyLocal True** 로 설정하여 해당 amqmdnetstd.dll 어셈블리가 애플리케이션의 bin 디렉터리에 복사되도록 합니다. 또한 이 특성을 설정하면 Microsoft Azure PaaS 클라우드 환경뿐 아니라 프로덕션 시스템에 배치하기 위해 애플리케이션 패키징 도구가 필수 2진 파일을 패키징하도록 하는 데도 도움이 됩니다.

글로벌 트랜잭션 지원 추가

- 애플리케이션이 이 애플리케이션 자체와 함께 시스템에 모니터 애플리케이션 WMQDotnetXAMonitor를 배치하도록 하십시오. 애플리케이션이 IBM MQ .NET 관리 글로벌 트랜잭션 기능을 사용할 경우, 애플리케이션 자체와 함께 시스템에 WMQDotnetXAMonitor도 배치해야 합니다. 이 유틸리티는 인다우트 트랜잭션을 복구하는 데 필요합니다.

추적 시작 및 중지

- 애플리케이션 구성 파일과 특정 추적 IBM MQ 구성 파일을 사용하여 추적을 시작하고 중지하는 방법에 IBM MQ classes for .NET Framework 대해서는 [애플리케이션 구성 파일을 사용한 .NET Framework 클라이언트용 IBM MQ 클래스 추적을 참조하십시오](#).

애플리케이션 구성 파일과 특정 추적 IBM MQ 구성 파일을 반드시 사용해야 합니다. 클라이언트 IBM MQ 전체 설치가 없기 때문에 추적 시작 및 중지, **strmqtrcendmqtrc** 그리고 에 사용되는 표준 도구를 사용할 수 없기 때문입니다.

참고:

- 이러한 추적 생성 방식은 재배포 가능 관리형 .NET 클라이언트와 독립 .NET 실행형 클라이언트 모두에 적용됩니다. [애플리케이션 런타임 - Windows.NET 전용으로만 참조하십시오](#).
- 응용 프로그램 구성 파일은 IBM MQ classes for .NET (.NET Standard 및 .NET 6 라이브러리)에서 지원되지 않습니다. IBM MQ classes for .NET (.NET Standard 및 .NET 6 lib) 라이브러리에 대한 추적을 활성화하려면 환경 **MQDOTNET_TRACE_ON** 변수를 사용합니다. [환경 변수를 사용하여 애플리케이션 IBM MQ.NET 추적하기를 참조하십시오](#).
- 파일과 mqclient.ini Trace 스탠자의 적절한 속성을 설정하여 추적을 시작하고 중지합니다. [mqclient.ini 를 사용한 애플리케이션 IBM MQ.NET 추적 참조](#).

에서 IBM MQ 9.4.0, 파일을 mqclient.ini 사용하여 추적을 구성하고 Trace 섹션의 적절한 속성을 설정할 수 있습니다. 추적 기능을 동적으로 활성화 및 비활성화하려면 파일을 mqclient.ini 사용하십시오.

애플리케이션 구성 파일에서 바인딩 경로 재지정 사용

- 어셈블리의 추후 버전에 대한 IBM MQ .NET 어셈블리의 컴파일 시간 바인딩 참조를 사용하려면 `<dependentAssembly>` 특성을 애플리케이션 구성 파일에 추가하십시오. 다음 예제 코드 app.config 조각은 (8.0.0.2 IBM MQ 8.0.0 Fix Pack 2) 버전의 어셈블리로 컴파일된 애플리케이션을 리디렉션합니다 IBM MQ.NET. 그러나 이후 적용된 수정 8.0.0.3패치는 IBM MQ 8.0.0 Fix Pack 3 해당 어셈블리를 IBM MQ.NET 로 업데이트했습니다.

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
```

```

<!-- amqmdnet related binding redirect -->
<dependentAssembly>
  <assemblyIdentity name="amqmdnet"
    publicKeyToken="dd3cb1c9aae9ec97"
    culture="neutral" />
  <codeBase version="8.0.0.2"
    href="file:///amqmdnet.dll"/>
  <bindingRedirect oldVersion="1.0.0.3-8.0.0.2"
    newVersion="8.0.0.3"/>
  <publisherPolicy apply="no" />
</dependentAssembly>
</assemblyBinding>
</runtime>

```

관련 개념

521 페이지의 『설치 IBM MQ classes for .NET』

IBM MQ classes for .NET 샘플을 포함한 모든 구성 요소가 에 IBM MQ 설치됩니다 Windows . Linux

[재배포 가능 클라이언트](#)

[.NET 애플리케이션 런타임 - Windows만 해당](#)

관련 태스크

539 페이지의 『WMQDotnetXAMonitor 애플리케이션 사용』

IBM MQ .NET 클라이언트는 불완전한 분산 트랜잭션을 복구하는 데 사용할 수 있는 XA 모니터 애플리케이션 WmqDotnetXAMonitor를 제공합니다. WmqDotnetXAMonitor 애플리케이션은 트랜잭션이 인다우트 상태인 큐 관리자에 대한 연결을 설정한 후 사용자가 설정한 매개변수를 기반으로 트랜잭션을 해결합니다.

[IBM MQ.NET 애플리케이션 추적](#)

OutboundSNI 특성

애플리케이션에서 속성은 **OutboundSNI** 속성 또는 환경 변수를 사용하여 설정할 수 있습니다.

IBM MQ 9.3.0에서, MQQueueManager 클래스를 사용하여 큐 관리자에 연결할 때 해시 테이블을 사용하여 애플리케이션에서 MQC.OUTBOUND_SNI_PROPERTY를 설정할 수 있습니다.

MQC.OUTBOUND_SNI_PROPERTY는 다음 값을 사용합니다.

- "CHANNEL"로 매핑되는 MQC.OUTBOUND_SNI_CHANNEL
- "HOSTNAME"으로 매핑되는 MQC.OUTBOUND_SNI_HOSTNAME
- "*"로 매핑되는 MQC.OUTBOUND_SNI_ASTERISK

또한 다음 값을 사용하는 MQOUTBOUND_SNI 환경 변수를 사용하여 **OutboundSNI** 특성을 설정할 수 있습니다.

- CHANNEL
- 호스트 이름
- *

그리고 다른 mqclient.ini 속성과 마찬가지로 해당 App.config 파일에서 값을 **OutboundSNI** 설정하십시오.

참고: 특정 값이 설정되지 않은 경우 특성의 기본값은 MQC.OUTBOUND_SNI_CHANNEL로 설정됩니다.

관리 노드에서 **OutboundSNI** 특성을 설정하는 우선순위는 다음과 같습니다.

1. 애플리케이션 레벨 특성
2. 환경 변수

비관리 노드에 있는 **OutboundSNI** 특성의 경우 mqclient.ini만 지원됩니다.

해당 App.config 파일에 설정된 속성은 응용 .NET Framework 프로그램에만 적용됩니다.

애플리케이션 레벨 또는 App.config 파일에서 유효하지 않은 값을 제공하면 리턴 코드 MQRC_OUTBOUND_SNI_NOT_VALID가 발행됩니다.

환경 변수를 유효하지 않게 설정하거나 mqclient.ini 파일에서 유효하지 않은 값을 제공하면 의 CHANNEL 기본값이 사용됩니다.

OutboundSNI 그리고 여러 개의 인증서

IBM MQ SNI 헤더를 사용하여 다중 인증서 기능을 제공합니다. 응용 프로그램이 CERTLABL 필드를 통해 다른 인증서를 사용하도록 구성된 IBM MQ 채널에 연결하는 경우, 해당 응용 프로그램은 CHANNEL 설정으로 **OutboundSNI** 연결해야 합니다.

CHANNEL 이외의 **OutboundSNI** 설정값을 가진 애플리케이션이 인증서 레이블이 구성된 채널에 연결할 경우, 해당 애플리케이션은 MQRC_SSL_INITIALIZATION_ERROR로 거부되며, 큐 관리자 오류 로그에 AMQ9673 메시지가 기록됩니다.

다중 인증서 기능을 제공하는 IBM MQ 방법에 대한 자세한 내용은 '[다중 인증서 기능 제공 IBM MQ 방법](#)'을 참조하십시오.

XMS .NET 애플리케이션 개발

IBM MQ Message Service Client (XMS) for .NET (XMS .NET)는 라는 애플리케이션 프로그래밍 XMS 인터페이스(API)를 제공하며, 이는 (JMS) Java Message Service API와 동일한 인터페이스 집합을 가집니다. IBM MQ Message Service Client (XMS) for .NET에는 XMS의 전체 관리 구현이 포함되며, 이는 .NET을 준수하는 어에서 사용할 수 있습니다.

시작하기 전에

Removed 에서 비추천된 로 .NET Standard 2.0 구축된 IBM MQ 9.3.1클라이언트 XMS.NET 라이브러리는 에서 제품에서 IBM MQ 9.4.0제거되었습니다.

V 9.4.0 **V 9.4.0** **Deprecated** 부터 IBM MQ 9.4.0, 에 IBM MQ classes for XMS .NET, 데이터 직렬화 및 역직렬화에 사용되는 메서드 WriteObject(), ReadObject(), CreateObjectMessage(), 그리고 클래스 ObjectMessage 및 XmsObjectMessageImpl 는 더 이상 사용되지 않습니다.

Removed **V 10.0.0** 부터 IBM MQ classes for XMS .NETIBM MQ 10.0.0, 객체의 직렬화 및 역직렬화에 사용되는 메서드 WriteObject(), ReadObject(), CreateObjectMessage() 및 클래스 ObjectMessage, XmsObjectMessageImpl 가 제거됩니다.

이 태스크 정보

XMS에서는 다음을 지원합니다.

- 포인트-투-포인트 메시징
- 발행/구독 메시징
- 동기 메시지 전달
- 비동기 메시지 전달

XMS 애플리케이션은 다음 유형의 애플리케이션과 메시지를 교환할 수 있습니다.

- XMS 애플리케이션
- IBM MQ classes for JMS 애플리케이션
- 기본 IBM MQ 애플리케이션
- 기본 메시징 IBM MQ 공급자를 사용하는 애플리케이션 JMS

XMS 애플리케이션은 다음 메시징 서버에 연결하고 이에 대한 자원을 사용할 수 있습니다.

IBM MQ 큐 관리자

애플리케이션은 바인딩 또는 클라이언트 모드에서 연결될 수 있습니다.

WebSphere Application Server service integration bus

애플리케이션은 직접 TCP/IP 연결을 사용하거나 TCP/IP를 통한 HTTP를 사용할 수도 있습니다.

IBM Integration Bus

메시지는 WebSphere MQ Real-Time Transport를 사용하여 애플리케이션과 브로커 사이에서 전송됩니다.
메시지는 WebSphere MQ Multicast Transport를 사용하여 애플리케이션에 전달될 수 있습니다.

IBM MQ 큐 관리자에 연결하면 XMS 애플리케이션이 WebSphere MQ Enterprise Transport를 사용하여 IBM Integration Bus와 통신할 수 있습니다. 또는 XMS 애플리케이션이 IBM MQ에 연결하여 발행 및 구독할 수 있습니다.

IBM MQ 9.4.0 대상 .NET 6 프레임워크를 기반으로 빌드된 클라이언트 XMS.NET 라이브러리를 제공합니다. 자세한 정보는 585 페이지의 『설치 IBM MQ classes for XMS .NET』의 내용을 참조하십시오.

,에서 IBM MQ 9.4.0, IBM MQ classes for XMS .NET를 사용하는 IBM MQ 애플리케이션을 .NET 8 지원합니다. 자세한 정보는 585 페이지의 『설치 IBM MQ classes for XMS .NET』의 내용을 참조하십시오.

XMS .NET 관리되는 애플리케이션은 클러스터화된 큐 관리자 간에 연결을 자동으로 분산할 수 있습니다. 와 IBM MQ classes for XMS .NET IBM MQ classes for XMS .NET Framework 라이브러리 모두 지원됩니다. 자세한 정보는 [균등 클러스터 정보 및 자동 애플리케이션 밸런싱](#)을 참조하십시오.

와 IBM MQ classes for XMS .NET Framework의 차이점에 대한 IBM MQ classes for XMS .NET 자세한 내용은 [참조하십시오 585 페이지의 『설치 IBM MQ classes for XMS .NET』](#).

관련 태스크

[문제 해결 정보 수집](#)

[XMS.NET 문제 해결](#)

XMS에서 지원되는 메시징 스타일

XMS에서는 포인트-투-포인트 및 발행/구독 스타일의 메시징을 지원합니다.

메시징 스타일을 메시징 도메인이라고도 합니다.

포인트-투-포인트 메시징

포인트-투-포인트 메시징의 일반 양식에서는 큐잉을 사용합니다. 가장 간단한 경우 애플리케이션은 목적지 큐를 암시적 또는 명시적으로 식별하여 다른 애플리케이션으로 메시지를 전송합니다. 기본 메시징 및 큐잉 시스템은 전송 애플리케이션으로부터 메시지를 수신하고 메시지를 목적지 큐로 라우팅합니다. 그런 다음 수신 애플리케이션이 큐에서 메시지를 검색합니다.

기본 메시징 및 큐잉 시스템에 IBM Integration Bus가 포함되는 경우 IBM Integration Bus는 메시지를 복제하고 메시지 사본을 다른 큐로 라우팅할 수 있습니다. 그러면 둘 이상의 애플리케이션이 메시지를 수신할 수 있습니다. IBM Integration Bus는 메시지를 변환하고 데이터를 여기에 추가할 수도 있습니다.

포인트-투-포인트 메시징의 핵심 특성은 애플리케이션이 메시지를 전송할 때 로컬 큐에 메시지를 배치하는 것입니다. 기본 메시징 및 큐잉 시스템은 메시지를 전송할 목적지 큐를 판별합니다. 기본 애플리케이션은 목적지 큐로부터 메시지를 수신합니다.

발행/구독 메시징

발행/구독 메시징에는 두 유형의 애플리케이션인 발행자와 구독자가 있습니다.

발행자는 발행 메시지 양식으로 정보를 제공합니다. 발행자가 메시지를 발행하면, 이는 메시지 내에서 정보의 토픽을 식별하는 토픽을 지정합니다.

구독자는 발행된 정보의 이용자입니다. 구독자는 구독을 작성하여 관심있는 토픽을 지정합니다.

발행/구독 시스템은 발행자의 발행물 그리고 구독자의 구독을 수신합니다. 이는 구독자에게 발행물을 라우팅합니다. 구독자는 구독하는 해당 토픽에 대한 발행물을 수신합니다.

발행/구독 메시징의 핵심 특성은 발행자가 메시지를 발행할 때 토픽을 식별하는 것입니다. 이는 구독자를 식별하지 않습니다. 메시지가 구독자가 없는 토픽에 대해 발행되면 애플리케이션은 메시지를 수신하지 않습니다.

애플리케이션은 발행자 및 구독자 모두가 될 수 있습니다.

XMS 오브젝트 모델

XMS API는 오브젝트 지향 인터페이스입니다. XMS 오브젝트 모델은 JMS 1.1 오브젝트 모델을 기반으로 합니다.

기본 XMS 클래스

기본 XMS 클래스 또는 오브젝트 유형은 다음과 같습니다.

ConnectionFactory

ConnectionFactory 오브젝트는 연결의 매개변수 세트를 캡슐화합니다. 애플리케이션은 ConnectionFactory를 사용하여 연결을 작성합니다. 애플리케이션은 런타임 시 매개변수를 제공하고 ConnectionFactory 오브젝트를 작성할 수 있습니다. 또는 연결 매개변수가 관리 대상 오브젝트 저장소에 저장될 수 있습니다. 애플리케이션은 저장소에서 오브젝트를 검색하고 여기에서 ConnectionFactory 오브젝트를 작성할 수 있습니다.

연결

Connection 오브젝트는 애플리케이션에서 메시징 서버로서의 활성 연결을 캡슐화합니다. 애플리케이션은 연결을 사용하여 세션을 작성합니다.

목적지

애플리케이션은 Destination 오브젝트를 사용하여 메시지를 전송하고 메시지를 수신합니다. 발행/구독 도메인에서 Destination 오브젝트는 토픽을 캡슐화하고, 포인트-투-포인트 도메인에서는 Destination 오브젝트가 큐를 캡슐화합니다. 애플리케이션은 런타임 시 Destination 오브젝트를 작성하는 매개변수를 제공할 수 있습니다. 또는 관리 대상 오브젝트 저장소에 저장되는 오브젝트 정의에서 Destination 오브젝트를 작성할 수 있습니다.

세션

Session 오브젝트는 메시지 전송 및 수신을 위한 단일 스레드 컨텍스트입니다. 애플리케이션은 Session 오브젝트를 사용하여 Message, MessageProducer 및 MessageConsumer 오브젝트를 작성합니다.

메시지

Message 오브젝트는 애플리케이션이 MessageProducer 오브젝트를 사용하여 전송하거나 MessageConsumer 오브젝트를 사용하여 수신하는 Message 오브젝트를 캡슐화합니다.

MessageProducer

MessageProducer 오브젝트는 애플리케이션이 목적지로 메시지를 전송할 때 사용됩니다.

MessageConsumer

MessageConsumer 오브젝트는 애플리케이션이 목적지로 전송된 메시지를 수신할 때 사용됩니다.

XMS 오브젝트 및 해당 관계

584 페이지의 그림 68에서는 XMS 오브젝트의 기본 유형(ConnectionFactory, Connection, Session, MessageProducer, MessageConsumer, Message 및 Destination)을 표시합니다. 애플리케이션은 연결 팩토리를 사용하여 연결을 작성하고 연결을 사용하여 세션을 작성합니다. 그런 다음 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.

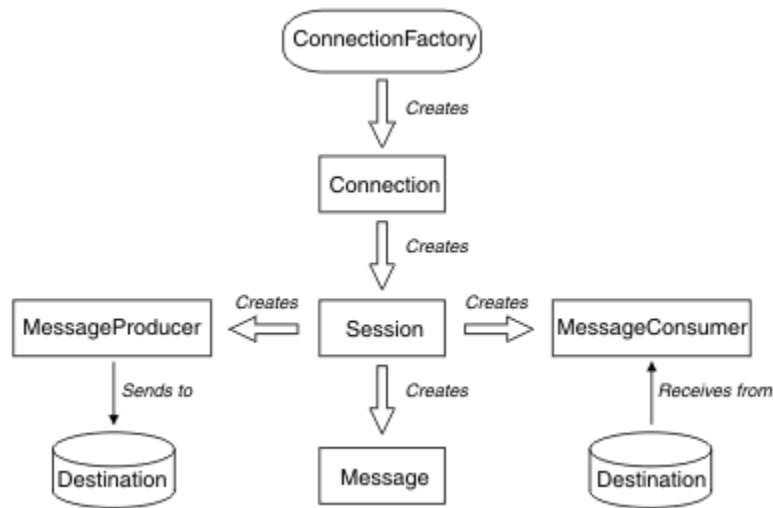


그림 68. XMS 오브젝트 및 해당 관계

XMS .NET에서 XMS 클래스는 .NET 인터페이스 세트로 정의됩니다. XMS .NET 애플리케이션을 코딩하는 경우 선언된 인터페이스만 필요합니다.

XMS 오브젝트 모델은 Java Message Service 스펙, 버전 1.1에서 설명한 도메인 독립 인터페이스를 기반으로 합니다. Topic, TopicPublisher 및 TopicSubscriber와 같은 도메인 특정 클래스는 제공되지 않습니다.

XMS 오브젝트의 속성 및 특성

XMS 오브젝트에는 여러 방식으로 구현되는 오브젝트 특성인 속성 및 특성이 있습니다.

속성

속성에 값이 없는 경우에도 항상 제공되며 스토리지를 사용하는 오브젝트 특성입니다. 이 점에서 속성은 고정 길이 데이터 구조의 필드와 유사합니다. 속성의 고유한 특징은 각 속성에 해당 값을 설정하고 가져오는 데 필요한 자체 메소드가 있다는 점입니다.

특성

값이 설정되어야 오브젝트의 특성이 제공되고 스토리지를 사용합니다. 값을 설정한 후 특성을 삭제하거나 스토리지를 복구할 수 없습니다. 값은 변경할 수 있습니다. XMS에서는 특성 값을 설정 및 가져오기 위한 일반 메소드 세트를 제공합니다.

관리 대상 오브젝트

관리 대상 오브젝트를 사용하여 중앙 저장소에서 관리 대상 클라이언트 애플리케이션이 사용하는 연결 설정을 관리할 수 있습니다. 애플리케이션은 중앙 저장소에서 오브젝트 정의를 검색하고 이를 사용하여 ConnectionFactory 및 Destination 오브젝트를 작성합니다. 관리 대상 오브젝트를 사용하면 런타임 시 사용하는 자원에서 애플리케이션을 커플링 해제할 수 있습니다.

예를 들면 테스트 환경에서 XMS 애플리케이션을 작성하여, 목적지 및 연결 세트를 참조하는 관리 대상 오브젝트와 함께 테스트할 수 있습니다. 애플리케이션이 배치되면 관리 대상 오브젝트는 프로덕션 환경에서 연결 및 목적지를 참조하는 애플리케이션을 구성하도록 변경될 수 있습니다.

XMS는 두 가지 유형의 관리 대상 오브젝트를 지원합니다.

- 애플리케이션이 서버에 대한 초기 연결을 작성하는 데 사용하는 ConnectionFactory 오브젝트.
- Destination 오브젝트는 애플리케이션이 전송되는 메시지의 목적지 및 수신되는 메시지의 소스를 지정할 때 사용합니다. 목적지는 애플리케이션이 연결하는 서버의 토픽이나 큐입니다.

관리 도구 JMSAdmin은 IBM MQ와 함께 제공됩니다. 이 도구는 관리 대상 오브젝트의 중앙 저장소에서 관리 대상 오브젝트(를) 작성하고 관리하는 데 사용됩니다.

저장소에 있는 관리 대상 오브젝트는 IBM MQ classes for JMS 및 XMS 애플리케이션이 사용할 수 있습니다. XMS 애플리케이션은 ConnectionFactory 및 Destination 오브젝트를 사용하여 IBM MQ 큐 관리자에 연

결할 수 있습니다. 관리자는 애플리케이션 코드에 영향을 주지 않고 저장소에 보관된 오브젝트 정의를 변경할 수 있습니다.

다음 다이어그램은 XMS 애플리케이션에서 관리 대상 오브젝트를 사용하는 일반적인 방법을 보여줍니다. 다이어그램의 왼쪽에서는 관리 콘솔을 사용하여 관리하는 ConnectionFactory 및 Destination 오브젝트 정의가 포함되는 저장소를 표시합니다. 다이어그램의 오른쪽은 저장소에서 오브젝트 정의를 검색하고 메시징 서버에 연결할 때 이러한 오브젝트 정의를 사용하는 XMS 애플리케이션을 표시합니다.

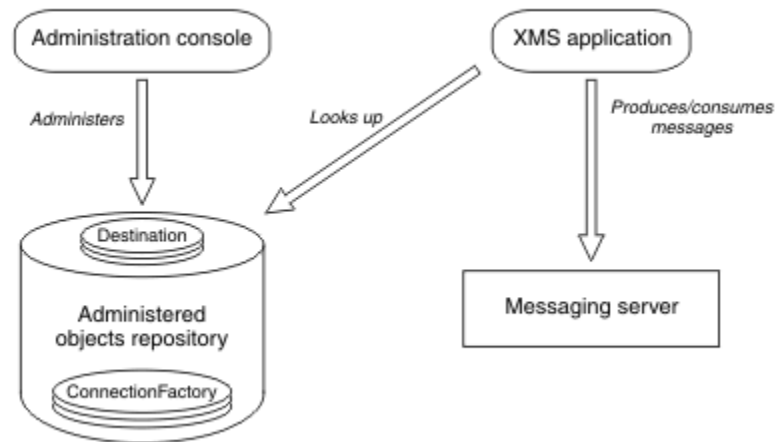


그림 69. XMS 애플리케이션이 일반적으로 관리 대상 오브젝트를 사용하는 방법

XMS 메시지 모델

XMS 메시지 모델은 IBM MQ classes for JMS 메시지 모델과 같습니다.

특히 XMS는 IBM MQ classes for JMS가 구현하는 것과 동일한 메시지 특성과 메시지 헤더 필드를 구현합니다.

- JMS 헤더 필드. 이러한 필드의 이름은 접두부 JMS로 시작합니다.
- JMS 정의 특성. 이러한 필드에는 이름이 접두부 JMSX로 시작하는 특성이 있습니다.
- IBM 정의 특성. 이러한 필드에는 이름이 접두부 JMS_IBM_로 시작하는 특성이 있습니다.

그러므로 XMS 애플리케이션은 IBM MQ classes for JMS 애플리케이션과 메시지를 교환할 수 있습니다. 각 메시지에서 일부 헤더 필드 및 특성은 애플리케이션에서 설정하고 나머지는 XMS 또는 IBM MQ classes for JMS에 의해 설정됩니다. XMS 또는 IBM MQ classes for JMS에서 설정하는 필드 일부는 메시지를 전송할 때 설정되고, 기타는 수신할 때 설정됩니다. 헤더 필드 및 특성은 적절한 경우 메시징 서버를 통해 메시지와 함께 전파됩니다. 이는 메시지를 수신하는 애플리케이션에서 사용할 수 있습니다.

관련 개념

[IBM MQ classes for JMS](#)

Windows

Linux

설치 IBM MQ classes for XMS .NET

IBM MQ classes for XMS .NET 샘플을 포함하여, 에 IBM MQ 설치됩니다 WindowsLinux .

설치

IBM MQ 9.4.0 대상 .NET 6 프레임워크로 를 기반으로 빌드된 클라이언트 XMS.NET 라이브러리를 제공합니다. 에서 IBM MQ 9.4.0, Microsoft .NET 6.0 는 를 타겟 프레임워크로 사용하여 .NET 6 빌드된 라이브러리를 IBM MQ 사용하는 애플리케이션을 실행하는 데 필요한 최소 버전입니다. 대상 프레임워크로 를 .NET 6 사용하여 빌드된 클라이언트 XMS.NET 라이브러리는 에서 WindowsMQ_INSTALLATION_PATH/bin 로, 에서 MQ_INSTALLATION_PATH/lib64 Linux 로 제공됩니다.

, 에서 IBM MQ 9.4.0, IBM MQ classes for XMS .NET 를 사용하는 IBM MQ 애플리케이션을 .NET 8 지원합니다. 해당 .NET 6 애플리케이션을 사용 중이라면, 파일을 runtimeconfig 약간 수정하여 값을

targetframeworkversion 로 설정함으로써 재컴파일 없이도 이 "net8.0" 애플리케이션을 실행할 수 있습니다.

에서 IBM MQ 10.0.0, 런타임 시 를 IBM MQ 사용하는 .NET 10 애플리케이션에 대한 IBM MQ classes for XMS .NET 를 지원합니다. 새로운 런타임 또는 컴포넌트에서 실행될 때 기존 바이너리는 로드 또는 실행 실패와 같은 동작상의 중대한 변경 사항을 겪을 수 있으며, 재컴파일이 필요할 수 있습니다. 또는 .NET 8.NET 6 애플리케이션을 사용 중인 경우, 파일을 runtimeconfig 약간 수정하여 값을 targetframeworkversion 로 설정하면 재컴파일 없이도 이 "net10.0" 애플리케이션을 실행할 수 있습니다.

 IBM MQ classes for XMS .NET는 IBM MQ 10.0.0 를 IBM MQ 대상으로 하는 애플리케이션을 위해 .NET 10 기반으로 구축된 클라이언트 .NET 10 라이브러리를 XMS 제공합니다. "net10.0" 또는 .NET 8 애플리케이션을 .NET 6 사용 중인 경우, 파일에서 runtimeconfig 값을 targetframeworkversion 로 설정하도록 약간만 수정하면 재컴파일 없이도 이 애플리케이션을 실행할 수 있습니다.

  부터 IBM MQ 9.4.0, 에 IBM MQ classes for XMS .NET, 데이터 직렬화 및 역직렬화에 사용되는 메서드 WriteObject(), ReadObject(), CreateObjectMessage(), 그리고 클래스 ObjectMessage 및 XmsObjectMessageImpl 는 더 이상 사용되지 않습니다.

 사용하여 빌드된 클라이언트 XMS.NET.NET Standard 2.0 라이브러리는 에 사용 IBM MQ 9.3.1 중단되었으며, 에 제품에서 IBM MQ 9.4.0 제거되었습니다.

amqmxmsstd.dll 라이브러리

다음 위치에서 를 대상으로 하는 IBM MQ 9.4.0 .NET 6 프레임워크로 구축된 amqmxmsstd.dll 라이브러리를 사용할 수 있습니다:

-  날짜 Windows: MQ_INSTALLATION_PATH\bin. 샘플 애플리케이션은 MQ_INSTALLATION_PATH\samp\dotnet\samples\cs\core\base에 설치됩니다.
-  날짜 Linux: MQ_INSTALLATION_PATH\lib64. 샘플들은 .NET . MQ_INSTALLATION_PATH\samp\dotnet\samples\cs\core\base에 있습니다.

에서 IBM MQ 9.4.4, 애플리케이션에서 XMS.NET 6 사용하는 TLS 인증서를 위한 사용자 지정 경로를 지정할 수 있습니다.

에서 비추천된 로 .NET Standard 2.0 구축된 IBM MQ 9.3.1 클라이언트 XMS.NET 라이브러리는 에 제품에서 IBM MQ 9.4.0 제거되었습니다.



주의: 부터 IBM MQ 9.4.0, 를 대상 XMS.NET.NET Standard 2.0 프레임워크로 사용하여 빌드된 클라이언트 라이브러리가 제거됩니다. 이 라이브러리들은 에 사용 IBM MQ 9.3.1 중단되었습니다.

 모든 IBM.XMS.* 라이브러리가 여전히 제공되지만 이러한 라이브러리는 안정화되어 있습니다. 즉, 새 기능이 도입되지 않습니다. 최신 기능을 위해서는 amqmxmsstd.dll 라이브러리로 마이그레이션해야 합니다. 그러나 IBM MQ 9.1 Long Term Support 또는 Continuous Delivery 릴리스에서 기존 라이브러리를 계속 사용할 수 있습니다.

라이브러리 netstandard2.0 제거 후 발생할 수 있는 두 가지 시나리오는 다음과 같습니다:

- 애플리케이션이 라이브러리(예 netstandard2.0 : amqmdnetstd.dll)를 IBM MQ classes for XMS .NET Framework 사용하여 빌드된 경우, 애플리케이션이 성공적으로 실행되도록 하려면 Microsoft.NET Framework 4.7.2 라이브러리(예: amqmdnet.dll)로 애플리케이션을 재빌드해야 합니다. 애플리케이션을 재빌드하지 않으면 " System.IO.Unexceptionable " 메시지가 발생할 수 있습니다:

예외 발생: System.IO.FileLoadException: 파일 또는 어셈블리 'amqmdnetstd, Version=9.3.5.0, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e ' 또는 그 종속성 중 하나를 로드할 수 없습니다. 찾은 어셈블리의 매니페스트 정의가 어셈블리 참조와 일치하지 않습니다. (0x80131040 예외 발생)
파일 이름: 'amqmdnetstd, Version=9.3.5.0, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e '
(SimplePut.SimplePut.PutMessages()
C:\SampleCode\Program 파일의 132번째 줄에 있는 SimplePut.SimplePut (String[] args)에서 오류가 발생했습니다.

- 해당 .NET 6 애플리케이션이 netstandard2.0 라이브러리로 빌드된 경우, 애플리케이션 런타임 디렉토리의 폴더에 bin 있는 동일한 .NET 6 라이브러리로 교체하기만 하면 됩니다. 재구축이 필요하지 않습니다.

참고: 대체 .NET 6 라이브러리는 항상 대체되는 netstandard2.0 라이브러리와 동일하거나 더 높은 수준이어야 합니다.

해당 파일들은 저장소에서 NuGet IBM MQ classes for XMS .NET Standard 다운로드할 수 있습니다. NuGet 패키지는 amqmxmsstd.dll 라이브러리와 amqmdnetstd.dll 라이브러리를 모두 포함합니다. amqmxmsstd.dll 에 amqmdnetstd.dll 의존하며, Core XMS .NET 애플리케이션을 패키징할 때 와 amqmxmsstd.dll 모두 Core amqmdnetstd.dll XMS .NET 애플리케이션과 함께 패키징되어야 합니다. 추가 정보는 589 페이지의 『NuGet 저장소에서 IBM MQ classes for XMS .NET 다운로드』의 내용을 참조하십시오.

dspmqver 명령

dspmqver 명령을 사용하여 .NET Core 컴포넌트에 대한 버전 및 빌드 정보를 표시할 수 있습니다.

와 IBM MQ classes for XMS .NET Framework IBM MQ classes for XMS .NET.NET 6 라이브러리 비교

다음 표는 의 기능을 및 .NET 6 IBM MQ classes for XMS .NET Framework IBM MQ classes for XMS .NET 라이브러리의 기능과 비교하여 나열합니다.

표 82. 와 IBM MQ classes for XMS .NET Framework 의 IBM MQ classes for XMS .NET 차이점		
기능	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET
클래스 이름(API)	모든 클래스는 각 네트워크에서 동일하게 유지됩니다.	모든 클래스는 각 네트워크에서 동일하게 유지됩니다.
운영 체제	Windows	Windows 도커화된(Dockerized) 컨테이너 Linux macOS
app.config 파일(재배포 가능 클라이언트에서 추적을 사용으로 설정하기 위한 구성 파일)	app.config file은 재배포 가능 패키지에 대한 추적을 활성화하는 데 사용됩니다.	app.config은(는) 지원되지 않습니다. 환경 변수를 사용하십시오.

표 82. 와 IBM MQ classes for XMS .NET Framework 의 IBM MQ classes for XMS .NET 차이점 (계속)

기능	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET
추적	클라이언트를 XMS .NET 추적하려면 기존 환경 변수를 사용할 수 있습니다. 예를 들어 추적을 활성화하는 데 사용되는 환경 XMS_TRACE_ON 변수 등이 있습니다. 자세한 내용은 환경 변수를 XMS 사용한 추적 XMS .NET 구성을 참조하십시오. 재배포 가능한 클라이언트의 경우, 파일을 app.config 사용하여 추적을 활성화할 수 있습니다. 에서 IBM MQ 9.4.0, 파일을 mqclient.ini 사용하여 추적을 활성화하거나 비활성화하고 Trace 섹션의 적절한 속성을 설정할 수 있습니다. 추적 기능을 동적으로 활성화하거나 비활성화하려면 파일을 mqclient.ini 사용하십시오. 자세한 내용은 을 참조하십시오 애플리케이션 XMS .NET 추적 mqclient.ini.	클라이언트를 XMS .NET 추적하려면 기존 환경 변수를 사용할 수 있습니다. 예를 들어 추적을 활성화하는 데 사용되는 환경 XMS_TRACE_ON 변수 등이 있습니다. 자세한 내용은 환경 변수를 XMS 사용한 추적 XMS .NET 구성을 참조하십시오. 에서 IBM MQ 9.4.0, 파일을 mqclient.ini 사용하여 추적을 활성화하거나 비활성화하고 Trace 섹션의 적절한 속성을 설정할 수 있습니다. 추적 기능을 동적으로 활성화하거나 비활성화하려면 파일을 mqclient.ini 사용하십시오. 자세한 내용은 을 참조하십시오 애플리케이션 XMS .NET 추적 mqclient.ini.
전송 모드	관리, 비관리 및 바인딩	관리됨
TLS	Windows 키 저장소가 인증서를 저장하는 데 사용됩니다.	Windows Windows에서는 인증서를 저장하는 데 키 저장소를 사용해야 합니다. 허용되는 값은 *USER 또는 *SYSTEM입니다. 입력을 기반으로 IBM MQ .NET 클라이언트에서 시스템 전체 또는 현재 사용자의 Windows 키 저장소를 확인합니다. Linux Linux에서는 X509Store 클래스를 사용하여 인증서를 설치하는 것이 좋으며 .NET Core에서는 ".dotnet/corefx/cryptography/x509stores" 위치에 인증서를 설치합니다. V 10.0.0 TLS 인증서를 저장할 사용자 지정 경로를 추가할 수 있으며 IBM MQ 10.0.0, 인증서에 대한 암호화된 암호를 생성하기 위해 일반 텍스트 암호 또는 Dotnet Password Encryptor(rundotnetcred) 도구를 사용할 수 있습니다. 자세한 내용은 TLS 연결 시 일반 텍스트 또는 암호화된 암호를 사용하는 경우 XMS.NET 의 인증서 사용자 지정 경로를 참조하십시오.
CCDT	지원됨	지원됨, CCDT 경로 설정은 .NET Framework 클래스와 동일합니다.
클라이언트 자동 다시 연결	지원됨	지원됨

표 82. 와 IBM MQ classes for XMS .NET Framework 의 IBM MQ classes for XMS .NET 차이점 (계속)		
기능	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET
분배 트랜잭션	지원됨	지원되지 않음
글로벌 어셈블리 캐시 (GAC)로 동적 링크 라이브러리(dll)의 설치	Dll은 IBM MQ 설치의 일부로 GAC에 설치됩니다.	Dll은 IBM MQ 설치의 일부로 설치되지 않습니다.
WMQ, WPM 및 RTT 연결 유형에 지원	WMQ, WPM 및 RTT 연결 유형 지원	WMQ에서만 지원
JNDI 관리 오브젝트	LDAP 및 파일 시스템 지원	FileSystem만 지원

V4.7.2 이상 버전을 Microsoft.NET Framework 설치해야 IBM MQ 9.3.0 실행할 IBM MQ classes for XMS .NET Framework 수 있습니다.

관련 태스크

595 페이지의 『XMS 샘플 애플리케이션 사용』

XMS .NET 샘플 애플리케이션은 각 API의 공통 기능 개요를 제공합니다. 이를 사용하여 설치 및 메시징 서버 설정을 확인하고 자신만의 애플리케이션을 빌드하는 데 도움을 받을 수 있습니다.

Windows Linux NuGet 저장소에서 IBM MQ classes for XMS .NET 다운로드

IBM MQ classes for XMS .NET은(는) NuGet 저장소에서 다운로드가 가능하기 때문에 .NET 개발자가 쉽게 이용할 수 있습니다.

이 태스크 정보

NuGet 개발 플랫폼을 Microsoft 위한 패키지 .NET관리자입니다. NuGet 클라이언트 도구는 패키지를 생성하고 이용하는 기능을 제공합니다. NuGet 패키지는 컴파일된 코드(DLL)를 포함한 .nupkg 확장이 있는 단일 압축 파일, 해당 코드에 관련된 기타 파일 및 패키지의 버전 번호와 같은 정보를 포함한 기술적 Manifest입니다.

모든 패키지 작성자와 사용자가 이용하는 중앙 패키지 저장소인 NuGet 갤러리에서 amqmdnetstd.dll 라이브러리와 amqmxmsstd.dll 라이브러리를 모두 포함하는 패키지를 IBMXMSDotnetClientNuGet 다운로드할 수 있습니다.

참고: 패키지는 NuGet 를 IBM MQ 9.4.0 대상으로 하는 .NET 6 프레임워크로 빌드된 라이브러리를 포함합니다.

Removed 에서 사용 중단된 로 .NET Standard 2.0 구축된 IBM MQ 9.3.1클라이언트 XMS .NET 라이브러리는 에서 제품에서 IBM MQ 9.4.0제거되었습니다.

, 에서 IBM MQ 9.4.0, IBM MQ classes for XMS .NET 를 사용하는 IBM MQ 애플리케이션을 .NET 8 지원합니다. 해당 .NET 6 애플리케이션을 사용 중이라면, 파일을 runtimeconfig 약간 수정하여 값을 targetframeworkversion 로 설정함으로써 재컴파일 없이도 이 "net8.0"애플리케이션을 실행할 수 있습니다.

IBMXMSDotnetClient 패키지를 다운로드하는 세 가지 방법이 있습니다.

- Microsoft Visual Studio를 사용합니다. NuGet은 Microsoft Visual Studio 확장으로서 배포됩니다. Microsoft Visual Studio 2012부터는 NuGet이 기본적으로 사전 설치됩니다.
- 명령행에서 NuGet 패키지 관리자 또는 .NET CLI를 사용합니다.
- 웹 브라우저를 사용합니다.

재배포가능 패키지의 경우에는 환경 변수 **XMS_TRACE_ON**을 사용하여 추적을 사용으로 설정합니다.

프로시저

- Microsoft Visual Studio에서 Package Manager UI를 사용하여 IBMXMSDotnetClient 패키지를 다운로드하려면 다음 단계를 완료하십시오.

- a) .NET 프로젝트를 마우스 오른쪽 단추로 클릭한 후 **Nuget 패키지 관리**를 클릭하십시오.
 - b) **찾아보기** 탭을 클릭하고, "IBMXMSDotnetClient"를 검색하세요.
 - c) 해당 패키지를 선택하고 **설치**를 클릭하십시오.
- 설치 중에, 패키지 관리자는 콘솔에 표시되는 문장의 형태로 진행상태 정보를 제공합니다.
- 명령행에서 IBMXMSDotnetClient 패키지를 다운로드하려면 다음 옵션 중 하나를 선택하십시오.
 - NuGet 패키지 관리자에서 다음 명령을 입력하십시오.

```
Install-Package IBMXMSDotnetClient -Version 9.1.4.0
```

설치 중에, 패키지 관리자는 콘솔에 표시되는 문장의 형태로 진행상태 정보를 제공합니다. 이 출력은 로그 파일로 경로 재지정할 수 있습니다.

- .NET CLI를 사용하여 다음 명령을 입력하십시오.

```
dotnet add package IBMXMSDotnetClient --version 9.1.4
```

- 웹 브라우저를 사용하여 <https://www.nuget.org/packages/IBMXMSDotnetClient>에서 IBMXMSDotnetClient 패키지를 다운로드하십시오.

관련 개념

521 페이지의 『설치 IBM MQ classes for .NET』

IBM MQ classes for .NET 샘플을 포함한 모든 구성 요소가 에 IBM MQ 설치됩니다 Windows . Linux

IBM MQ 라이선스 정보 .NET 클라이언트

관련 태스크

526 페이지의 『저장소에서 NuGet IBM MQ classes for .NET 다운로드 중』

IBM MQ classes for .NET 파일들은 저장소에서 NuGet 다운로드 가능하므로 개발자들이 .NET 쉽게 활용할 수 있습니다.

메시징 서버 환경 설정

이 절의 주제에서는 XMS 애플리케이션이 서버에 연결할 수 있도록 메시징 서버 환경을 설정하는 방법에 대해 설명합니다.

이 태스크 정보

IBM MQ 큐 관리자에 연결하는 애플리케이션의 경우, IBM MQ 클라이언트(또는 바인딩 모드의 경우 큐 관리자)가 필요합니다.

현재, 브로커에 대한 실시간 연결을 사용하는 애플리케이션의 경우 전제조건이 없습니다.

XMS에서 제공하는 샘플 애플리케이션을 포함하여 XMS 애플리케이션을 실행하기 전에 메시징 서버 환경을 설정해야 합니다.

이 절에는 다음 주제가 포함되어 있습니다.

- 592 페이지의 『IBM MQ 큐 관리자에 연결하는 애플리케이션의 큐 관리자 및 브로커 구성』
- 585 페이지의 『설치 IBM MQ classes for XMS .NET』
- 594 페이지의 『브로커에 대한 실시간 연결을 사용하는 애플리케이션의 브로커 구성』
- 594 페이지의 『WebSphere Application Server에 연결하는 애플리케이션의 서비스 통합 버스 구성』

XMS .NET의 메시지 리스너

메시지 리스너는 메시지를 비동기적으로 수신하는 데 사용됩니다. 콜과 `MessageConsumer.receive()` 달리, 메시지 리스너는 호출 스레드를 차단하지 않고 대신 지정된 콜백 메서드(일반적으로 `onMessage()` `onMessage` 메서드)로 메시지를 전달합니다.

메시지 전달은 `Connection.Start()` 메서드가 호출되는 즉시 시작됩니다. 메시지 전달은 각각 및 `Connection.Start()` `Connection.Stop()` 메서드를 사용하여 언제든지 중지하거나 재개할 수 있습니다.

세션 내 최소 한 명의 소비자에게 메시지 리스너를 설정한 후 해당 `Connection.Start()` 메서드가 호출되면, 해당 세션은 비동기 세션이 됩니다. 세션이 비동기적으로 전환되면 동기식 XMS.NET 메서드를 호출할 수 없습니다. 예: `MessageProducer.Send()`. 이렇게 하면 예외가 발생하며, 그 IBM MQ 사유 코드는 `MQRC_HCONN_ASYNC_ACTIVE(2500)`입니다.

비동기 세션 내의 동기식 호출

`Session.Close` 비동기 세션에서 허용되는 유일한 동기식 호출입니다. 애플리케이션은 메시지 리스너 콜백 메서드, 즉 `onMessage` 메서드를 사용하여 동기식 호출(를 제외하고 `Session.Close`)도 수행할 수 있습니다.

이 두 가지 옵션 외에는 애플리케이션이 동기식 호출을 수행하려면 반드시 `Connection.Stop()` 메서드를 사용하여 연결을 중지해야 합니다. 통화가 완료된 후에는 `Connection.Start()` 메서드를 사용하여 연결을 다시 재개해야 합니다. 메시지 전달을 재시작합니다.

세션은 몇 개의 비동기 메시지 소비자(consumer)를 가질 수 있나요?

세션은 여러 비동기 메시지 소비자들을 가질 수 있습니다. 그러나 어떤 시점에서도 메시지는 단 하나의 소비자만에게 전달됩니다. 이것이 실질적으로 의미하는 바는, 첫 번째 메시지를 전달하기 위해 컨슈머의 `onMessage()` 메서드를 호출한 XMS.NET 상태에서 두 번째 메시지가 도착할 경우, `onMessage()` 메서드가 반환될 때까지 해당 세션 내 컨슈머에게 두 번째 메시지가 전달되지 않는다는 것입니다.

두 번째 메시지는 해당 `onMessage()` 메서드가 반환된 후에야 세션 내의 소비자에게 전달됩니다. 이는 세션이 단일 스레드만을 사용하여 소비자에게 메시지 전달을 관리하기 때문입니다. 이는 한 번에 하나의 메시지만 전달될 수 있으며, 수신자는 그 중 어느 것이든 될 수 있음을 의미합니다.

응용 프로그램이 동시 메시지 전달을 요구하는 경우, 즉 모든 소비자(consumer)가 동시에 메시지를 수신해야 하는 경우, 해당 응용 프로그램은 여러 세션을 생성해야 하며 각 세션에는 하나의 비동기 메시지 소비자(asynchronous message consumer)가 있어야 합니다.

다음 예시들은 이 기능을 더 명확하게 보여줍니다.

첫 번째 예시에서는 세션 내에 여러 비동기 메시지 소비자(asynchronous message consumers)가 존재합니다. 세션 S에는 세 개의 비동기 메시지 소비자(AMC1, AMC2, AMC3)가 있으며, 이들은 각각 다른 세 개의 목적지(Q1, Q2, Q3)로부터 메시지를 수신합니다.

세션이 S 하나뿐이므로, 소비자 AMC1, AMC2, 및 로 메시지를 전달할 메시지 AMC3 전달 스레드도 하나뿐입니다. 세션이 메시지를 전달할 때, 다른 두 소비자 AMC2, AMC1 와 는 와 Q2 에 전달 Q3 AMC3 준비가 된 메시지가 있더라도 대기합니다.

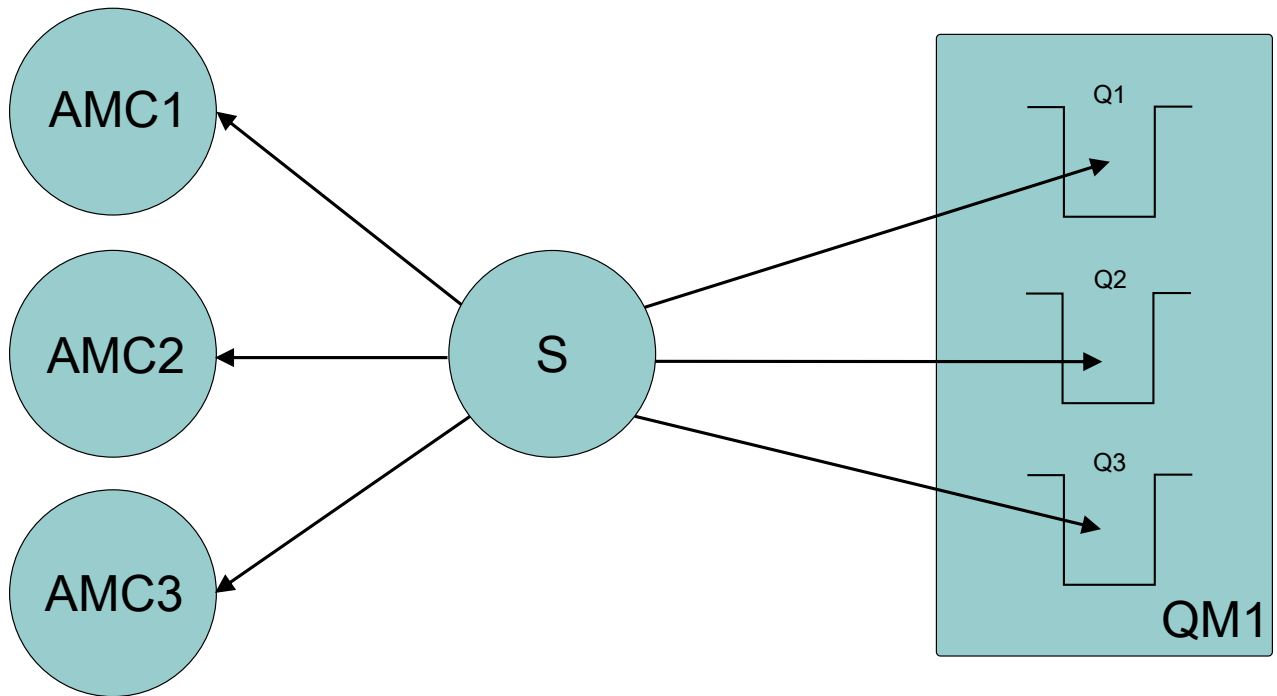


그림 70. 하나의 세션에 세 개의 비동기 메시지 소비자

두 번째 경우에는, S2, 등 여러 세션이 S3 S1 존재하며, 각각 하나의 비동기 메시지 소비자 AMC1, AMC2, 를 AMC3 각각 보유하고 있습니다. 각 세션마다 하나의 소비자(consumer)가 존재하므로, 메시지는 소비자들에게 동시적으로 전달됩니다.

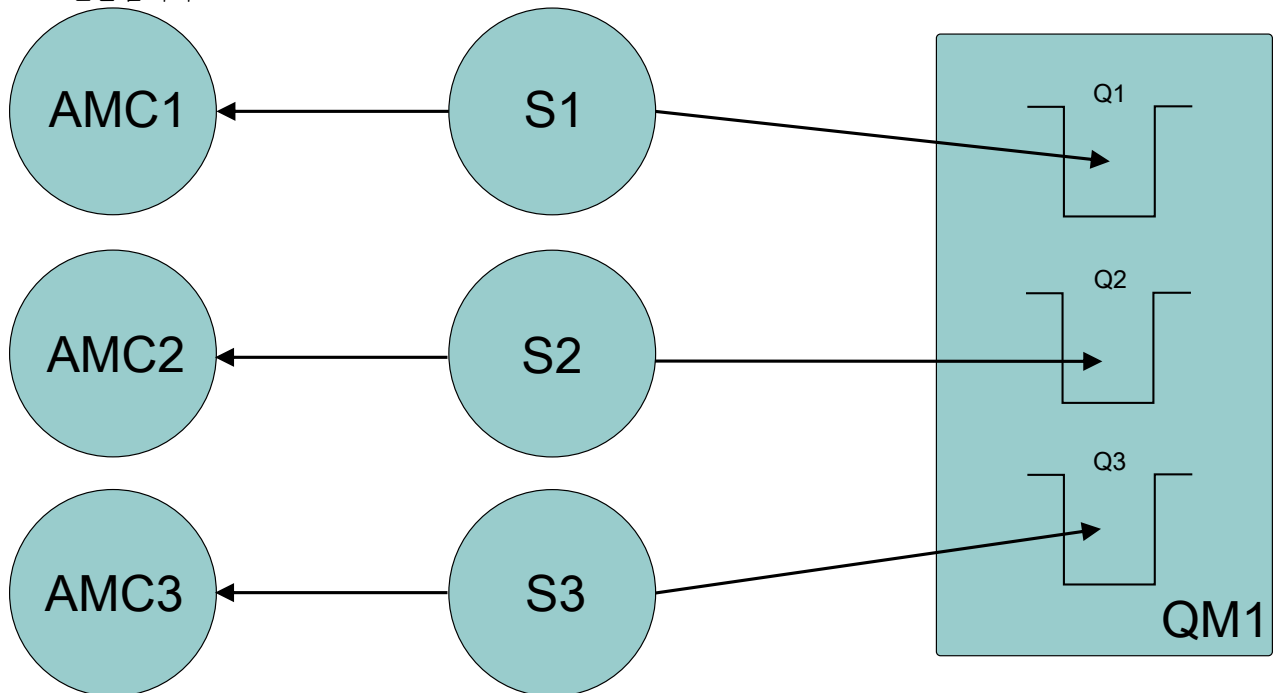


그림 71. 여러 세션, 각 세션마다 하나의 비동기 메시지 소비자

이는 동시 메시지 전달이 필요한 경우 여러 세션이 필요함을 보여줍니다.

IBM MQ 큐 관리자에 연결하는 애플리케이션의 큐 관리자 및 브로커 구성

이 절에서는 사용자가 IBM WebSphere MQ 7.0.1 이상을 사용 중인 것으로 가정합니다. IBM MQ 큐 관리자에 연결하는 애플리케이션을 실행하려면 큐 관리자를 구성해야 합니다. 발행/구독 애플리케이션의 경우, 큐에 지정된 발행/구독 인터페이스를 사용 중인 경우 몇 가지 추가 구성이 필요합니다.

시작하기 전에

XMS가 IBM Integration Bus 또는 WebSphere Message Broker 6.1 이상과 함께 사용되고 있습니다.

이 태스크를 시작하기 전에 다음 단계를 수행하십시오.

- 애플리케이션이 실행 중인 큐 관리자에 액세스할 수 있는지 확인하십시오.
- 애플리케이션이 발행/구독 애플리케이션이고 큐 지정된 발행/구독 인터페이스를 사용하는 경우, **PSMODE** 속성이 큐 관리자에서 **ENABLED**로 설정되어 있는지 확인하십시오.
- 애플리케이션이 큐 관리자에 연결하도록 특성이 적절히 설정되어 있는 연결 팩토리를 사용하는지 확인하십시오. 애플리케이션이 발행/구독 애플리케이션인 경우, 브로커 사용에 적합하도록 애플리케이션 연결 팩토리 특성이 설정되어 있는지 확인하십시오. 연결 팩토리 특성에 대한 자세한 정보는 [ConnectionFactory](#) 특성을 참조하십시오.

이 태스크 정보

IBM MQ JMS 애플리케이션을 실행하도록 큐 관리자 및 큐 지정된 발행/구독 인터페이스를 구성하는 것과 동일한 방식으로 XMS 애플리케이션을 실행하도록 큐 관리자 및 브로커를 구성합니다. 다음은 수행해야 하는 작업을 요약한 단계입니다.

프로시저

1. 큐 관리자에서 애플리케이션에 필요한 큐를 작성하십시오.

큐를 작성하는 방법에 대한 개요는 [큐 정의](#)를 참조하십시오.

애플리케이션이 발행/구독 애플리케이션이고 IBM MQ classes for JMS 시스템 큐에 액세스해야 하는 큐 지정된 발행/구독 인터페이스를 사용하는 경우 큐를 작성하기 전에 [4a](#)단계까지 대기하십시오.

2. 애플리케이션과 연관된 사용자 ID에 큐 관리자에 연결할 수 있는 권한과 큐에 액세스하기에 적합한 권한을 부여하십시오.

권한 부여 개요는 [보안](#)을 참조하십시오. 애플리케이션이 클라이언트 모드에서 큐 관리자에 연결하는 경우 [클라이언트 및 서버](#)도 참조하십시오.

3. 애플리케이션이 클라이언트 모드의 큐 관리자에 연결하는 경우, 서버 연결 채널이 큐 관리자에 정의되어 있는지 그리고 리스너가 시작되었는지 확인하십시오.

큐 관리자에 연결하는 각 애플리케이션에 대해 이 단계를 수행할 필요가 없습니다. 하나의 서버 연결 채널 정의와 하나의 리스너가 클라이언트 모드에서 연결되는 모든 애플리케이션을 지원할 수 있습니다.

4. 애플리케이션이 발행/구독 애플리케이션이고 큐에 지정된 발행/구독 인터페이스를 사용하는 경우 다음 단계를 수행하십시오.

- a) 큐 관리자에 대해, 함께 제공되는 MQSC 명령 스크립트를 IBM MQ 실행하여 시스템 IBM MQ classes for JMS 큐를 생성하십시오. IBM Integration Bus 또는 WebSphere Message Broker와 연관된 사용자 ID에 큐 액세스 권한이 있는지 확인하십시오.

스크립트 찾는 위치 및 실행 방법에 대한 정보는 [사용 IBM MQ classes for Java](#)의 내용을 참조하십시오.

큐 관리자에 대해 한 번만 이 단계를 수행하십시오. 동일한 IBM MQ classes for JMS 시스템 큐 세트는 큐 관리자에 연결하는 모든 XMS 및 IBM MQ classes for JMS 애플리케이션을 지원할 수 있습니다.

- b) 애플리케이션과 연관되는 사용자 ID에 IBM MQ classes for JMS 시스템 큐에 액세스하는 권한을 부여하십시오.

사용자 ID에게 필요한 권한 정보는 [사용 IBM MQ classes for JMS](#)을 참조하십시오.

- c) IBM Integration Bus 또는 WebSphere Message Broker 브로커의 경우 메시지 플로우를 작성하고 전개하여 애플리케이션이 발행하는 메시지를 전송하는 큐에 서비스하십시오.

기본 메시지 플로우는 발행된 메시지를 읽기 위한 MQInput 메시지 처리 노드와 메시지를 발행하기 위한 Publication 메시지 처리 노드를 포함합니다.

메시지 흐름을 생성하고 배포하는 방법에 대한 정보는 [에서 제공되는 또는 WebSphere Message Broker 제품 IBM Integration Bus](#) [IBM Integration Bus 제품 설명서 라이브러리 웹 페이지 설명서](#)를 참조하십시오.

적합한 메시지 플로우가 이미 브로커에 배치된 경우 이러한 단계를 수행할 필요가 없습니다.

결과

이제 애플리케이션을 시작할 수 있습니다.

브로커에 대한 실시간 연결을 사용하는 애플리케이션의 브로커 구성

브로커에 대한 실시간 연결을 사용하는 애플리케이션을 실행하려면 해당 브로커를 구성해야 합니다.

시작하기 전에

이 태스크를 시작하기 전에 다음 단계를 수행하십시오.

- 애플리케이션이 실행 중인 브로커에 액세스 권한을 가지고 있는지 확인하십시오.
- 애플리케이션이 브로커에 대한 실시간 연결에 적합하도록 특성이 설정되어 있는 연결 팩토리를 사용하는지 확인하십시오. 연결 팩토리 특성에 대한 자세한 정보는 [ConnectionFactory](#) 특성을 참조하십시오.

이 태스크 정보

XMS 애플리케이션을 실행하도록 브로커를 구성하는 것과 동일한 방식으로 IBM MQ classes for JMS 애플리케이션을 실행하도록 브로커를 구성하십시오. 다음은 수행해야 할 작업을 요약한 단계입니다.

프로시저

1. 브로커가 청취하고 메시지를 발행하는 TCP/IP에서 메시지를 읽는 메시지 플로우를 작성 및 배치합니다.

다음 방법 중 하나로 이를 수행할 수 있습니다.

- **Real-timeOptimizedFlow** 메시지 처리 노드를 포함하는 메시지 플로우를 작성합니다.
- **Real-timeInput** 메시지 처리 노드 및 Publication 메시지 처리 노드를 포함하는 메시지 플로우를 작성합니다.

실시간 연결에 사용되는 포트를 청취하도록 **Real-timeOptimizedFlow** 또는 **Real-timeInput** 노드를 구성해야 합니다. XMS에서 실시간 연결을 위한 기본 포트 번호는 1506입니다.

적합한 메시지 플로우가 이미 브로커에 배치된 경우 이러한 단계를 수행할 필요가 없습니다.

2. IBM MQ classes for JMS를 사용하여 애플리케이션에 메시지를 전달해야 할 경우 멀티캐스트가 가능하도록 브로커를 구성하십시오. 신뢰할 수 있는 멀티캐스트가 필요한 해당 토픽의 신뢰 가능한 서비스 품질(QoS)을 지정하여 멀티캐스트가 가능해야 하는 토픽을 구성하십시오.
3. 브로커에 연결할 때 애플리케이션이 사용자 ID와 비밀번호를 제공하고 이러한 정보를 이용하여 브로커가 애플리케이션을 인증할 수 있도록 하려면, 단순한 텔넷과 유사한 비밀번호 인증에 적합하도록 사용자 이름 서버와 브로커를 구성하십시오.

결과

이제 애플리케이션을 시작할 수 있습니다.

WebSphere Application Server에 연결하는 애플리케이션의 서비스 통합 버스 구성

WebSphere Application Server service integration technologies 서비스 통합 버스에 연결하는 애플리케이션을 실행하려면, 기본 메시징 제공자를 사용하는 JMS 애플리케이션을 실행하도록 서비스 통합 버스를 구성하는 것과 동일한 방식으로 서비스 통합을 구성해야 합니다.

시작하기 전에

이 태스크를 시작하기 전에 다음 단계를 수행해야 합니다.

- 메시징 버스가 작성되고 서버가 버스 멤버로서 버스에 추가되는지 확인하십시오.
- 애플리케이션에 실행 중인 하나 이상의 메시징 엔진이 있는 서비스 통합 버스에 대한 액세스 권한이 있는지 확인하십시오.

- HTTP 조작이 필요할 경우 HTTP 메시징 엔진 인바운드 전송 채널을 정의해야 합니다. 기본적으로 SSL 및 TCP에 대한 채널은 서버 설치 도중 정의됩니다.
- 애플리케이션이 부트스트랩 서버를 사용하여 서비스 통합 버스에 연결하도록 특성이 적절히 설정되어 있는 연결 팩토리를 사용하는지 확인하십시오. 필요한 최소 정보는 다음과 같습니다.
 - 메시징 서버로의 연결을 조정할 때 사용하는 위치와 프로토콜에 대해 설명하는 제공자 엔드포인트입니다 (즉, 부트스트랩 서버를 통해 조정). 가장 단순한 양식에서 기본 설정으로 설정된 서버의 경우, 서버의 호스트 이름으로 제공자 엔드포인트를 설정할 수 있습니다.
 - 메시지를 전송하는 버스 이름.

연결 팩토리 특성에 대한 자세한 정보는 [ConnectionFactory](#) 특성을 참조하십시오.

이 태스크 정보

필요한 큐 또는 토픽 공간을 정의해야 합니다. 기본적으로 `Default.Topic.Space` 토픽 공간이 서버 설치 도중 정의되지만 추가 토픽 공간이 필요한 경우 스스로 이러한 토픽 공간을 작성해야 합니다. 서버는 필요에 따라 동적으로 이러한 개별 토픽을 인스턴스화하므로 토픽 공간 내에서 개별 토픽을 사전 정의할 필요는 없습니다.

다음은 수행해야 하는 작업을 요약한 단계입니다.

프로시저

1. 애플리케이션에서 포인트-투-포인트 메시징에 필요한 큐를 작성하십시오.
2. 애플리케이션에서 발행/구독 메시징에 필요한 추가 토픽 공간을 작성하십시오.

결과

이제 애플리케이션을 시작할 수 있습니다.

XMS 샘플 애플리케이션 사용

XMS .NET 샘플 애플리케이션은 각 API의 공통 기능 개요를 제공합니다. 이를 사용하여 설치 및 메시징 서버 설정을 확인하고 자신만의 애플리케이션을 빌드하는 데 도움을 받을 수 있습니다.

이 태스크 정보

자체 애플리케이션을 작성하는 데 도움이 필요하다면 시작점으로서 샘플 애플리케이션을 사용할 수 있습니다. 각 애플리케이션에 대해 소스 버전과 컴파일 버전 모두가 제공됩니다. 샘플 소스 코드를 검토하고 애플리케이션의 필수 오브젝트(`ConnectionFactory`, `Connection`, `Session`, `Destination` 및 `Producer` 또는 `Consumer` 또는 모두)를 작성하고 애플리케이션의 작업 방법을 지정하는 데 필요한 특정 특성을 설정하는 핵심 단계를 식별하십시오. 자세한 정보는 598 페이지의 『XMS .NET 애플리케이션 작성』의 내용을 참조하십시오. 샘플은 XMS의 이후 릴리스에서 변경됩니다.

다음 표는 XMS에서 제공되는 샘플 애플리케이션 세트(각 API당 하나)를 표시합니다.

표 83. XMS .NET용 샘플 애플리케이션	
샘플 이름	설명
SampleConsumerCS	큐에서 메시지를 가져오거나 토픽을 구독하는 메시지 이용자 애플리케이션.
SampleProducerCS	큐 또는 토픽에서 메시지를 생성하는 메시지 생성자 애플리케이션.
SampleConfigCS	파일 기반의 관리 대상 오브젝트 저장소를 작성하는 데 사용할 수 있는 구성 애플리케이션. 애플리케이션에는 특정 연결 설정의 연결 팩토리 및 목적지가 포함됩니다. 이 관리 대상 오브젝트 저장소는 샘플 이용자 및 생성자 애플리케이션 각각에서 사용할 수 있습니다.

다양한 API에서 동일한 기능을 지원하는 샘플에는 구문 상 차이가 있습니다.

- 샘플 메시지 이용자 및 생성자 애플리케이션은 모두 다음 기능을 지원합니다.

- IBM MQ, IBM Integration Bus(브로커에 대한 실시간 연결 사용) 및 WebSphere Application Server service integration bus에 연결
 - 초기 컨텍스트 인터페이스를 사용하여 관리 대상 오브젝트 저장소 검색
 - 큐(IBM MQ 및 WebSphere Application Server service integration bus) 및 토픽(IBM MQ, 브로커에 대한 실시간 연결 및 WebSphere Application Server service integration bus)에 연결
 - 기본, 바이트, 맵, 오브젝트, 스트림 및 텍스트 메시지
- 샘플 메시지 이용자 애플리케이션은 동기 및 비동기 수신 모드와 SQL 선택자 명령문을 지원합니다.
 - 샘플 메시지 생성자 애플리케이션은 지속 전달 모드와 비지속 전달 모드를 지원합니다.

샘플은 두 모드 중 하나에서 작동할 수 있습니다.

단순 모드

최소 사용자 입력으로 샘플을 실행할 수 있습니다.

고급 모드

샘플이 작동하는 방식을 좀 더 세밀하게 사용자 정의할 수 있습니다.

모든 샘플은 호환 가능하므로 여러 언어로 운영될 수 있습니다.

Windows IBM MQ 환경 Windows 내 애플리케이션을 XMS.NET 위한 코어를 .NET 지원합니다. 샘플을 포함한 IBM MQ classes for .NET Standard는 기본적으로 IBM MQ 표준 설치의 일부로 설치됩니다.

Linux IBM MQ 또한 환경 Linux 내 애플리케이션을 위한 Core를 .NET 지원합니다.

XMS.NET의 샘플 애플리케이션은 &MQINSTALL_PATH&/samp/dotnet/samples/cs/core/xms에 설치됩니다.

자세한 정보는 585 페이지의 『설치 IBM MQ classes for XMS .NET』의 내용을 참조하십시오.

.NET 샘플 애플리케이션 실행

단순 또는 고급 모드에서는 대화식으로 그리고 자동 생성 또는 사용자 정의된 응답 파일을 사용하는 경우는 비대화식으로 .NET 샘플 애플리케이션을 실행할 수 있습니다.

시작하기 전에

제공된 샘플 애플리케이션을 실행하기 전에 애플리케이션이 서버에 연결할 수 있도록 먼저 메시징 서버 환경을 설정해야 합니다. 590 페이지의 『메시징 서버 환경 설정』의 내용을 참조하십시오.

프로시저

.NET 샘플 애플리케이션을 실행하려면 다음 단계를 완료하십시오.

팁: 샘플 애플리케이션을 실행할 때, 다음 수행할 작업에 대해 도움을 받으려면 언제든지 ?를 입력하십시오.

1. 샘플 애플리케이션을 실행할 모드를 선택하십시오.

Advanced 또는 Simple을 입력하십시오.

2. 질문에 응답하십시오.

질문 끝에서 대괄호로 묶여 표시되는 기본값을 선택하려면 Enter를 누르십시오. 다른 값을 선택하려면 적절한 값을 입력하고 Enter를 누르십시오.

다음은 질문의 예입니다.

```
Enter connection type [wpm]:
```

이 경우 기본값은 wpm(WebSphere Application Server service integration bus에 연결)입니다.

결과

샘플 애플리케이션을 실행하면 응답 파일이 현재 작업 디렉토리에서 자동으로 생성됩니다. 응답 파일의 이름은 `connection_type-sample_type.rsp` 형식입니다(예: `wpm-producer.rsp`). 필요한 경우 다시 옵션을

입력할 필요가 없도록 생성된 응답 파일을 사용하여 동일한 옵션으로 샘플 애플리케이션을 다시 실행할 수 있습니다.

관련 태스크

[.NET 샘플 애플리케이션 빌드](#)

샘플 .NET 애플리케이션을 빌드하는 경우 선택한 샘플의 실행 파일이 작성됩니다.

[자체 애플리케이션 빌드](#)

샘플 애플리케이션을 빌드하는 것처럼 자체 애플리케이션을 빌드할 수 있습니다.

.NET 샘플 애플리케이션 빌드

샘플 .NET 애플리케이션을 빌드하는 경우 선택한 샘플의 실행 파일이 작성됩니다.

시작하기 전에

적절한 컴파일러를 설치하십시오. 이 태스크에서는 사용자가 Microsoft Visual Studio 2012를 설치했고 익숙하게 사용한다고 가정합니다.

프로시저

.NET 샘플 애플리케이션을 빌드하려면 다음 단계를 완료하십시오.

1. .NET 샘플과 함께 제공된 Samples.sln 솔루션 파일을 클릭하십시오.
2. 솔루션 탐색기 창에서 솔루션 Samples을 마우스의 오른쪽 단추로 클릭하고 **솔루션 빌드**를 선택하십시오.

결과

선택한 구성에 따라 샘플의 해당 하위 폴더(bin/Debug 또는 bin/Release)에 실행 가능 프로그램이 작성됩니다. 이 프로그램은 이름이 폴더와 동일하며 접미부는 CS입니다. 예를 들어 메시지 생성자 샘플 애플리케이션의 C# 버전을 빌드하는 경우 SampleProducerCS.exe가 SampleProducer 폴더에서 작성됩니다.

관련 태스크

[.NET 샘플 애플리케이션 실행](#)

단순 또는 고급 모드에서는 대화식으로 그리고 자동 생성 또는 사용자 정의된 응답 파일을 사용하는 경우는 비대화식으로 .NET 샘플 애플리케이션을 실행할 수 있습니다.

[자체 애플리케이션 빌드](#)

샘플 애플리케이션을 빌드하는 것처럼 자체 애플리케이션을 빌드할 수 있습니다.

자체 애플리케이션 빌드

샘플 애플리케이션을 빌드하는 것처럼 자체 애플리케이션을 빌드할 수 있습니다.

시작하기 전에

적절한 컴파일러를 설치하십시오. 이 태스크에서는 사용자가 Microsoft Visual Studio 2012를 설치했고 익숙하게 사용한다고 가정합니다.

프로시저

- [597 페이지의 『.NET 샘플 애플리케이션 빌드』](#)에 설명된 대로 .NET 애플리케이션을 빌드하십시오.
자체 애플리케이션의 빌드 방법에 대한 추가 지침은 각 샘플 애플리케이션에 제공된 makefile을 사용하십시오.

팁: 실패할 경우 문제점 진단을 위해서는 제공된 기호로 애플리케이션을 컴파일하는 것이 좋습니다.

관련 태스크

[.NET 샘플 애플리케이션 실행](#)

단순 또는 고급 모드에서는 대화식으로 그리고 자동 생성 또는 사용자 정의된 응답 파일을 사용하는 경우는 비대화식으로 .NET 샘플 애플리케이션을 실행할 수 있습니다.

XMS .NET 애플리케이션 작성

이 섹션은 속성, 데이터 유형 및 오류 처리에 대한 정보를 포함하여 애플리케이션 XMS .NET 작성 시 도움이 되는 정보를 제공합니다.

시작하기 전에

V 9.4.0 **V 9.4.0** **Deprecated** 부터 IBM MQ 9.4.0, 에 IBM MQ classes for XMS .NET, 데이터 직렬화 및 역직렬화에 사용되는 메서드 WriteObject(), ReadObject(), CreateObjectMessage(), 그리고 클래스 ObjectMessage 및 XmsObjectMessageImpl 는 더 이상 사용되지 않습니다.

Removed 사용하여 빌드된 클라이언트 XMS.NET.NET Standard 2.0 라이브러리는 에 사용 IBM MQ 9.3.1 중단되었으며, 에 제품에서 IBM MQ 9.4.0 제거되었습니다.

부터 IBM MQ 9.2.0, 동적 링크 XMS .NET 라이브러리(DLL)의 수가 크게 줄어들어 총 다섯 개로 축소되었습니다. 5개의 동적 링크 라이브러리는 다음과 같습니다.

- IBM.XMS.dll - 모든 자국어 메시지를 포함함
- IBM.XMS.Comms.RMM.dll
- 세 개의 정책 동적 링크 라이브러리:
 - policy.8.0.IBM.XMS.dll
 - policy.9.0.IBM.XMS.dll
 - policy.9.1.IBM.XMS.dll

XMS .NET에서 모든 문자열은 고유 .NET 문자열을 사용하여 전달됩니다. 이는 고정된 인코딩이므로 해석하는 데 추가 정보가 필요하지 않습니다. 따라서 XMSC_CLIENT_CCSID 속성은 애플리케이션에 XMS .NET 필요하지 않습니다.

이 태스크 정보

이 절에는 다음 주제가 포함되어 있습니다.

- [599 페이지의 『.NET의 관리 조작 및 비관리 조작』](#)
- [600 페이지의 『스레드 모델』](#)
- [600 페이지의 『부동산 XMS .NET』](#)
- [601 페이지의 『ConnectionFactory 오브젝트 및 Connection 오브젝트』](#)
- [603 페이지의 『세션』](#)
- [606 페이지의 『목적지』](#)
- [609 페이지의 『메시지 생성자』](#)
- [609 페이지의 『메시지 이용자』](#)
- [612 페이지의 『큐 브라우저』](#)
- [613 페이지의 『요청자』](#)
- [613 페이지의 『오브젝트 삭제』](#)
- [613 페이지의 『데이터 유형 XMS .NET』](#)
- [614 페이지의 『XMS 기본 유형』](#)
- [615 페이지의 『한 데이터 유형에서 다른 데이터 유형으로 특성 값의 암시적 변환』](#)
- [617 페이지의 『반복기』](#)
- [620 페이지의 『XMS .NET의 오류 처리』](#)
- [617 페이지의 『.NET에서 메시지 및 예외 리스너 사용』](#)

- [618 페이지의 『XMS을\(를\) 통한 자동 IBM MQ 클라이언트 재연결』](#)

.NET의 관리 조작 및 비관리 조작

관리 코드는 .NET 공통 언어 런타임 환경 내에서 배타적으로 실행되며 전적으로 해당 런타임에서 제공되는 서비스에 종속됩니다. 애플리케이션의 일부가 .NET 공용 언어 런타임 환경 외부에서 실행되거나 서비스를 호출하면 애플리케이션은 비관리로 분류됩니다.

관리 .NET 환경에서는 현재 특정 고급 기능을 지원할 수 없습니다.

완전히 관리되는 환경에서 현재 지원되지 않는 일부 기능이 애플리케이션에 필요한 경우 애플리케이션을 실제 변경할 필요 없이 애플리케이션이 비관리 환경을 사용하도록 변경할 수 있습니다. 그러나 이 선택사항이 지정되면 XMS 스택이 비관리 코드를 사용합니다.

IBM MQ 큐 관리자에 대한 연결

WMQ_CM_CLIENT에 대한 관리 연결에서는 비TCP 통신 및 채널 압축을 지원하지 않습니다. 그러나 이러한 연결은 비관리 연결(WMQ_CM_CLIENT_UNMANAGED)을 사용하여 지원될 수 있습니다. 자세한 정보는 [520 페이지의 『.NET 애플리케이션 개발』](#)의 내용을 참조하십시오.

비관리 환경의 관리 대상 오브젝트에서 연결 팩토리를 작성하는 경우 수동으로 연결 모드 값을 XMSC_WMQ_CM_CLIENT_UNMANAGED로 변경해야 합니다.

WebSphere Application Server 서비스 통합 버스 메시징 엔진에 대한 연결

SSL 프로토콜(HTTPS 포함)을 사용해야 하는 WebSphere Application Server 서비스 통합 버스 메시징 엔진에 대한 연결은 현재 관리 코드로서 지원되지 않습니다.

IBM MQ XMS .NET 프로젝트 템플릿 사용

IBM MQ XMS .NET 클라이언트는 XMS .NET Core 애플리케이션 개발을 지원하기 위해 프로젝트 템플릿을 사용하는 기능을 제공합니다.

시작하기 전에

Microsoft Visual Studio 2017 이상 및 .NET Core 2.1이 시스템에 있어야 합니다.

XMS .NET 템플릿을 복사해야 합니다.

```
&MQ_INSTALL_ROOT%\tools\dotnet\samples\cs\core\xms\ProjectTemplates\IBMXMS.NETClientApp.zip
```

디렉토리에서

```
&USER_HOME_DIRECTORY%\Documents\&Visual_Studio_Version%\Templates\ProjectTemplates
```

디렉토리로 복사합니다. 여기서,

- &MQ_INSTALL_ROOT는 설치의 루트 디렉토리입니다.
- &USER_HOME_DIRECTORY는 홈 디렉토리입니다.

템플릿을 선택하려면 Microsoft Visual Studio를 중지한 후 재시작해야 합니다.

이 태스크 정보

XMS .NET 프로젝트 템플릿은 애플리케이션을 개발하는 데 사용할 수 있는 몇 가지 공통 코드를 포함합니다. 내장 코드를 통해 IBM MQ 큐 관리자에 연결하고, 내장 코드에서 특성을 수정하여 넣기 및 가져오기 조작을 수행할 수 있습니다.

프로시저

1. Microsoft Visual Studio를 여십시오.
2. 파일을 클릭하고 새로 작성, 프로젝트를 연속해서 클릭하십시오.
3. 새 프로젝트 만들기 창에서 IBM XMS .NET Client App (.NET Core) 선택하고 다음을 클릭합니다.
4. 새 프로젝트 구성 창에서 원하는 경우 프로젝트의 프로젝트 이름을 변경하고 작성을 클릭하여 XMS.NET 프로젝트를 작성하십시오.
XMSDotnetApp.cs은(는) 프로젝트 파일과 함께 작성되는 파일입니다. 이 파일은 큐 관리자에 연결되는 코드를 포함하여, 송신 및 수신 조작을 수행합니다.
연결 속성은 기본값으로 설정됩니다.
 - WMQ_CONNECTION_NAME_LIST는 localhost(1414)로 설정됨
 - XMSC.WMQ_CHANNEL은 DOTNET.SVRCONN으로 설정됨큐는 Q1로 설정되며, 이 속성을 적절히 수정할 수 있습니다.
5. 애플리케이션을 컴파일하고 실행하십시오.

관련 개념

[IBM MQ 컴포넌트 및 기능](#)

[.NET 애플리케이션 런타임 - Windows만 해당](#)

스레드 모델

멀티스레드 애플리케이션이 XMS 오브젝트를 사용하는 방법에 적용되는 일반 규칙입니다.

- 다음 유형의 오브젝트만 다른 스레드에서 동시에 사용할 수 있습니다.
 - ConnectionFactory
 - 연결
 - ConnectionMetaData
 - 목적지
- Session 오브젝트는 한 번에 하나의 스레드에서만 사용할 수 있습니다.

이러한 규칙에 대한 예외는 [IBM Message Service Client for .NET](#) 참조에 있는 메소드의 인터페이스 정의에서 "스레드 컨텍스트" 로 레이블된 항목으로 표시됩니다.

부동산 XMS .NET

.NET 애플리케이션은 PropertyContext 인터페이스의 메소드를 사용하여 오브젝트의 특성을 가져오고 설정합니다. XMS.NET에서 존재하지 않는 특성 처리는 대략적으로 JMS 스펙 및 XMS의 C 및 C++ 구현과 일치합니다.

.NETXMS 속성과 그 값들

PropertyContext 인터페이스는 특성 가져오기 및 설정 메소드를 캡슐화합니다. 이러한 메소드는 다음 클래스에 의해 직접 또는 간접적으로 상속됩니다.

- [BytesMessage](#)
- [연결](#)
- [ConnectionFactory](#)
- [ConnectionMetaData](#)
- [대상](#)
- [MapMessage](#)
- [메시지](#)
- [MessageConsumer](#)
- [MessageProducer](#)

- [ObjectMessage](#)
- [QueueBrowser](#)
- [세션](#)
- [StreamMessage](#)
- [TextMessage](#)

애플리케이션이 특성 값을 설정하면 새 값이 특성에 있는 이전 값을 대체합니다. XMS 특성에 대한 자세한 정보는 [XMS 오브젝트 특성](#)을 참조하십시오.

편리한 사용을 위해 XMS의 XMS 특성 이름 및 값이 XMSC 구조체의 공용 상수로 미리 정의되어 있습니다. 이러한 상수의 이름은 XMSC.*constant* 양식입니다. 예: XMSC.USERID(특성 이름 상수) 및 XMSC.DELIVERY_AS_APP(값 상수).

또한 IBM.XMS.MQC 구조체를 사용하면 IBM MQ 상수에도 액세스할 수 있습니다. IBM.XMS 네임스페이스를 이미 가져온 경우, MQC.*constant* 양식의 이 특성 값에 액세스할 수 있습니다. 예: MQC.MQRO_COA_WITH_FULL_DATA.

하이브리드 애플리케이션에서 와 IBM MQ XMS.NET 클래스를 모두 사용하여 를 .NET 구현하고, 동시에 IBM.XMS 및 IBM.WMQ 네임스페이스를 모두 임포트하는 경우, 각 발생이 고유하도록 보장하기 위해 MQC 구조체 네임스페이스를 완전히 한정해야 합니다.

참고: 관리되는 .NET 환경에서는 현재 일부 고급 기능이 지원되지 않습니다. 자세한 정보는 [599 페이지의 『.NET의 관리 조작 및 비관리 조작』](#)의 내용을 참조하십시오.

존재하지 않는 속성의 처리 XMS .NET

JMS에서 존재하지 않는 특성에 액세스하면 메소드가 필수 유형으로 존재하지 않는(널) 값을 변환하려 할 때 Java 시스템 예외가 발생합니다. 특성이 존재하지 않는 경우 다음 예외가 발생합니다.

- getStringProperty 및 getObjectProperty에서 널을 리턴함
- Boolean.valueOf(null)가 false를 리턴하므로 getBooleanProperty가 false를 리턴함
- Integer.valueOf(null)가 예외를 처리하므로 getIntProperty 등이 java.lang.NumberFormatException을 처리함

속성이 에 존재하지 않으면 XMS.NET 다음 예외가 발생합니다:

- GetStringProperty 및 GetObjectProperty(및 GetBytesProperty)에서 널을 리턴함(이는 Java와 동일함)
- GetBooleanProperty가 System.NullReferenceException을 처리함
- GetIntProperty 등이 System.NullReferenceException을 처리함

이 구현은 Java와 다르지만 대략적으로 JMS 스펙 및 XMS C/C++ 인터페이스와 일치합니다. Java 구현에서와 같이 XMS .NET는 System.Convert 호출의 예외를 호출자에게 전파합니다. 그러나 Java와는 달리 XMS는 .NET 프레임워크의 고유 작동을 사용하는 대신 시스템 변환 루틴에 널을 전달함으로써 NullReferenceExceptions를 명시적으로 전달합니다. 애플리케이션이 "abc"와 같은 문자열로 특성을 설정하고 GetIntProperty를 호출하면, Convert.ToInt32("abc")에서 전달된 System.FormatException이 호출자에게 전파되고 이는 Java와 일치합니다. MessageFormatException은 setProperty와 getProperty에 사용된 유형이 호환되지 않는 경우에만 처리됩니다. 이러한 작동은 Java와도 일치합니다.

ConnectionFactory 오브젝트 및 Connection 오브젝트

ConnectionFactory 오브젝트는 애플리케이션에서 Connection 오브젝트를 작성하는 데 사용하는 템플릿을 제공합니다. 애플리케이션은 Connection 오브젝트를 사용하여 Session 오브젝트를 작성합니다.

.NET의 경우 XMS 애플리케이션은 먼저 XMSFactoryFactory 오브젝트를 사용하여 필요한 프로토콜 유형에 적절한 ConnectionFactory 오브젝트에 대한 참조를 가져옵니다. 이 ConnectionFactory 오브젝트는 해당 프로토콜 유형에 대해서만 연결을 생성할 수 있습니다.

XMS 애플리케이션은 여러 연결을 작성하고, 멀티스레드 애플리케이션은 여러 스레드에서 동시에 단일 Connection 오브젝트를 사용할 수 있습니다. Connection 오브젝트는 애플리케이션과 메시징 서버 사이의 통신 연결을 캡슐화합니다.

연결은 다음과 같은 여러 가지 용도로 사용됩니다.

- 애플리케이션에서 연결을 작성하면 애플리케이션은 인증 받을 수 있습니다.
- 애플리케이션은 고유한 클라이언트 ID를 연결과 연관시킬 수 있습니다. 클라이언트 ID는 발행/구독 도메인에서 지속 가능 구독을 지원할 때 사용됩니다. 클라이언트 ID는 두 가지 방법으로 설정할 수 있습니다.

연결 클라이언트 ID를 지정하는 바람직한 방법은 특성을 사용하여 클라이언트 고유의 ConnectionFactory 오브젝트에서 구성하고 이를 투명하게 작성되는 연결에 지정하는 것입니다.

다른 방법으로는, Connection 오브젝트에 설정되어 있는 제공자 고유 값을 사용하여 클라이언트 ID를 지정하는 방법이 있습니다. 이 값은 관리상 구성된 ID를 대체하지 않습니다. 관리상 지정된 ID가 존재하지 않는 경우에 제공됩니다. 관리상 지정된 ID가 존재하는 경우 제공자 고유 값으로 대체하려 시도하면 예외가 발생합니다. 애플리케이션이 ID를 명시적으로 설정하는 경우 이는 연결을 작성한 후 그리고 연결에 대한 다른 조치를 수행하기 바로 전에 수행해야 합니다. 그렇지 않으면 예외가 발생합니다.

XMS 애플리케이션은 일반적으로 하나의 연결, 하나 이상의 세션 및 여러 메시지 생성자 및 메시지 이용자를 작성합니다.

연결 작성은 통신 연결을 생성하고 애플리케이션을 인증하는 작업과도 관련될 수 있으므로 시스템 자원 측면에서는 상당히 소모적입니다.

연결 시작됨 및 중지됨 모드

연결은 시작됨 또는 중지됨 모드에서 작동될 수 있습니다.

애플리케이션에서 연결을 작성하면 연결은 중지됨 모드가 됩니다. 연결이 중지됨 모드인 경우 애플리케이션은 세션을 초기화할 수 있고 동기적 또는 비동기적으로 메시지를 전송할 수 있으나 수신할 수 없습니다.

애플리케이션은 Start Connection 메소드를 호출하여 연결을 시작할 수 있습니다. 연결이 시작됨 모드인 경우 애플리케이션은 메시지를 전송하거나 수신할 수 있습니다. 그런 다음 애플리케이션은 Stop Connection 및 Start Connection 메소드를 호출하여 연결을 중지한 후 다시 시작할 수 있습니다.

연결 닫기

애플리케이션은 Close Connection 메소드를 호출하여 연결을 닫습니다. 애플리케이션에서 연결을 닫는 경우 XMS는 다음 조치를 수행합니다.

- 연결과 연관된 세션을 모두 닫고 해당 세션과 연관된 특정 오브젝트를 삭제합니다. 삭제되는 오브젝트에 대한 자세한 정보는 613 페이지의 『오브젝트 삭제』의 내용을 참조하십시오. 동시에 XMS는 세션 내에서 현재 진행 중인 모든 트랜잭션을 롤백합니다.
- 메시징 서버와의 통신 연결을 종료합니다.
- 연결에서 사용된 메모리 및 기타 내부 자원을 해제합니다.

XMS는 연결을 닫기 전, 세션 중에 수신확인에 실패한 메시지의 수신을 수신확인하지 않습니다. 메시지 수신 수신 확인에 대한 자세한 정보는 604 페이지의 『메시지 수신확인』의 내용을 참조하십시오.

예외 처리

XMS .NET 모든 예외는 System.Exception 에서 파생됩니다. 자세한 정보는 620 페이지의 『XMS .NET의 오류 처리』의 내용을 참조하십시오.

서비스 통합 버스에 대한 연결

XMS 애플리케이션은 직접 TCP/IP 연결 또는 TCP/IP를 통한 HTTP를 사용하여 WebSphere Application Server 서비스 통합 버스에 연결할 수 있습니다.

TCP/IP와의 직접 연결이 불가능한 경우 HTTP 프로토콜을 사용할 수 있습니다. 일반적인 상황은 예를 들어, 두 기업이 메시지를 교환하는 경우와 같이 방화벽을 통해 통신하는 경우입니다. HTTP를 사용하여 방화벽을 통해 통신하는 것은 종종 HTTP 터널링이라고 합니다. 그러나 HTTP 터널링은 직접 TCP/IP 연결을 사용하는 것보다 본질적으로 느립니다. HTTP 헤더는 전송되는 데이터 양에 상당하게 추가되고 HTTP 프로토콜에서는 TCP/IP보다 더 많은 통신 플로우가 필요하기 때문입니다.

TCP/IP 연결을 작성하기 위해 애플리케이션은 XMSC_WPM_TARGET_TRANSPORT_CHAIN 특성이 XMSC_WPM_TARGET_TRANSPORT_CHAIN_BASIC으로 설정된 연결 팩토리를 사용할 수 있습니다. 이것이 특성의 기본값입니다. 연결이 작성되면 연결의 XMSC_WPM_CONNECTION_PROTOCOL 특성이 XMSC_WPM_CP_TCP로 설정됩니다.

HTTP를 사용하는 연결을 작성하려면, 애플리케이션이 HTTP 전송 채널을 사용하도록 구성된 인바운드 전송 채널의 이름으로 XMSC_WPM_TARGET_TRANSPORT_CHAIN 특성이 설정된 연결 팩토리를 사용해야 합니다. 연결이 작성되면 연결의 XMSC_WPM_CONNECTION_PROTOCOL 특성이 XMSC_WPM_CP_TCP로 설정됩니다. 운송 채널 구성 방법에 대한 자세한 내용은 제품 WebSphere Application Server 설명서의 '운송 채널 구성' 항목을 참조하십시오.

부트스트랩 서버에 연결할 경우에도 애플리케이션은 유사한 통신 프로토콜을 선택하게 됩니다. 연결 팩토리의 XMSC_WPM_PROVIDER_ENDPOINTS 특성은 하나 이상의 부트스트랩 서버 엔드포인트 주소입니다. 각 엔드포인트 주소의 부트스트랩 전송 채널 컴포넌트는 부트스트랩 서버와 TCP/IP로 연결될 경우에는 XMSC_WPM_BOOTSTRAP_TCP이며, HTTP를 사용하는 연결일 경우에는 XMSC_WPM_BOOTSTRAP_HTTP입니다.

세션

세션은 메시지 송신 및 수신을 위한 단일 스레드 컨텍스트입니다.

애플리케이션은 세션을 사용하여 메시지, 메시지 생성자, 메시지 사용자, 큐 브라우저 및 임시 목적지를 작성할 수 있습니다. 또한 세션을 사용하여 로컬 트랜잭션을 실행할 수도 있습니다.

애플리케이션은 다중 세션을 작성할 수 있으며 각 세션은 다른 세션과 독립적으로 메시지를 생성하고 이용합니다. 별도의 세션(또는 동일한 세션)에 있는 두 메시지 이용자가 같은 토픽을 구독하면 각각 해당 토픽에 발행된 임의의 메시지의 사본을 수신합니다.

Connection 오브젝트와 달리 Session 오브젝트는 다른 스레드에서 동시에 사용할 수 없습니다. Session 오브젝트의 Close Session 메소드만 당시 Session 오브젝트가 사용 중인 스레드 이외의 스레드에서 호출할 수 있습니다. Close Session 메소드는 세션을 끝내고 세션에 할당된 시스템 자원을 릴리스합니다.

애플리케이션이 둘 이상의 스레드에서 동시에 메시지를 처리해야 하는 경우, 애플리케이션은 각 스레드에서 하나의 세션을 작성하고, 해당 스레드 내에서의 전송 또는 수신 조작에 해당 세션을 사용해야 합니다.

트랜잭션 세션

XMS 애플리케이션은 로컬 트랜잭션을 실행할 수 있습니다. 로컬 트랜잭션은 애플리케이션이 연결된 큐 관리자나 서비스 통합 버스의 자원에 대한 변경사항만 포함하는 트랜잭션입니다.

이 토픽에 있는 정보는 애플리케이션이 IBM MQ 큐 관리자 또는 WebSphere Application Server 서비스 통합 버스에 연결하는 경우에만 관련됩니다. 정보는 브로커에 대한 실시간 연결과 관련이 없습니다.

로컬 트랜잭션을 실행하기 위해 애플리케이션은 Connection 오브젝트의 Create Session 메소드를 호출하고 세션이 트랜잭션되는 매개변수를 지정하여 트랜잭션되는 세션을 먼저 작성해야 합니다. 결과적으로, 세션 내에서 송신 및 수신된 모든 메시지는 트랜잭션의 시퀀스로 그룹화됩니다. 트랜잭션이 시작된 이후에 송신하거나 수신한 메시지를 애플리케이션이 커밋하거나 롤백하면 트랜잭션이 종료됩니다.

트랜잭션을 커밋하기 위해 애플리케이션은 Session 오브젝트의 Commit 메소드를 호출합니다. 트랜잭션이 커밋되는 경우, 트랜잭션 내에서 송신된 모든 메시지가 다른 애플리케이션에 전달하기 위해 사용 가능하며 트랜잭션 내에서 수신된 모든 메시지가 수신확인되므로 메시징 서버는 이를 다시 애플리케이션에 전달하려고 시도하지 않습니다. 포인트-투-포인트 도메인에서, 메시징 서버는 해당 큐에서 수신된 메시지도 제거합니다.

트랜잭션을 롤백하기 위해 애플리케이션은 Session 오브젝트의 Rollback 메소드를 호출합니다. 트랜잭션이 롤백되면 메시징 서버는 트랜잭션 내의 송신된 모든 메시지를 버리며, 트랜잭션 내의 수신된 모든 메시지가 다시 전달을 위해 사용 가능합니다. 포인트-투-포인트 도메인에서, 수신된 메시지는 해당 큐에 다시 놓이며 다시 기타 애플리케이션에서 이를 볼 수 있습니다.

애플리케이션이 트랜잭션된 세션을 작성하거나 Commit 또는 Rollback 메소드를 호출하면 새로운 트랜잭션이 자동으로 시작됩니다. 따라서 트랜잭션 세션에는 항상 활성 트랜잭션이 있습니다.

애플리케이션이 트랜잭션 세션을 닫으면 암시적 롤백이 발생합니다. 애플리케이션이 연결을 닫으면 모든 연결의 트랜잭션 세션에 대해 암시적 롤백이 발생합니다.

트랜잭션은 전적으로 트랜잭션 세션 내에 포함됩니다. 트랜잭션은 세션 간에 걸칠 수 없습니다. 이는 애플리케이션이 두 개 이상의 트랜잭션 세션에서 메시지를 송신하고 수신한 후에 이 모든 조치를 단일 트랜잭션으로 커밋하거나 롤백할 수 없음을 의미합니다.

관련 개념

메시지 수신확인

트랜잭션되지 않은 모든 세션에는 애플리케이션이 수신한 메시지가 수신확인되는 방법을 결정하는 수신확인 모드가 있습니다. 3개의 수신확인 모드가 사용 가능하며, 수신확인 모드의 선택사항은 애플리케이션의 디자인에 영향을 줍니다.

메시지 전달

XMS에서는 메시지 전달의 지속 및 비지속 모드를 지원하며 메시지의 동기 및 비동기 전달을 지원합니다.

XMS을(를) 통한 관리 IBM MQ XA 트랜잭션

관리 IBM MQ XA 트랜잭션은 XMS를 통해 사용될 수 있습니다.

메시지 수신확인

트랜잭션되지 않은 모든 세션에는 애플리케이션이 수신한 메시지가 수신확인되는 방법을 결정하는 수신확인 모드가 있습니다. 3개의 수신확인 모드가 사용 가능하며, 수신확인 모드의 선택사항은 애플리케이션의 디자인에 영향을 줍니다.

참고: 이 주제는 애플리케이션이 큐 IBM MQ 관리자 또는 서비스 통합 WebSphere Application Server 버스에 연결하는 경우에만 관련이 있습니다. 정보는 브로커에 대한 실시간 연결과 관련이 없습니다.

XMS는 JMS가 사용하는 메시지의 수신 수신확인에 동일한 메커니즘을 사용합니다.

세션이 트랜잭션되지 않은 경우, 애플리케이션이 수신한 메시지가 수신확인되는 방법은 세션의 수신확인 모드에 의해 판별됩니다. 세 가지 확인 모드가 있습니다:

XMSC_AUTO_ACKNOWLEDGE

세션은 애플리케이션이 수신한 각 메시지를 자동으로 수신확인합니다.

메시지가 애플리케이션에 동기적으로 전달되는 경우, 세션은 수신 호출이 완료될 때마다 메시지의 수신을 수신확인합니다. 애플리케이션이 메시지를 성공적으로 수신하지만 장애로 인해 수신확인이 발생하지 않는 경우, 해당 메시지는 다시 전달에 사용될 수 있습니다. 따라서 애플리케이션은 다시 전달되는 메시지를 처리할 수 있어야 합니다.

XMSC_DUPS_OK_ACKNOWLEDGE

세션은 선택된 시간에 애플리케이션이 수신한 메시지를 수신확인합니다.

이 수신확인 모드를 사용하면 세션이 수행해야 하는 작업의 양이 줄어들지만, 메시지 수신확인을 방지하는 장애의 결과로 인해 둘 이상의 메시지가 다시 전달에 사용될 수 있습니다. 따라서 애플리케이션은 다시 전달되는 메시지를 처리할 수 있어야 합니다.

XMSC_CLIENT_ACKNOWLEDGE

애플리케이션은 Message 클래스의 Acknowledge 메소드를 호출하여 수신한 메시지를 수신확인합니다.

애플리케이션은 각 메시지의 수신을 개별적으로 수신확인할 수도 있으며, 또는 메시지의 일괄처리를 수신하고 수신한 마지막 메시지에 대해서만 Acknowledge 메소드를 호출할 수도 있습니다. Acknowledge 메소드가 호출되는 경우, 메소드가 마지막으로 호출된 후에 수신된 모든 메시지가 수신확인됩니다.

이러한 수신확인 모드와 결합하여, 애플리케이션은 Session 클래스의 Recover 메소드를 호출하여 세션에서 메시지의 전달을 중지하고 다시 시작할 수 있습니다. 수신자가 이전에 수신확인하지 않은 메시지가 다시 전달됩니다. 그러나 이는 이전에 전달되었던 동일한 순서대로 전달되지 않을 수 있습니다. 그 동안에 보다 높은 우선순위의 메시지가 도착할 수도 있으며, 원래 메시지 중 일부가 만료될 수도 있습니다. 또한 포인트-투-포인트 도메인에서 원래 메시지의 일부를 다른 애플리케이션이 이용했을 수도 있습니다.

애플리케이션은 메시지의 JMSRedelivered 헤더 필드의 콘텐츠를 검사하여 메시지를 재전달하는지 여부를 판별할 수 있습니다. 애플리케이션은 Message 클래스의 Get JMSRedelivered 메소드를 호출하여 메시지의 재전달 여부를 결정합니다.

관련 개념

트랜잭션 세션

XMS 애플리케이션은 로컬 트랜잭션을 실행할 수 있습니다. 로컬 트랜잭션은 애플리케이션이 연결된 큐 관리자나 서비스 통합 버스의 자원에 대한 변경사항만 포함하는 트랜잭션입니다.

메시지 전달

XMS에서는 메시지 전달의 지속 및 비지속 모드를 지원하며 메시지의 동기 및 비동기 전달을 지원합니다.

XMS을(를) 통한 관리 IBM MQ XA 트랜잭션

관리 IBM MQ XA 트랜잭션은 XMS를 통해 사용될 수 있습니다.

메시지 전달

XMS에서는 메시지 전달의 지속 및 비지속 모드를 지원하며 메시지의 동기 및 비동기 전달을 지원합니다.

메시지 전달 방식

XMS에서는 두 가지 모드의 메시지 전달을 지원합니다.

• 지속

지속 메시지는 한 번 전달됩니다. 메시징 서버는 메시지 로깅과 같은 특수 예방 조치를 수행하여 실패한 경우에도 전송 중인 지속적 메시지가 손실되지 않도록 합니다.

• 비지속

비지속 메시지는 한 번만 전달됩니다. 비지속 메시지는 실패 시 전송 중에 손실될 수 있으므로 지속 메시지보다 덜 안정적입니다.

전달 모드를 선택하면 신뢰도와 성능이 상충됩니다. 일반적으로 비지속 메시지는 지속 메시지보다 빨리 전송됩니다.

비동기 메시지 전달

XMS에서는 하나의 스레드를 사용하여 세션에 대한 모든 비동기 메시지 전달을 처리합니다. 이는 한 번에 하나의 메시지 리스너 기능 또는 하나의 `onMessage()` 메소드만 실행할 수 있음을 의미합니다.

세션에 있는 둘 이상의 메시지 이용자가 메시지를 비동기식으로 수신하고 메시지 리스너 기능 또는 `onMessage()` 메소드가 메시지를 메시지 이용자에게 전달하는 경우, 동일한 메시지를 대기 중인 기타 메시지 이용자는 계속 대기해야 합니다. 세션으로 전달되기를 대기하는 다른 메시지도 계속 대기해야 합니다.

애플리케이션에서 메시지를 동시에 전달해야 하는 경우, XMS에서 둘 이상의 스레드를 사용하여 비동기식으로 메시지 전달을 처리할 수 있도록 둘 이상의 세션을 작성하십시오. 이러한 방식으로 하나 이상의 메시지 리스너 기능 또는 `onMessage()` 메소드는 동시에 실행될 수 있습니다.

이용자에게 메시지 리스너를 지정하면 세션이 비동기가 되지 않습니다. `Connection.Start` 메소드가 호출되는 경우에만 세션이 비동기가 됩니다. `Connection.Start` 메소드가 호출될 때까지 모든 동기 호출이 허용됩니다. `Connection.Start`가 호출되면 이용자로의 메시지 전달이 시작됩니다.

비동기 세션에서 소비자 또는 생산자 생성 같은 동기 호출을 수행해야 하는 경우, 반드시 호출해야 `Connection.Stop` 합니다. 세션은 메시지 전달을 시작하기 위해 `Connection.Start` 메소드를 호출하여 재개할 수 있습니다. 이에 대한 유일한 예외는 콜백 기능으로 메시지를 전달하는 스레드인 세션 메시지 전달 스레드입니다. 이 스레드는 메시지 콜백 기능에서 세션에 대한 호출(단기 호출 제외)을 작성할 수 있습니다.

참고: 비관리 모드에서 콜백 기능 내부의 MQDISC 호출은 IBM MQ .NET 클라이언트에서 지원되지 않습니다. 그러므로 클라이언트 애플리케이션은 비동기 수신 모드의 `MessageListener` 콜백 내에서 세션을 작성하거나 닫을 수 없습니다. `MessageListener` 메소드 외부에서 세션을 작성하고 처리하십시오.

동기 메시지 전달

애플리케이션이 `MessageConsumer` 오브젝트의 `Receive` 메소드를 사용할 경우 메시지는 애플리케이션에 동기 로 전달됩니다.

애플리케이션은 `Receive` 메소드를 사용하여 지정된 기간 동안 메시지를 대기하거나 무기한 대기할 수 있습니다. 또는 애플리케이션이 메시지를 대기하지 않을 경우 `Receive with No Wait` 메소드를 사용할 수 있습니다.

관련 개념

트랜잭션 세션

XMS 애플리케이션은 로컬 트랜잭션을 실행할 수 있습니다. 로컬 트랜잭션은 애플리케이션이 연결된 큐 관리자나 서비스 통합 버스의 자원에 대한 변경사항만 포함하는 트랜잭션입니다.

메시지 수신확인

트랜잭션되지 않은 모든 세션에는 애플리케이션이 수신한 메시지가 수신확인되는 방법을 결정하는 수신확인 모드가 있습니다. 3개의 수신확인 모드가 사용 가능하며, 수신확인 모드의 선택사항은 애플리케이션의 디자인에 영향을 줍니다.

XMS을(를) 통한 관리 IBM MQ XA 트랜잭션

관리 IBM MQ XA 트랜잭션은 XMS를 통해 사용될 수 있습니다.

XMS을(를) 통한 관리 IBM MQ XA 트랜잭션

관리 IBM MQ XA 트랜잭션은 XMS를 통해 사용될 수 있습니다.

XMS를 통해 XA 트랜잭션을 사용하려면 트랜잭션 세션을 작성해야 합니다. XA 트랜잭션이 사용 중인 경우, 트랜잭션은 DTC(Distributed Transaction Coordinator) 글로벌 트랜잭션을 통해 제어하며 XMS 세션을 통하지 않습니다. XA 트랜잭션을 사용하는 경우 `Session.commit` 또는 `Session.rollback`은 XMS 세션에서 발행될 수 없습니다. 대신 `Transscope.Commit` 또는 `Transscope.Rollback` DTC 메소드를 사용하여 트랜잭션을 커밋하거나 롤백하십시오. 세션이 XA 트랜잭션에 사용되는 경우, 세션을 사용하여 작성되는 생성자 또는 이용자는 XA 트랜잭션의 일부여야 합니다. 이들은 XA 트랜잭션 범위 이외의 조작에 대해서는 사용할 수 없습니다. 이들은 XA 트랜잭션 외부의 `Producer.send` 또는 `Consumer.receive`와 같은 조작에 사용할 수 없습니다.

다음의 경우 `IllegalStateException` 예외 오브젝트가 전달됩니다.

- `Session.commit` 또는 `Session.rollback`에 대해 XA 트랜잭션 세션이 사용됩니다.
- XA 트랜잭션 세션에서 사용되었던 생성자 또는 이용자는 XA 트랜잭션 범위 외부에서 사용됩니다.

XA 트랜잭션은 비동기 이용자에서 지원되지 않습니다.

참고:

1. XA 트랜잭션을 커밋하기 전에 `Producer`, `Consumer`, `Session` 또는 `Connection` 오브젝트에서 닫기가 발행될 수 있습니다. 이런 경우 트랜잭션의 메시지는 롤백됩니다. 마찬가지로 XA 트랜잭션을 커밋하기 전에 연결이 끊어지면 트랜잭션에 있는 모든 메시지는 롤백됩니다. `Producer` 오브젝트의 경우 롤백은 메시지가 큐에 넣어지지 않음을 의미합니다. `Consumer` 오브젝트의 경우 롤백은 메시지가 큐에 남아 있음을 의미합니다.
2. `Producer` 오브젝트가 `TimeToLive` 상태의 메시지를 `TransactionScope`에 넣고 시간이 경과하면 `commit`을 발행하는 경우, `commit`을 발행하기 전에 메시지가 만료될 수 있습니다. 이런 경우 메시지는 `Consumer` 오브젝트에서 사용할 수 없게 됩니다.
3. `Session` 오브젝트는 스레드에서 지원되지 않습니다. 스레드에서 공유되는 `Session` 오브젝트에서의 트랜잭션 사용은 지원되지 않습니다.

관련 개념

트랜잭션 세션

XMS 애플리케이션은 로컬 트랜잭션을 실행할 수 있습니다. 로컬 트랜잭션은 애플리케이션이 연결된 큐 관리자나 서비스 통합 버스의 자원에 대한 변경사항만 포함하는 트랜잭션입니다.

메시지 수신확인

트랜잭션되지 않은 모든 세션에는 애플리케이션이 수신한 메시지가 수신확인되는 방법을 결정하는 수신확인 모드가 있습니다. 3개의 수신확인 모드가 사용 가능하며, 수신확인 모드의 선택사항은 애플리케이션의 디자인에 영향을 줍니다.

메시지 전달

XMS에서는 메시지 전달의 지속 및 비지속 모드를 지원하며 메시지의 동기 및 비동기 전달을 지원합니다.

목적지

XMS 애플리케이션은 `Destination` 오브젝트를 사용하여 전송 중인 메시지의 목적지와 수신 중인 메시지의 소스를 지정합니다.

XMS 애플리케이션은 런타임 시 `Destination` 오브젝트를 작성하거나 관리 대상 오브젝트 저장소에서 사전 정의된 목적지를 가져올 수 있습니다.

`ConnectionFactory`의 경우와 마찬가지로 XMS 애플리케이션이 목적지를 지정하는 가장 유연한 방법은 목적지를 관리 대상 오브젝트로 정의하는 것입니다. 이러한 접근 방법을 사용하면 C, C++ 및 .NET 언어 및 Java로 작성

된 애플리케이션이 목적지의 정의를 공유할 수 있습니다. 코드를 변경하지 않고도 관리 대상 Destination 오브젝트의 특성을 변경할 수 있습니다.

.NET에서의 목적지

.NET에서 목적지는 프로토콜 유형에 따라 작성되며 목적지가 작성된 프로토콜 유형에서만 사용될 수 있습니다. 목적지를 생성하는 두 가지 방법이 제공됩니다. 하나는 토픽용이고 다른 하나는 큐용입니다:

- `IDestination CreateTopic(String topic);`
- `IDestination CreateQueue(String queue);`

이러한 메서드는 API에서 .NET 다음 두 개체에서 사용할 수 있습니다:

- `ISession`
- `XMSFactoryFactory`

두 경우 모두, 이러한 메서드는 매개변수를 포함할 수 있는 URI 스타일 문자열을 다음 형식으로 받아들일 수 있습니다:

```
"topic://some/topic/name?priority=5"
```

또는 이러한 메서드에서 목적지 이름 즉, `topic://` 또는 `queue://` 접두부 그리고 매개변수가 없는 이름만 승인할 수도 있습니다. 따라서 URI 스타일 문자열은 다음과 같습니다.

```
CreateTopic("topic://some/topic/name");
```

다음과 같은 목적지 이름과 동일한 결과를 생성합니다.

```
CreateTopic("some/topic/name");
```

자세한 내용은 [IDestination](#)을 참조하십시오.

WebSphere Application Server 서비스 통합 버스 JMS에서와 같이, 토픽은 축약 양식으로 지정될 수도 있고, 여기에는 `topicname` 및 `topicspace` 모두가 포함되지만 매개변수를 포함할 수 없습니다.

```
CreateTopic("topicspace:topicname");
```

토픽 URI(Uniform Resource Identifier)

토픽 URI(Uniform Resource Identifier)는 토픽의 이름을 지정합니다. 또한, 토픽 특성을 한 개 이상 지정할 수 있습니다.

토픽 URI는 순서 토픽://로 시작하며 그 뒤에 토픽 이름이 옵니다. 선택적으로 나머지 토픽 특성을 설정하는 이름-값 쌍 목록이 뒤에 올 수 있습니다. 토픽 이름은 공백이 될 수 없습니다.

다음은 .NET 코드의 단편으로 된 예제입니다.

```
topic = session.CreateTopic("topic://Sport/Football/Results?multicast=7");
```

URI에서 사용할 수 있는 이름 및 올바른 값을 포함하여 토픽 특성에 대한 자세한 정보는 [Destination](#) 특성을 참조하십시오.

구독에 사용할 토픽 URI를 지정할 때 와일드카드를 사용할 수 있습니다. 이러한 와일드카드의 구문은 연결 유형과 브로커 버전에 따라 달라지며, 다음 옵션이 제공됩니다:

- WebSphere Application Server 서비스 통합 버스

WebSphere Application Server 서비스 통합 버스

WebSphere Application Server 서비스 통합 버스에서는 다음 와일드카드 문자를 사용합니다.

- * 계층 구조에서 한 레벨의 임의의 문자 일치
- // 0 이상의 레벨 일치
- // . 0 이상의 레벨 일치(토픽 표현식 끝에서)

608 페이지의 표 84에는 이 와일드카드 설계의 사용 방법에 대한 몇 가지 예가 있습니다.

표 84. WebSphere Application Server 서비스 통합 버스의 와일드카드 설계를 사용하는 예제 URI		
URI(Uniform Resource Identifier)	일치	예
"topic://Sport/*ball/Results"	Sport와 Results 사이에 이름이 "ball"로 끝나는 단일 계층 구조 레벨이 있는 모든 토픽	"topic://Sport/Football/Results" 및 "topic://Sport/Netball/Results"
"topic://Sport//Results"	"Sport/"로 시작하고 "/Results"로 끝나는 모든 토픽	"topic://Sport/Football/Results" 및 "topic://Sport/Hockey/National/Div3/Results"
"topic://Sport/Football//."	"Sport/Football/"로 시작하는 모든 토픽	"topic://Sport/Football/Results" 및 "topic://Sport/Football/TeamNews/Signings/Managerial"
"topic://Sport/*ball//Results//."	토픽	"topic://Sport/Football/Results" 및 "topic://Sport/Netball/National/Div3/Results/2002/November"

관련 개념

[큐 URI\(Uniform Resource Identifier\)](#)

큐 URI는 큐의 이름을 지정합니다. 또한, 큐 특성을 한 개 이상 지정할 수 있습니다.

[임시 목적지](#)

XMS 애플리케이션은 임시 목적지를 작성하고 사용할 수 있습니다.

큐 URI(Uniform Resource Identifier)

큐 URI는 큐의 이름을 지정합니다. 또한, 큐 특성을 한 개 이상 지정할 수 있습니다.

큐의 URI는 queue:// 시퀀스로 시작하고, 그 뒤에 큐의 이름이 오며, 나머지 큐 특성을 설정하는 이름-값 쌍 목록도 포함될 수 있습니다.

IBM MQ 큐의 경우(WebSphere Application Server 기본 메시징 제공자 큐의 경우는 아님), 큐가 상주하는 큐 관리자를 큐 앞에 지정할 수 있으며, 이 경우 /로 큐 관리자 이름과 큐 이름을 구분합니다.

큐 관리자가 지정되면 큐 관리자는 XMS가 이 큐를 사용하여 직접 연결되는 큐 관리자이거나 이 큐에서 액세스할 수 있어야 합니다. 리모트 큐 관리자는 큐에서 메시지를 검색하는 경우에만 지원되고 메시지를 큐에 넣는 경우에는 지원되지 않습니다. 전체 내용은 IBM MQ 큐 관리자 문서를 참조하십시오.

큐 관리자가 지정되지 않는 경우 추가 / 구분 기호는 선택사항이며 구분 기호가 있거나 없어도 큐 정의에는 차이가 없습니다.

다음 대기열 정의는 모두 'XMS'가 직접 연결된 QM_A라는 대기열 관리자의 QB라는 'IBM MQ 대기열에 대해 동일합니다:

```
queue://QB
queue:///QB
queue://QM_A/QB
```

관련 개념

[토픽 URI\(Uniform Resource Identifier\)](#)

토픽 URI(Uniform Resource Identifier)는 토픽의 이름을 지정합니다. 또한, 토픽 특성을 한 개 이상 지정할 수 있습니다.

[임시 목적지](#)

XMS 애플리케이션은 임시 목적지를 작성하고 사용할 수 있습니다.

임시 목적지

XMS 애플리케이션은 임시 목적지를 작성하고 사용할 수 있습니다.

애플리케이션은 일반적으로 임시 목적지를 사용하여 요청 메시지에 대한 응답을 수신합니다. 요청 메시지에 대한 응답을 보낼 목적지를 지정하기 위해 애플리케이션은 요청 메시지를 표시하는 **Message** 오브젝트의 **Set JMSReplyTo** 메소드를 호출합니다. 호출에 지정한 목적지는 임시 목적지가 될 수 있습니다.

임시 목적지를 작성하는 데 세션이 사용되지만 실제로 임시 목적지의 범위는 세션을 작성하는 데 사용된 연결입니다. 연결의 세션은 임시 목적지에 대한 메시지 생성자와 메시지 이용자를 작성할 수 있습니다. 임시 목적지는 명시적으로 삭제되거나 연결이 종료되거나 먼저 발생하는 때까지 그대로 유지됩니다.

애플리케이션이 임시 큐를 작성할 때 애플리케이션이 연결되어 있는 메시징 서버에서 큐가 작성됩니다. 애플리케이션이 큐 관리자에 연결되면 **XMSC WMQ TEMPORARY MODEL** 특성에 의해 이름이 지정되는 모델 큐에서 동적 큐가 작성되고 동적 큐의 이름을 구성하는 데 사용되는 접두부는 **XMSC WMQ TEMP Q PREFIX** 특성에 의해 지정됩니다. 애플리케이션이 서비스 통합 버스에 연결되면 버스에서 임시 큐가 작성되고 임시 큐의 이름을 구성하는 데 사용되는 접두부는 **XMSC WPM TEMP Q PREFIX** 특성에 의해 지정됩니다.

서비스 통합 버스에 연결된 애플리케이션이 임시 토픽을 작성하면 임시 토픽의 이름을 구성하는 데 사용되는 접두부는 **XMSC WPM TEMP TOPIC PREFIX** 특성에 의해 지정됩니다.

관련 개념

토픽 URI(Uniform Resource Identifier)

토픽 URI(Uniform Resource Identifier)는 토픽의 이름을 지정합니다. 또한, 토픽 특성을 한 개 이상 지정할 수 있습니다.

큐 URI(Uniform Resource Identifier)

큐 URI는 큐의 이름을 지정합니다. 또한, 큐 특성을 한 개 이상 지정할 수 있습니다.

메시지 생성자

XMS에서 메시지 생성자는 유효한 목적지가 있는 상태로 또는 연관된 목적지가 없는 상태로 작성될 수 있습니다. 목적지가 널인 메시지 생성자를 작성하는 경우 메시지를 전송할 때 유효한 목적지를 지정해야 합니다.

연관된 목적지가 있는 메시지 생성자

이 시나리오에서 메시지 생성자는 유효한 목적지를 사용하여 작성됩니다. 전송 조작 중 목적지를 지정할 필요가 없습니다.

연관된 목적지가 없는 메시지 생성자

XMS.NET에서 메시지 생성자는 널 목적지로 작성될 수 있습니다.

.NET API를 사용할 때 연관된 목적지가 없는 메시지 생성자를 작성하려면 **ISession** 오브젝트의 **CreateProducer()** 메소드에 매개변수로 **NULL**을 전달해야 합니다 (예: **session.CreateProducer(null)**). 그러나 메시지를 전송할 때 올바른 대상을 지정해야 합니다.

메시지 이용자

메시지 이용자는 지속 가능/비지속 가능 구독자와 동기/비동기 메시지 이용자로 구분될 수 있습니다.

지속 가능 구독자

지속 가능 구독자는 구독자가 비활성인 동안 발행된 메시지를 포함하여 토픽에 대해 발행된 모든 메시지를 수신하는 메시지 이용자입니다.



주의: 이 정보는 애플리케이션이 큐 IBM MQ 관리자 또는 서비스 통합 WebSphere Application Server 버스에 연결하는 경우에만 관련이 있습니다. 정보는 브로커에 대한 실시간 연결과 관련이 없습니다.

토픽에 대한 지속 가능 구독자를 작성하기 위해 애플리케이션은 지속 가능 구독자를 식별하는 이름과 토픽을 표시하는 **Destination** 오브젝트를 매개변수로 지정하여 **Session** 오브젝트의 **Create Durable Subscriber** 메소드를 호출합니다. 애플리케이션은 메시지 선택자를 사용하거나 메시지 선택자를 사용하지 않고 지속 가능 구독자를 작성할 수 있으며 지속 가능 구독자가 자체 연결에 의해 발행된 메시지를 수신할지 여부를 지정할 수 있습니다.

지속 가능 구독자를 작성하는 데 사용되는 세션에는 연관된 클라이언트 ID가 있어야 합니다. 클라이언트 ID는 세션 작성에 사용되는 연결과 연관된 ID와 동일하며 601 페이지의 『ConnectionFactories 오브젝트 및 Connection 오브젝트』에 설명되어 있는 대로 지정됩니다.

지속 가능 구독을 식별하는 이름은 클라이언트 ID 내에서 고유해야 하므로 클라이언트 ID는 지속 가능 구독의 고유한 전체 ID 일부를 구성합니다. 메시징 서버는 지속 가능 구독의 레코드를 유지보수하며 주제에 발행된 모든 메시지가 지속 가능 구독자에 의해 수신확인되거나 만료될 때까지 보존합니다.

메시징 서버는 지속 가능 구독자가 닫힌 후에도 계속 지속 가능 구독의 레코드를 유지보수합니다. 이미 작성된 지속 가능 구독을 재사용하려면 애플리케이션이 지속 가능 구독과 연관된 같은 구독 이름을 지정하고 같은 클라이언트 ID가 있는 세션을 사용하여 지속 가능 구독자를 작성해야 합니다. 한 번에 한 세션에만 특정 지속 가능 구독의 지속 가능 구독자가 있을 수 있습니다.

지속 가능한 구독의 범위는 구독 레코드를 유지보수하는 메시징 서버입니다. 다른 메시징 서버에 연결된 두 애플리케이션이 각각 같은 구독 이름과 클라이언트 ID를 사용하여 지속 가능 구독자를 작성하면 완전히 독립된 두 개의 지속 가능 구독이 작성됩니다.

지속 가능 구독을 삭제하기 위해 애플리케이션은 지속 가능 구독을 식별하는 이름을 매개변수로 지정하여 Session 오브젝트의 Unsubscribe 메소드를 호출합니다. 세션과 연관된 클라이언트 ID는 지속 가능 구독과 연관된 ID와 같아야 합니다. 메시징 서버는 유지보수 중인 지속 가능 구독의 레코드를 삭제하고 지속 가능 구독자에게 추가 메시지를 전송하지 않습니다.

기존 구독을 변경하기 위해 애플리케이션은 동일한 구독 이름과 클라이언트 ID를 사용하고 다른 토픽이나 메시지 선택자(또는 둘 다)를 지정하여 지속 가능 구독자를 작성할 수 있습니다. 지속 가능 구독을 변경하는 것은 구독을 삭제하고 새로 작성하는 것과 같습니다.

큐 IBM MQ 매니저에 연결하는 애플리케이션의 경우, 구독자 XMS 큐를 관리합니다. 그러므로 애플리케이션은 구독자 큐를 지정할 필요가 없습니다. XMS는 구독자 큐가 지정된 경우 이를 무시합니다.

지속 가능 구독에 대한 구독자 큐는 변경할 수 없습니다. 구독자 큐를 변경하는 유일한 방법은 구독을 삭제하고 새로 작성하는 것입니다.

서비스 통합 버스에 연결된 애플리케이션의 경우 각 지속 가능 구독자에는 지정된 지속 가능 구독 홈이 있어야 합니다. 동일한 연결을 사용하는 모든 지속 가능 등록자에 대한 지속 가능 구독 홈을 지정하려면 연결을 작성하는 데 사용되는 ConnectionFactory 오브젝트의 XMSC WPM DUR SUB HOME 특성을 설정하십시오. 각 토픽에 대한 지속 가능 구독 홈을 지정하려면 토픽을 표시하는 Destination 오브젝트의 XMSC_WPM_DUR_SUB_HOME 특성을 설정하십시오. 연결에 대한 지속 가능 구독 홈을 지정해야 애플리케이션에서 연결을 사용하는 지속 가능 구독자를 작성할 수 있습니다. 목적지에 지정한 값은 연결에 지정한 값을 대체합니다.

동기 메시지 이용자

동기식 메시지 소비자(Synchronous Message Consumer)는 큐(Queue)로부터 메시지를 동기적으로 수신하며, 한 번에 하나의 메시지만 수신합니다. Receive(wait interval) 메소드가 사용되는 경우 호출은 지정된 기간(밀리초) 동안 또는 메시지 이용자가 닫을 때까지 메시지를 대기합니다.

ReceiveNoWait() 메소드가 사용되는 경우 동기 메시지 이용자는 지연 없이 메시지를 수신합니다. 다음 메시지가 사용 가능한 경우 즉시 수신됩니다. 그렇지 않으면 널 Message 오브젝트에 대한 포인터가 리턴됩니다.

비동기 메시지 이용자

비동기식 메시지 이용자는 비동기적으로 큐로부터 메시지를 수신합니다. 애플리케이션에 의해 등록된 메시지 리스너는 큐에서 새 메시지를 사용할 수 있을 때마다 호출됩니다.

XMS의 변조 메시지

변조 메시지는 수신 MDB 애플리케이션이 처리할 수 없는 메시지입니다. 독성 메시지가 발견되면, XMS MessageConsumer 객체는 두 개의 큐 속성, **BOQUEUE**, 및 에 따라 해당 **BOTHRESH** 메시지를 재큐잉할 수 있습니다.

일부 환경에서 MDB에 전달된 메시지는 IBM MQ 큐에 롤백될 수 있습니다. 예를 들어 메시지가 나중에 롤백되는 작업 단위 내에서 전달되는 경우 발생할 수 있습니다. 롤백되는 메시지는 일반적으로 다시 전달되지만, 잘못된 형식화된 메시지로 인해 반복적으로 MDB가 실패하므로 전달될 수 없습니다. 이러한 메시지는 변조 메시지라고 합니다. 추가 조사를 위해 변조 메시지가 다른 큐로 자동 전송되거나 버려질 수 있도록 IBM MQ를 구성할 수 있습니다.

다. 이러한 방식으로 IBM MQ 구성하는 방법에 대한 정보는 [을 참조하십시오 612 페이지의 『ASF에서 변조 메시지 핸들링』](#).

종종 잘못된 형식의 메시지가 큐에 도착합니다. 이 컨텍스트에서, 잘못된 형식은 수신 애플리케이션이 메시지를 올바르게 처리할 수 없음을 의미합니다. 해당 메시지로 인해 수신 애플리케이션이 실패할 수 있으며, 이 잘못된 형식의 메시지를 백아웃할 수 있습니다. 그리고 메시지는 반복적으로 입력 큐에 전달되고 반복적으로 애플리케이션에 의해 백아웃될 수 있습니다. 이러한 메시지를 변조 메시지라고 합니다. XMS MessageConsumer 오브젝트는 변조 메시지를 감지하고 이를 대체 목적으로 다시 라우팅합니다.

IBM MQ 큐 관리자는 각 메시지가 백아웃한 횟수의 레코드를 유지합니다. 이 숫자가 구성 가능한 임계값에 도달한 경우, 메시지 이용자는 메시지를 이름 지정된 백아웃 큐에 리큐잉합니다. 이 리큐잉이 어떤 이유로든 실패하는 경우, 메시지가 입력 큐에서 제거되며 데드-레터 큐에 리큐잉되거나 제거됩니다.

XMS ConnectionConsumer 오브젝트는 동일한 방식으로 변조 메시지를 처리하고 동일한 큐 특성을 사용합니다. 다수의 연결 이용자가 동일한 큐를 모니터링하는 경우, 리큐잉이 발생하기 전에 임계값을 초과하는 횟수로 변조 메시지가 애플리케이션에 전달될 수 있습니다. 이 작동은 개별 연결 이용자 모니터가 변조 메시지를 큐잉하고 리큐잉하는 방식 때문입니다.

백아웃 큐의 이름 및 임계값은 IBM MQ 큐의 속성입니다. 속성의 이름은 **BackoutThreshold**입니다 **BackoutRequeueQName**. 이러한 속성이 적용되는 큐는 다음과 같습니다.

- 포인트-투-포인트 메시징의 경우, 이는 기본 로컬 큐입니다. 이는 메시지 이용자 및 연결 이용자가 큐 알리언스를 사용할 때 중요합니다.
- IBM MQ 메시징 제공자 정상 모드에서의 발행/구독 메시징의 경우, 이는 토픽의 관리 큐가 작성되는 모델 큐입니다.
- IBM MQ 메시징 제공자 마이그레이션 모드에서의 발행/구독 메시징의 경우, 이는 TopicConnectionFactory 오브젝트에 정의된 CCSub 큐이거나 Topic 오브젝트에 정의된 CCSub 큐입니다.

및 **BackoutRequeueQNameBackoutThreshold** 속성을 설정하려면 다음 MQSC 명령을 실행하십시오:

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

발행/구독 메시징의 경우, 시스템이 각 구독에 대해 동적 큐를 작성할 때 이러한 속성 값은 IBM MQ classes for JMS 모델 큐인 SYSTEM.JMS.MODEL.QUEUE에서 확보됩니다. 이러한 설정을 대체하려면 다음을 사용하십시오.

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

백아웃 임계값이 0인 경우에는 변조 메시지 핸들링이 사용되지 않으며 변조 메시지가 입력 큐에 남아 있습니다. 그렇지 않으면, 백아웃 계수가 임계값에 도달할 때 메시지가 이름 지정된 백아웃 큐에 송신됩니다.

백아웃 수가 임계값에 도달하지만 메시지를 백아웃 큐로 이동시킬 수 없으면 메시지는 데드-레터 큐로 전송되거나 메시지가 비지속 메시지만 경우 제거됩니다.

이러한 상황은 백아웃 큐가 정의되지 않았거나 MessageConsumer 오브젝트가 메시지를 백아웃 큐에 송신할 수 없는 경우에 발생합니다.

변조 메시지 핸들링을 수행하도록 시스템 구성

XMS .NET에서 **BOTHRESH** 및 **BOQNAME** 속성을 조회할 때 사용할 큐는 수행하는 메시징 스타일에 따라 달라집니다.

- 포인트-투-포인트 메시징의 경우, 이는 기본 로컬 큐입니다. XMS .NET 애플리케이션이 알리언스 큐 또는 클러스터 큐의 메시지를 이용할 경우에 중요합니다.
- 발행/구독 메시징의 경우, 애플리케이션의 메시지를 보관하기 위해 관리 큐가 작성됩니다. XMS .NET에서는 관리 큐를 조회하여 **BOTHRESH** 및 **BOQNAME** 속성의 값을 판별합니다.

관리 큐는 애플리케이션이 구독하는 토픽 오브젝트와 연관된 모델 큐에서 작성되며, **BOTHRESH** 및 **BOQNAME** 속성의 값을 모델 큐로부터 상속받습니다. 수신 애플리케이션이 지속 가능 구독 또는 지속 불가능 구독을 제공했는지 여부에 따라 사용되는 모델 큐가 달라집니다.

- 지속 가능 구독에 사용되는 모델 큐는 토픽의 **MDURMDL** 속성에 지정됩니다. 이 속성의 기본값은 **SYSTEM.DURABLE.MODEL.QUEUE**입니다.
- 지속 불가능 구독에 사용되는 모델 큐는 **MNDURMDL** 속성에 지정됩니다. **MNDURMDL** 속성의 기본값은 **SYSTEM.NDURABLE.MODEL.QUEUE**입니다.

BOTHRESH 및 **BOQNAME** 속성을 조회할 때 XMS.NET에서는 다음을 수행합니다.

- 로컬 큐 또는 알리어스 큐의 대상 큐를 엽니다.
- **BOTHRESH** 및 **BOQNAME** 속성을 조회합니다.
- 로컬 큐 또는 알리어스 큐의 대상 큐를 닫습니다.

로컬 큐 또는 알리어스 큐의 대상 큐를 열 때 사용되는 열기 옵션은 사용 중인 IBM MQ의 버전에 따라 다릅니다.

- IBM MQ 9.1.0 Fix Pack 4 Long Term Support 이전 및 IBM MQ 9.1.4 Continuous Delivery 이전: 로컬 큐 또는 알리어스 큐의 대상 큐가 클러스터 큐인 경우 XMS.NET에서는 **MQOO_INPUT_AS_Q_DEF**, **MQOO_INQUIRE**, **MQOO_FAIL_IF QUIESCING** 옵션으로 큐를 엽니다. 즉, 수신 애플리케이션을 실행하는 사용자가 클러스터 큐의 로컬 인스턴스를 조회하고 이 인스턴스에 액세스할 수 있는 권한을 갖고 있어야 합니다.

XMS.NET에서는 열기 옵션 **MQOO_INQUIRE** 및 **MQOO_FAIL_IF QUIESCING**을 사용하여 로컬 큐의 다른 모든 유형을 엽니다. XMS.NET에서 속성 값을 조회하려면 수신 애플리케이션을 실행하는 사용자가 로컬 큐에 대한 조회 액세스 권한을 갖고 있습니다.

독성 메시지를 백아웃 재큐잉 큐 또는 큐 관리자의 불활성 메시지 큐로 이동시키려면, 애플리케이션을 실행하는 사용자에게 **put** 권한과 **passall** 권한을 부여해야 합니다.

ASF에서 변조 메시지 핸들링

ASF(Application Server Facilities)를 사용할 때는 **MessageConsumer**가 아닌 **ConnectionConsumer**가 변조 메시지를 처리합니다. **ConnectionConsumer**는 큐의 및 **BackoutQueueQName** **BackoutThreshold** 속성에 따라 메시지를 재큐잉합니다.

애플리케이션이 **ConnectionConsumers**를 사용하는 경우, 메시지가 백아웃되는 환경은 애플리케이션 서버가 제공하는 세션에 달려 있습니다.

- 세션이 **AUTO_ACKNOWLEDGE** 또는 **DUPS_OK_ACKNOWLEDGE**로 트랜잭션되지 않은 경우, 메시지는 시스템 오류 이후에만 또는 애플리케이션이 예상치 못하게 종료된 경우에 백아웃됩니다.
- 세션이 **CLIENT_ACKNOWLEDGE**로 트랜잭션되지 않은 경우, 수신확인되지 않은 메시지는 **Session.recover()**를 호출하는 애플리케이션 서버에 의해 백아웃될 수 있습니다.

일반적으로 **MessageListener** 또는 애플리케이션 서버의 클라이언트 구현에서는 **Message.acknowledge()**를 호출합니다. **Message.acknowledge()**는 지금까지 세션에 전달된 모든 메시지를 수신확인합니다.

- 세션이 트랜잭션되면, 수신확인되지 않은 메시지는 **Session.rollback()**을 호출하는 애플리케이션 서버에 의해 백아웃될 수 있습니다.

큐 브라우저

애플리케이션은 큐 브라우저를 사용하여 큐의 메시지를 제거하지 않고 찾아봅니다.

큐 브라우저를 작성하기 위해, 애플리케이션은 찾을 큐를 식별하는 **Destination** 오브젝트를 매개변수로서 지정하는 **ISession** 오브젝트의 **Create Queue Browser** 메소드를 호출합니다. 애플리케이션은 메시지 선택자를 사용하거나 메시지 선택자 없이 큐 브라우저를 작성할 수 있습니다.

큐 브라우저를 작성한 후 애플리케이션은 **IQueueBrowser** 오브젝트의 **GetEnumerator** 메소드를 호출하여 큐에서 메시지 목록을 가져올 수 있습니다. 이 메소드는 **Message** 오브젝트의 목록을 캡슐화하는 열거자를 리턴합니다. 목록에 있는 **Message** 오브젝트의 순서는 큐에서 메시지를 검색하는 순서와 같습니다. 애플리케이션은 열거자를 사용하여 각 메시지를 교대로 찾아볼 수 있습니다.

열거자는 메시지가 큐에 넣어지고 큐에서 제거될 때 동적으로 업데이트됩니다. 애플리케이션이 큐에서 다음 메시지를 찾기 위해 **IEnumerator.MoveNext()**를 호출할 때마다 메시지는 큐의 현재 컨테츠를 반영합니다.

애플리케이션은 지정된 큐 브라우저에 대해 GetEnumerator 메소드를 두 번 이상 호출할 수 있습니다. 각 호출에서는 새로운 열거자를 리턴합니다. 그러므로 애플리케이션은 둘 이상의 열거자를 사용하여 큐의 메시지를 찾아 보고 큐 내에 여러 위치를 유지보수할 수 있습니다.

애플리케이션은 큐 브라우저를 사용하여 큐에서 제거할 수 있는 적합한 메시지를 검색한 후 메시지 선택자와 함께 메시지 이용자를 사용하여 메시지를 제거할 수 있습니다. 메시지 선택자는 JMSMessageID 헤더 필드의 값에 따라 메시지를 선택할 수 있습니다. 이 헤더 필드 및 기타 JMS 메시지 헤더 필드에 대한 정보는 [633 페이지의 『XMS 메시지의 헤더 필드』](#)의 내용을 참조하십시오.

요청자

애플리케이션은 요청자를 사용하여 요청 메시지를 보낸 후 응답을 대기 및 수신합니다.

대부분의 메시징 애플리케이션은 요청 메시지를 보낸 후 응답을 대기하는 알고리즘을 기반으로 합니다. XMS는 이런 스타일의 애플리케이션 개발에 도움이 되는 Requestor 클래스를 제공합니다.

요청자를 작성하기 위해 애플리케이션은 요청 메시지를 보낼 위치를 식별하는 Destination 및 Session 오브젝트를 매개변수로 지정하여 Requestor 클래스의 Create Requestor 생성자를 호출합니다. 세션을 트랜잭션 처리하지 않고 수신확인 모드인 XMSC_CLIENT_ACKNOWLEDGE도 없어야 합니다. 생성자는 응답 메시지를 보낼 임시 큐 또는 토픽을 자동으로 작성합니다.

요청자가 작성되면 애플리케이션은 Requestor 오브젝트의 Request 메소드를 호출하여 요청 메시지를 보낸 후 요청 메시지를 수신하는 애플리케이션으로부터 응답을 대기 및 수신합니다. 호출은 응답을 수신하거나 세션이 끝나거나 빨리 발생하는 이벤트를 대기합니다. 요청자에서는 각 요청 메시지에 대해 한 개의 응답만 필요합니다.

애플리케이션이 요청자를 닫으면 임시 큐 또는 토픽이 삭제됩니다. 그러나 연관된 세션은 닫히지 않습니다.

오브젝트 삭제

애플리케이션이 작성된 XMS 오브젝트를 삭제하는 경우 XMS는 오브젝트에 할당된 내부 자원을 릴리스합니다.

애플리케이션이 XMS 오브젝트를 작성하는 경우 XMS는 메모리 및 기타 내부 자원을 오브젝트에 할당합니다. XMS는 XMS가 내부 자원을 릴리스할 때 오브젝트의 닫기 또는 삭제 메소드를 호출하여 애플리케이션이 오브젝트를 명시적으로 삭제할 때까지 이러한 내부 자원을 보존합니다. 애플리케이션이 이미 삭제된 오브젝트를 삭제하려고 하면 호출이 무시됩니다.

애플리케이션이 Connection 또는 Session 오브젝트를 삭제하면 XMS는 연관된 특정 오브젝트를 자동으로 삭제하고 내부 자원을 릴리스합니다. 이는 Connection 또는 Session 오브젝트에서 작성한 오브젝트이며 오브젝트에 연동되어 작동합니다. 이러한 오브젝트는 [613 페이지의 표 85](#)에 표시되어 있습니다.

참고: 애플리케이션이 종속 세션과의 연결을 닫으면 해당 세션에 종속되는 모든 오브젝트도 삭제됩니다. Connection 또는 Session 오브젝트에만 종속 오브젝트가 있을 수 있습니다.

표 85. 자동으로 삭제되는 오브젝트		
삭제된 오브젝트	메소드	자동으로 삭제되는 종속 오브젝트
연결	Close Connection	ConnectionMetaData 및 Session 오브젝트
세션	Close Session	MessageConsumer, MessageProducer, QueueBrowser 및 Requestor 오브젝트

데이터 유형 XMS .NET

XMS .NET에서는 System.Boolean, System.Byte, System.SByte, System.Char, System.String, System.Single, System.Double, System.Decimal, System.Int16, System.Int32, System.Int64, System.UInt16, System.UInt32, System.UInt64 및 System.Object를 지원합니다. XMS .NET의 데이터 유형은 XMS C/C++의 데이터 유형과 다릅니다. 이 주제를 사용하여 해당 데이터 유형을 식별할 수 있습니다.

다음 표는 해당 XMS .NET 및 XMS C/C++ 데이터 유형을 표시하고 이에 대해 간단히 설명합니다.

표 86. XMS .NET 및 XMS C/C++의 데이터 유형

XMS .NET 유형	XMS C/C++ 유형	설명
System.SByte	xmsSBYTE xmsINT8	부호가 있는 8비트 값
System.Byte	xmsBYTE xmsUINT8	부호가 없는 8비트 값
System.Int16	xmsINT16 xmsSHORT	부호가 있는 16비트 값
System.UInt16	xmsUINT16 xmsUSHORT	부호가 없는 16비트 값
System.Int32	xmsINT32 xmsINT	부호가 있는 32비트 값
System.UInt32	xmsUINT32 xmsUINT	부호가 없는 32비트 값
System.Int64	xmsLONG xmsINT64	부호가 있는 64비트 값
System.UInt64	xmsULONG xmsUINT64	부호가 없는 64비트 값
System.Char	xmsCHAR16	부호가 없는 16비트 문자(.NET용 유니 코드)
System.Single	xmsFLOAT	IEEE 32비트 Float
System.Double	xmsDOUBLE	IEEE 64비트 Float
System.Boolean	xmsBOOL	True/False 값
적용할 수 없음	xmsCHAR	부호가 있는 또는 부호가 없는 8비트 값 (부호의 유무는 플랫폼에 따라 결정됨)
System.Decimal	적용할 수 없음	부호가 있는 96비트 정수 곱하기 $10^0 - 10^{28}$
System.Object	적용할 수 없음	모든 유형의 기본
System.String	적용할 수 없음	문자열 유형

XMS 기본 유형

XMS에서는 동일한 8개의 Java 기본 유형(byte, short, int, long, float, double, char 및 boolean)을 제공합니다. 이를 사용하면 데이터를 유실하거나 손상되지 않고 XMS 및 JMS 사이에서 메시지를 교환할 수 있습니다.

615 페이지의 표 87에는 Java에 상응하는 데이터 유형, 크기 및 각 XMS 기본 유형의 최소값 및 최대값이 표시됩니다.

표 87. XMS 데이터 유형 및 해당 Java 동일 유형				
XMS 데이터 유형	호환 가능 Java 데이터 유형	크기	최소값	최대값
System.Boolean	부울	32비트	false	true
System.SBYTE	byte	8비트	-2 ⁷ (-128)	2 ⁷ -1(127)
System.BYTE	byte	8비트	-2 ⁷ (-128)	2 ⁷ -1(127)
System.CHAR	byte	8비트	-2 ⁷ (-128)	2 ⁷ -1(127)
System.Int16	short	16비트	-2 ¹⁵ (-32768)	2 ¹⁵ -1(32767)
System.Int32	int	32비트	-2 ³¹ (-2147483648)	2 ³¹ -1(2147483647)
System.Int64	long	64비트	-2 ⁶³ (-9223372036854775808)	2 ⁶³ -1(9223372036854775807)
System.Single	float	32비트	-3.402823E+38(7자리 정밀도까지)	3.402823E+38(7자리 정밀도까지)
System.Double	double	64비트	-1.79769313486231E+308(15자리 정밀도까지)	1.79769313486231E+308(15자리 정밀도까지)

한 데이터 유형에서 다른 데이터 유형으로 특성 값의 암시적 변환

애플리케이션이 특성의 값을 가져오면 값은 XMS에 의해 다른 데이터 유형으로 변환될 수 있습니다. 지원되는 변환 및 XMS의 변환 수행 방법에 적용되는 규칙은 여러 가지가 있습니다.

오브젝트 특성에는 이름과 값이 있으며 값에는 연관된 데이터 유형이 있습니다. 여기서 특성 값은 특성 유형이라고도 합니다.

애플리케이션은 PropertyContext 클래스의 메소드를 사용하여 오브젝트의 특성을 가져오고 설정합니다. 애플리케이션은 특성 값을 가져오기 위해 특성 유형에 적합한 메소드를 호출합니다. 예를 들어 정수 특성 값을 가져오기 위해 애플리케이션은 일반적으로 GetIntProperty 메소드를 호출합니다.

그러나 애플리케이션이 특성의 값을 가져오면 값은 XMS에 의해 다른 데이터 유형으로 변환될 수 있습니다. 예를 들어 정수 특성 값을 가져오기 위해 애플리케이션은 GetStringProperty 메소드를 호출할 수 있습니다. 이는 특성 값을 문자열로 리턴합니다. XMS에서 지원하는 변환은 615 페이지의 표 88에 표시되어 있습니다.

표 88. 특성 유형에서 다른 데이터 유형으로 지원되는 변환	
특성 유형	지원되는 대상 데이터 유형
System.String	System.Boolean, System.Double, System.Float, System.Int32, System.Int64, System.SByte, System.Int16
System.Boolean	System.String, System.SByte, System.Int32, System.Int64, System.Int16
System.Char	System.String
System.Double	System.String
System.Float	System.String, System.Double
System.Int32	System.String, System.Int64
System.Int64	System.String
System.SByte	System.String, System.Int32, System.Int64, System.Int16
System.SByte array	System.String
System.Int16	System.String, System.Int32, System.Int64

지원되는 변환에는 다음 일반 규칙이 적용됩니다.

- 변환 도중 데이터가 손실되지 않으면 숫자 등록 정보 값을 다른 데이터 유형으로 변환할 수 있습니다. 예를 들어 System.Int32 데이터 유형의 특성 값은 System.Int64 데이터 유형의 값으로 변환될 수 있지만, 이는 System.Int16 데이터 유형의 값으로 변환될 수 없습니다.
- 데이터 유형의 특성 값은 문자열로 변환될 수 있습니다.
- 문자열이 변환에 맞게 형식화된 경우 문자열 특성 값은 다른 데이터 유형으로 변환될 수 있습니다. 애플리케이션이 올바르게 형식화되지 않은 문자열 특성 값을 변환하려고 하면 XMS가 오류를 리턴할 수 있습니다.
- 애플리케이션이 지원되지 않는 변환을 시도하면 XMS가 오류를 리턴할 수 있습니다.

다음 규칙은 특성 값이 하나의 데이터 유형에서 다른 데이터 유형으로 변환되는 경우에 적용됩니다.

- Boolean 특성 값을 문자열로 변환하는 경우, True 값은 "true" 문자열로 변환되고, False 값은 "false" 문자열로 변환됩니다.
- System.SByte를 포함하여 Boolean 특성 값을 숫자 데이터 유형으로 변환하는 경우 True 값은 1로 변환되고 False 값은 0으로 변환됩니다.
- 문자열 특성 값을 Boolean 값으로 변환하는 경우, "true" 문자열(대소문자 구분 없음) 또는 "1"은 True로 변환되고 "false" 문자열(대소문자 구분 없음) 또는 "0"은 False로 변환됩니다. 기타 모든 문자열은 변환될 수 없습니다.
- 문자열 특성 값을 System.Int32, System.Int64, System.SByte 또는 System.Int16 데이터 유형의 값으로 변환하는 경우, 문자열은 다음 형식이어야 합니다.

[blanks][sign] digits

문자열 컴포넌트는 다음과 같이 정의됩니다.

공백

선택적 선두 공백 문자입니다.

sign

선택적 더하기 부호(+) 또는 빼기 부호(-) 문자.

digits

연속적인 순서의 숫자 문자(0-9). 하나 이상의 숫자가 있어야 합니다.

문자열은 숫자의 연속 뒤에 숫자가 아닌 다른 문자를 포함할 수 있지만 이러한 문자의 첫 번째에 이르면 변환을 중지합니다. 문자열은 10진수 정수를 표시한다고 가정됩니다.

XMS에서는 문자열이 올바르게 형식화되지 않으면 오류를 리턴할 수 있습니다.

- 문자열 특성 값을 System.Double 또는 System.Float 데이터 유형의 값으로 변환하는 경우, 문자열의 형식은 다음과 같아야 합니다.

[blanks][sign][digits][point[d_digits]][e_char[e_sign] e_digits]

문자열 컴포넌트는 다음과 같이 정의됩니다.

공백

(선택적) 선행 공백 문자.

sign

(선택적) 더하기 부호(+) 또는 빼기 부호(-) 문자.

digits

연속적인 순서의 숫자 문자(0-9). digits 또는 d_digits에는 하나 이상의 숫자가 있어야 합니다.

point

(선택사항) 소수점(.).

d_digits

연속적인 순서의 숫자 문자(0-9). digits 또는 d_digits에는 하나 이상의 숫자가 있어야 합니다.

e_char

지수 문자(E 또는 e)입니다.

e_sign

(선택적) 지수의 더하기 부호(+) 또는 빼기 부호(-) 문자.

e_digits

지수에 대한 연속적인 순서의 숫자 문자(0-9). 문자열에 지수 문자가 포함된 경우 하나 이상의 숫자가 있어야 합니다.

문자열은 숫자의 연속 또는 지수를 나타내는 선택적 문자 뒤에 숫자가 아닌 다른 문자를 포함할 수 있지만 이러한 문자의 첫 번째에 이르면 변환을 중지합니다. 문자열은 10제곱인 지수의 10진수 부동 소수점 숫자를 표시한다고 가정됩니다.

XMS에서는 문자열이 올바르게 형식화되지 않으면 오류를 리턴할 수 있습니다.

- System.SByte 데이터 유형의 특성 값을 포함하는 숫자 특성 값을 문자열로 변환하는 경우, 값은 10진수로서의 값 문자열 표현으로 변환됩니다. 해당 값에 대한 ASCII 문자를 포함하는 문자열로 변환되지 않습니다. 예를 들어, 정수 65는 문자열 "A"가 아닌 문자열 "65"로 변환됩니다.
- 바이트 배열 특성 값을 문자열로 변환하면 각 바이트는 바이트를 표시하는 두 개의 16진 문자로 변환됩니다. 예를 들어 바이트 배열 {0xF1, 0x12, 0x00, 0xFF}은 문자열 "F11200FF"로 변환됩니다.

특성 유형에서 다른 데이터 유형으로의 변환은 Property와 PropertyContext 클래스의 메소드에 의해 지원됩니다.

반복기

반복기는 목록에서 현재 위치를 유지하는 커서와 오브젝트의 목록을 캡슐화합니다. IBM MQ Message Service Client (XMS) for C/C++에서 사용할 수 있는 반복기 개념은 IBM MQ Message Service Client (XMS) for .NET에서 IEnumerator 인터페이스를 사용하여 구현됩니다.

반복기가 작성되면 커서의 위치는 첫번째 오브젝트의 앞에 옵니다. 애플리케이션은 반복기를 사용하여 차례로 각 오브젝트를 검색합니다.

IBM MQ Message Service Client (XMS) for C/C++의 Iterator 클래스는 Java의 Enumerator 클래스와 동일합니다. IBM MQ Message Service Client (XMS) for .NET는 Java와 유사하며 IEnumerator 인터페이스를 사용합니다.

애플리케이션은 IEnumerator를 사용하여 다음 태스크를 수행할 수 있습니다.

- 메시지 특성 가져오기
- 맵 메시지의 본문에서 이름-값 쌍 가져오기
- 큐에서 메시지 찾아보기
- 연결에서 지원되는 JMS 정의 메시지 특성의 이름 가져오기

.NET에서 메시지 및 예외 리스너 사용

.NET 애플리케이션은 메시지 리스너를 사용하여 메시지를 비동기로 수신하고 예외 리스너를 사용하여 연결 문제점을 비동기로 알립니다.

이 태스크 정보

메시지 및 예외 리스너의 기능은 .NET 및 C++에 동일합니다. 하지만 몇 가지 작은 구현의 차이가 있습니다.

프로시저

- 메시지를 비동기식으로 수신하기 위해 메시지 리스너를 설정하려면 다음 단계를 완료하십시오.
 - a) 메시지 리스너 위임의 서명과 일치하는 메소드를 정의하십시오.
정의할 메소드는 정적 또는 인스턴스 메소드가 될 수 있으며 액세스 가능한 클래스에 정의할 수 있습니다. 위임 서명은 다음과 같습니다.

```
public delegate void MessageListener(IMessage msg);
```

그러므로 메소드를 다음으로 정의할 수 있습니다.

```
void SomeMethodName(IMessage msg);
```

b) 다음 예제와 유사한 항목을 사용하여 이 메소드를 위임으로 인스턴스화하십시오.

```
MessageListener OnMsgMethod = new MessageListener(SomeMethodName)
```

c) 다음과 같이 위임을 이용자의 MessageListener 특성으로 설정하여 하나 이상의 이용자에 등록하십시오.

```
consumer.MessageListener = OnMsgMethod;
```

MessageListener를 다시 널로 설정하여 위임을 제거할 수 있습니다.

```
consumer.MessageListener = null;
```

- 예외 리스너를 설정하려면 다음 단계를 완료하십시오.

예외 리스너는 메시지 리스너와 상당히 유사한 방식으로 작동하지만 위임 정의가 다르며 메시지 이용자 이외의 연결에 지정됩니다. 이는 C++에서도 동일합니다.

a) 메소드를 정의하십시오.

위임 서명은 다음과 같습니다.

```
public delegate void ExceptionListener(Exception ex);
```

그러므로 메소드를 다음으로 정의할 수 있습니다.

```
void SomeMethodName(Exception ex);
```

b) 다음 예제와 유사한 항목을 사용하여 이 메소드를 위임으로 인스턴스화하십시오.

```
ExceptionListener OnExMethod = new ExceptionListener(SomeMethodName)
```

c) ExceptionListener 특성을 설정하여 위임을 연결에 등록하십시오.

```
connection.ExceptionListener = OnExMethod ;
```

ExceptionListener를 다음과 같이 재설정하여 위임을 제거할 수 있습니다.

```
null: connection.ExceptionListener = null;
```

XMS을(를) 통한 자동 IBM MQ 클라이언트 재연결

IBM WebSphere MQ 7.1 클라이언트 이상을 메시지 제공자로 사용하는 동안 네트워크, 큐 관리자 또는 서버 실패 후 자동으로 다시 연결하도록 XMS 클라이언트를 구성하십시오.

MQConnectionFactory 클래스의 WMQ_CONNECTION_NAME_LIST 및 WMQ_CLIENT_RECONNECT_OPTIONS 특성을 사용하여 자동으로 다시 연결하도록 클라이언트 연결을 구성하십시오. 클라이언트 자동 재연결은 연결 실패 후 또는 큐 관리자를 중지한 후 옵션으로서 클라이언트를 재연결합니다. 일부 클라이언트 애플리케이션의 디자인은 자동 연결에 적합하지 않습니다.

일단 연결이 설정되면 다시 연결 가능한 클라이언트 연결은 다시 연결 가능으로 자동 전환됩니다.

참고: CCDT(Client Channel Definition Table)를 통해 또는 mqclient.ini 파일을 통해 클라이언트 재연결을 사용하여 클라이언트 재연결 옵션, 클라이언트 재연결 제한시간 및 연결 이름 목록 특성을 설정할 수도 있습니다.

참고: 다시 연결 특성이 CCDT에서와 함께 ConnectionFactory 오브젝트에서 설정되는 경우 서열 규칙은 다음과 같습니다. 연결 이름 목록 특성의 기본 값이 ConnectionFactory 오브젝트에서 설정되면 CCDT가 이에 우선합니다. 연결 이름 목록이 기본값으로 설정되지 않으면, ConnectionFactory 오브젝트에서 설정된 특성 값이 우선합니다. 연결 이름 목록의 기본값은 localhost(1414)입니다.

애플리케이션에서 XMS .NET 클라이언트 연결 유지 동작 사용자 지정 XMS .NET

IBM MQ 10.0.0에서, 큐 관리자가 채널 연결 해제 간격(DISCINT)을 적용할 때 발생할 수 있는 예기치 않은 연결 해제를 방지하기 위해 연결의 XMS 연결 유지 동작을 사용자 정의할 수 있습니다.

클라이언트를 IBM MQXMS.NET 사용할 때:

- 애플리케이션에 의해 생성된 모든 XMS 연결에는 큐 관리자에 대한 연결 핸들이 있습니다.
- 연결을 통해 생성된 모든 XMS 세션에는 각각 고유한 연결 핸들이 있습니다.
- 사양에 JMS 따라 XMS.NET, 애플리케이션이 생성하는 모든 XMSContext에는 두 개의 연결 핸들이 있습니다. 연결 핸들 중 하나는 연결용이고, 다른 하나는 세션용입니다.

애플리케이션과 큐 관리자 간의 모든 통신은 세션과 XMS 연결된 연결 핸들을 통해 이루어집니다. 해당 XMS 연결에 대한 연결 핸들의 활동은 거의 없습니다.

연결이 XMS 속성 **DISCINT** 값이 0 보다 큰 채널을 통해 큐 관리자에 연결된 경우, 큐 관리자는 활동 부족을 이유로 해당 채널의 연결을 끊을 수 있습니다. 이로 인해 애플리케이션에서 이유 코드 2009(MQRC_CONNECTION_BROKEN)가 포함된 IBM MQ XMSException이 예기치 않게 발생합니다.

DISCINT IBM MQ 10.0.0부터, 연결에서 XMS 사용하는 채널 인스턴스가 계속 열려 있도록 하고, 로 인해 닫히지 않도록 보장하는 새로운 메커니즘이 제공됩니다. 이를 통해 애플리케이션의 안정성이 향상됩니다.

기본 동작

기본적으로 클라이언트는 XMS.NET 관련 XMS 세션을 모두 활성 상태로 유지하기 위해 각 XMS 연결에 대해 주기적으로 MQSTAT API 호출을 수행합니다.

- **HBINT** 간격은 채널 속성에 의해 결정됩니다.
- 이 HBINT=0 경우 기본적으로 300초 간격이 사용됩니다.

이 키프 얼라이브(keep alive) 흐름은 큐 관리자가 채널 연결 해제 간격(**DISCINT**)을 적용할 때 발생할 수 있는 예기치 않은 연결 해제를 방지하는 데 도움이 됩니다.

간격 사용자 지정

다음 XMSC 시스템 속성을 설정하여 애플리케이션을 실행하면 기본 간격을 재정의할 수 있습니다:

```
connectionfactory.SetLongProperty(XMSC.WMQ_CONNECTION_KEEP_ALIVE, <value>);
```

이는 해당 값을 'ConnectionFactory' 객체를 사용하여 구성해야 함을 의미합니다.

- 이 값은 필요한 차단 간격을 초 단위로 나타내는 양의 정수여야 합니다.
- 권장 사항: 값을 해당 채널 **DISCINT** 값의 절반으로 설정하세요.
- 애플리케이션이 서로 다른 **DISCINT** 값을 가진 여러 채널을 통해 큐 관리자에 연결하는 경우, 해당 채널들에 설정된 값 중 가장 작은 **DISCINT** 값을 확인한 후, 해당 값의 정확히 절반으로 속성을 설정하십시오.
- 값이 -1 이면 연결 유지 흐름이 XMS 비활성화됩니다.
- **HBINT** 0 또는 -1을 제외한 다른 음수 값은 무시되며, 구간은 에서 도출됩니다.

적용성

이 기능은 버전(및 이후 버전) XMS.NET의 애플리케이션이 IBM MQ 10.0.0 버전 IBM MQ 9.0 (또는 이후 버전)의 큐 관리자에 연결하는 경우에만 사용할 수 있습니다.

키프 얼라이브 호출 확인

검증이나 문제 해결을 위해 애플리케이션의 추적 기능을 활성화할 XMS 수 있습니다:

- 추적 기능이 활성화되면, 각 MQSTAT 호출이 추적 파일에 XMS 기록됩니다.
- 기본값은 false이며, 이 경우 로깅이 비활성화됩니다.

다음 예제는 추적 파일에 XMS 기록되는 로그 메시지를 보여줍니다:

```
" 09:45:41.608940 79200.0007 - d XMS 연결 유지 흐름이 XMS 연결(연결 이름: MQ )에 대해 큐 매니저 <QM>으로 전송  
됨 < 9.20.194.107 (1414)>"
```

설명:

XMS 연결의 모든 XMS 세션이 활성 상태를 유지하도록 하기 위해, MQSTAT API 호출 형태의 XMS 연결 유지 흐름이 지정된 큐 매니저로 전송되었습니다. XMS 연결에 대한 다음 키프 얼라이브 패킷은 약 300초 후에 전송될 수 있습니다.

ACTION:
없음

XMS .NET의 오류 처리

XMS .NET 모든 예외는 System.Exception 에서 파생됩니다.

XMS .NET 예외

XMS 메소드 호출은 MessageFormatException과 같은 특정 XMS 예외, 일반 XMSException 또는 NullReferenceException과 같은 시스템 예외를 처리할 수 있습니다.

이러한 오류 중 어느 것이 발생하더라도, 애플리케이션 요구사항에 맞게 특정 catch 블록이나 일반 System.Exception catch 블록에서 해당 오류를 처리하는 애플리케이션을 작성하십시오.

XMS 오류 및 예외 코드

XMS는 다양한 오류 코드를 사용하여 실패를 표시합니다. 이러한 오류 코드는 릴리스마다 다를 수 있으므로 본 문서에 명시적으로 나열되지 않았습니다. XMS 예외 코드(XMS_X_... 형식)는 XMS 릴리스 사이에서 동일하게 유지되므로 이에 대해서만 설명합니다.

관련 정보

[MQException.NET 클래스](#)

[클라이언트 XMS .NET 라이브러리가 발생시키는 일반적인 SSL 오류 코드](#)

[XMS .NET 애플리케이션에 대한 FFDC 구성](#)

[XMS .NET 애플리케이션 추적](#)

XMS .NET 관래 대상 오브젝트에 대한 작업

이 절의 주제에서는 관리 대상 오브젝트에 대한 정보를 제공합니다. XMS 애플리케이션은 중앙 관리 대상 오브젝트 저장소에서 오브젝트 정의를 검색하여 연결 팩토리 및 목적지를 작성하는 데 사용할 수 있습니다.

이 태스크 정보

이 절에서는 관리 대상 오브젝트를 작성하고 관리하는 데 도움이 되는 정보를 제공하며, XMS에서 지원하는 관리 대상 오브젝트 저장소 유형에 대해 설명합니다. 이 절에서는 XMS 애플리케이션이 관리 대상 오브젝트 저장소에 연결하여 필요한 관리 대상 오브젝트를 검색하는 방법도 설명합니다.

이 절에는 다음 주제가 포함되어 있습니다.

- [620 페이지의 『관리 대상 오브젝트 저장소에 대해 XMS .NET에서 지원되는 유형』](#)
- [621 페이지의 『관리 대상 오브젝트의 XMS .NET 특성 맵핑』](#)
- [623 페이지의 『XMS .NET에서는 관리 대상 ConnectionFactory 오브젝트에 대한 특성이 필요함』](#)
- [624 페이지의 『XMS .NET에서는 관리 대상 Destination 오브젝트에 대한 특성이 필요함』](#)
- [624 페이지의 『XMS .NET에서 관리 대상 오브젝트를 작성함』](#)
- [625 페이지의 『XMS .NET에서 InitialContext 오브젝트를 작성함』](#)
- [625 페이지의 『XMS .NET InitialContext 특성』](#)
- [626 페이지의 『XMS 초기 컨텍스트의 URI 형식』](#)
- [626 페이지의 『XMS .NET의 JNDI 찾아보기 웹 서비스』](#)
- [627 페이지의 『관리 대상 오브젝트의 XMS .NET 검색』](#)

관리 대상 오브젝트 저장소에 대해 XMS .NET에서 지원되는 유형

파일 시스템 및 LDAP 관리 대상 오브젝트는 IBM MQ 및 WebSphere Application Server에 연결할 때 사용될 수 있는 반면, COS 이름 지정은 WebSphere Application Server에 연결할 때에만 사용될 수 있습니다.

파일 시스템 오브젝트 디렉토리는 직렬화된 Java Naming Directory Interface(JNDI) 오브젝트의 양식을 사용합니다. LDAP 오브젝트 디렉토리는 JNDI 오브젝트가 포함되는 디렉토리입니다. 파일 시스템 및 LDAP 객체 디렉토리는 및 이후 IBM MQ 버전에서 제공되는 로 관리할 수 IBM MQ Explorer 있습니다. 파일 시스템과 LDAP 객체 디렉터리 모두 연결 팩토리와 IBM MQ 목적지를 중앙 집중화하여 클라이언트 연결을 관리하는 데 사용할 수 있습니다. 네트워크 관리자는 동일한 중앙 저장소를 참조하고 중앙 저장소에서 작성한 연결 설정의 변경사항을 반영하도록 자동으로 업데이트되는 여러 애플리케이션을 배치할 수 있습니다.

COS 이름 지정 디렉토리에는 WebSphere Application Server service integration bus 연결 팩토리 및 목적지가 포함되며 WebSphere Application Server 관리 콘솔을 사용하여 관리할 수 있습니다. XMS 애플리케이션이 COS 이름 지정 디렉토리에서 오브젝트를 검색하려면 JNDI 검색 웹 서비스가 배치되어야 합니다. 이 웹 서비스는 일부 WebSphere Application Server service integration technologies에서 사용할 수 없습니다. 자세한 내용은 제품 문서를 참조하십시오.

참고: 오브젝트 디렉토리의 변경사항을 적용하려면 애플리케이션 연결을 다시 시작하십시오.

관리 대상 오브젝트의 XMS .NET 특성 맵핑

XMS .NET 애플리케이션이 IBM MQ JMS 및 WebSphere Application Server 연결 팩토리 및 대상 오브젝트 정의를 사용할 수 있도록 하려면, 이러한 정의에서 검색하는 특성이 XMS 연결 팩토리 및 대상에서 설정할 수 있는 해당 XMS 특성으로 맵핑되어야 합니다.

예를 들어, JMS 연결 IBM MQ 팩토리에서 가져온 속성을 가진 연결 XMS 팩토리를 생성하려면 두 팩토리 간에 속성을 매핑해야 합니다.

모든 특성 맵핑은 자동으로 수행됩니다.

621 페이지의 표 89에서는 연결 팩토리 및 대상의 가장 일반적인 일부 특성 사이의 맵핑에 대해 설명합니다. 이 표에 표시된 특성은 일부 예일 뿐이며, 표시된 모든 특성이 연결 유형 및 서버 전체와 관련된 것은 아닙니다.

표 89. 연결 팩토리 및 목적지 특성의 이름 맵핑 예		
IBM MQ JMS 특성 이름	XMS 특성 이름	WebSphere Application Server service integration bus 특성 이름
PERSISTENCE(PER)	<u>XMSC_DELIVERY_MODE</u>	
EXPIRY(EXP)	<u>XMSC_TIME_TO_LIVE</u>	
PRIORITY(PRI)	<u>XMSC_PRIORITY</u>	
	<u>XMSC_WPM_HOST_NAME</u>	serverName
	<u>XMSC_WPM_BUS_NAME</u>	busName
	<u>XMSC_WPM_TOPIC_SPACE</u>	topicName

참고: 621 페이지의 표 90에 표시되는 특성은 JMS와 XMS .NET에도 해당됩니다.

표 90. XMS .NET 특성					
특성	오브젝트 유형				
	CF	QCF	TCF	큐	토픽
<u>APPLICATIONNAME</u>	Y	Y	Y	해당사항 없음	해당사항 없음
<u>ASYNCEXCEPTION</u>	Y	Y	Y	해당사항 없음	해당사항 없음
<u>CCDTURL</u>	Y	Y	Y	해당사항 없음	해당사항 없음
<u>CHANNEL</u>	Y	Y	Y	해당사항 없음	해당사항 없음
<u>CONNECTIONNAMELIST</u>	Y	Y	Y	해당사항 없음	해당사항 없음

표 90. XMS.NET 특성 (계속)

특성	오브젝트 유형				
	CF	QCF	TCF	큐	토픽
CLIENTRECONNECTOPTIONS	Y	Y	Y	해당사항 없음	해당사항 없음
CLIENTRECONNECTTIMEOUT	Y	Y	Y	해당사항 없음	해당사항 없음
CLIENTID	해당사항 없음	Y	해당사항 없음	해당사항 없음	해당사항 없음
컴프드 622 페이지의 『1』	Y	해당사항 없음	Y	해당사항 없음	해당사항 없음
COMPMSG 622 페이지의 『1』	Y	Y	Y	해당사항 없음	해당사항 없음
CONNOPT 622 페이지의 『1』	Y	Y	Y	해당사항 없음	해당사항 없음
CONNTAG 622 페이지의 『1』	Y	Y	Y	해당사항 없음	해당사항 없음
설명 622 페이지의 『1』	해당사항 없음	Y	해당사항 없음	Y	Y
만료 622 페이지의 『1』	해당사항 없음	해당사항 없음	해당사항 없음	Y	Y
FAILIFQUIESCE	Y	Y	Y	Y	Y
HOSTNAME	해당사항 없음	Y	해당사항 없음	해당사항 없음	해당사항 없음
LOCALADDRESSES	해당사항 없음	Y	해당사항 없음	해당사항 없음	해당사항 없음
PERSISTENCE	해당사항 없음	해당사항 없음	해당사항 없음	Y	Y
PORT	해당사항 없음	Y	해당사항 없음	해당사항 없음	해당사항 없음
우선순위 622 페이지의 『1』	해당사항 없음	해당사항 없음	해당사항 없음	Y	Y
PROVIDERVERSION 622 페이지의 『1』	해당사항 없음	Y	해당사항 없음	해당사항 없음	해당사항 없음
QMANAGER	Y	Y	Y	Y	해당사항 없음
큐(queue) 622 페이지의 『1』	해당사항 없음	해당사항 없음	해당사항 없음	Y	해당사항 없음
SHARECONVALLOWED	Y	Y	Y	해당사항 없음	해당사항 없음
주제 622 페이지의 『1』	해당사항 없음	해당사항 없음	해당사항 없음	해당사항 없음	Y
전송 622 페이지의 『1』	해당사항 없음	Y	해당사항 없음	해당사항 없음	해당사항 없음

참고:

1. 해당 특성에는 애플리케이션 레벨 특성이 없지만 관리 특성을 사용하여 선택적으로 설정할 수 있습니다.

OutboundSNI 특성

IBM MQ 9.3.0에서 애플리케이션의 **OutboundSNI** 특성을 설정하는 XMSC_WMQ_OUTBOUND_SNI 특성을 설정할 수 있습니다.

XMSC_WMQ_OUTBOUND_SNI_PROPERTY는 다음 값을 사용합니다.

- "CHANNEL"로 맵핑되는 XMSC_WMQ_OUTBOUND_SNI_CHANNEL
- "HOSTNAME"으로 맵핑되는 XMSC_WMQ_OUTBOUND_SNI_HOSTNAME
- "*"로 맵핑되는 XMSC_WMQ_OUTBOUND_SNI_ASTERISK

또한 다음 값을 사용하는 MQOUTBOUND_SNI 환경 변수를 사용하여 **OutboundSNI** 특성을 설정할 수 있습니다.

- CHANNEL
- 호스트 이름
- *

참고: 특정 값이 설정되지 않은 경우 특성의 기본값은 XMSC_WMQ_OUTBOUND_SNI_CHANNEL로 설정됩니다.

관리 노드에서 **OutboundSNI** 특성을 설정하는 우선순위는 다음과 같습니다.

1. 애플리케이션 레벨 특성
2. 환경 변수

비관리 노드에 있는 **OutboundSNI** 특성의 경우 mqclient.ini만 지원됩니다.

XMS .NET에서는 관리 대상 ConnectionFactory 오브젝트에 대한 특성이 필요함

애플리케이션이 연결 팩토리를 작성하는 경우 메시징 서버로의 연결을 작성하려면 여러 가지 특성이 정의되어야 합니다.

아래 표에 나열된 특성은 애플리케이션이 메시징 서버로의 연결을 작성하도록 설정하는 데 필요한 최소한의 특성입니다. 연결을 작성하는 방법을 사용자 정의하기 위해 애플리케이션이 ConnectionFactory 오브젝트의 추가 특성을 필요에 따라 설정할 수 있습니다. 자세한 정보는 [ConnectionFactory](#) 특성을 참조하십시오. 사용 가능한 특성의 전체 목록이 포함됩니다.

IBM MQ 큐 관리자에 대한 연결

표 91. IBM MQ 큐 관리자에 대한 연결에 필요한 관리 대상 ConnectionFactory 오브젝트의 특성 설정	
필수 XMS 특성	동일한 필수 IBM MQ JMS 특성
XMSC_CONNECTION_TYPE	XMS가 연결 팩토리 클래스 이름 및 TRANSPORT(TRAN) 특성에서 이에 대해 작업합니다.
XMSC_WMQ_HOST_NAME	HOSTNAME(HOST)
XMSC_WMQ_PORT	포트
XMSC_WMQ_QUEUE_MANAGER	큐 관리자의 이름

브로커에 대한 실시간 연결

표 92. 브로커에 대한 실시간 연결에 필요한 관리 대상 ConnectionFactory 오브젝트의 특성 설정	
필수 XMS	동일한 필수 IBM MQ JMS 특성
XMSC_CONNECTION_TYPE	XMS가 연결 팩토리 클래스 이름 및 TRANSPORT(TRAN) 특성에서 이에 대해 작업합니다.
XMSC_RTT_HOST_NAME	HOSTNAME(HOST)

표 92. 브로커에 대한 실시간 연결에 필요한 관리 대상 <i>ConnectionFactory</i> 오브젝트의 특성 설정 (계속)	
필수 XMS	동일한 필수 IBM MQ JMS 특성
<u>XMSC_RTT_PORT</u>	포트

WebSphere Application Server service integration bus에 대한 연결

표 93. <i>WebSphere Application Server service integration bus</i> 에 대한 연결에 필요한 관리 대상 <i>ConnectionFactory</i> 오브젝트의 특성 설정	
XMS 특성	설명
<u>XMSC_CONNECTION_TYPE</u>	애플리케이션이 연결되는 메시징 서버의 유형입니다. 이는 연결 팩토리 클래스 이름에서 결정됩니다.
<u>XMSC_WPM_BUS_NAME</u>	연결 팩토리의 경우, 애플리케이션이 연결하는 서비스 통합 버스의 이름이거나, 또는 목적지의 경우에는 목적지가 존재하는 서비스 통합 버스의 이름입니다.

XMS .NET에서는 관리 대상 Destination 오브젝트에 대한 특성이 필요함

목적지를 작성하는 애플리케이션은 관리 대상 Destination 오브젝트에 대한 여러 특성을 설정해야 합니다.

표 94. 관리 대상 <i>Destination</i> 오브젝트의 특성 설정		
연결 유형	특성	설명
IBM MQ 큐 관리자	QUEUE(QU)	연결하려는 큐
	TOPIC(TOP)	애플리케이션이 목적지로 사용하는 토픽
브로커에 대한 실시간 연결	TOPIC(TOP)	애플리케이션이 목적지로 사용하는 토픽
WebSphere Application Server service integration bus	topicName	애플리케이션이 토픽에 연결되는 경우
	queueName	애플리케이션이 큐에 연결되는 경우

XMS .NET에서 관리 대상 오브젝트를 작성함

XMS 애플리케이션이 메시징 서버에 연결하는 데 필요한 *ConnectionFactory* 및 *Destination* 오브젝트 정의는 적절한 관리 도구를 사용하여 작성해야 합니다.

시작하기 전에

XMS가 지원하는 여러 유형의 관리 대상 오브젝트 저장소에 대한 자세한 정보는 620 페이지의 『[관리 대상 오브젝트 저장소에 대해 XMS .NET에서 지원되는 유형](#)』의 내용을 참조하십시오.

이 태스크 정보

IBM MQ의 관리 대상 오브젝트를 작성하려면 IBM MQ Explorer 또는 IBM MQ JMS 관리(JMSAdmin) 도구를 사용하십시오.

IBM MQ 또는 IBM Integration Bus의 관리 대상 오브젝트를 작성하려면 IBM MQ JMS 관리(JMSAdmin) 도구를 사용하십시오.

WebSphere Application Server service integration bus에 대한 관리 대상 오브젝트를 작성하려면 WebSphere Application Server 관리 콘솔을 사용하십시오.

관리 도구에서 특성을 간단히 **APPLICATIONNAME** 또는 **APPNAME**이라고 합니다.

참고: TRANSPORT(UNMANAGED)를 설정할 때 JMSAdmin을 사용할 수 없습니다. 그러므로 관리를 위해 선택된 애플리케이션 이름을 사용하여 관리되지 않은 XMS 클라이언트를 가져오려면 다음 명령을 입력해야 합니다.

```
cf.SetIntProperty(XMSC.WMQ_CONNECTION_MODE, XMSC.WMQ_CM_CLIENT_UNMANAGED);
```

아래 단계는 관리 대상 오브젝트 작성을 위해 수행하는 작업이 요약되어 있습니다.

프로시저

1. 연결 팩토리를 작성하고 애플리케이션에서 선택한 서버로의 연결을 작성하는 데 필요한 특성을 정의하십시오.

XMS가 연결을 작성하는 데 필요한 최소 특성은 623 페이지의 『XMS.NET에서는 관리 대상 [ConnectionFactory](#) 오브젝트에 대한 특성이 필요함』에 정의되어 있습니다.

2. 애플리케이션이 연결하는 메시징 서버에서 필수 목적지를 작성하십시오.

- IBM MQ 큐 관리자에 대한 연결의 경우에는 큐 또는 토픽을 작성하십시오.
- 브로커에 대한 실시간 연결의 경우 토픽을 작성하십시오.
- WebSphere Application Server service integration bus에 연결할 경우 큐 또는 토픽을 작성하십시오.

XMS가 연결을 작성하는 데 필요한 최소 특성은 624 페이지의 『XMS.NET에서는 관리 대상 [Destination](#) 오브젝트에 대한 특성이 필요함』에 정의되어 있습니다.

XMS.NET에서 InitialContext 오브젝트를 작성함

애플리케이션은 필수 관리 대상 오브젝트를 검색할 수 있도록 관리 대상 오브젝트 저장소에 대한 연결을 작성하는 데 사용되는 초기 컨텍스트를 작성해야 합니다.

이 태스크 정보

InitialContext 오브젝트는 저장소에 대한 연결을 캡슐화합니다. XMS API는 다음 태스크를 수행하는 메소드를 제공합니다.

- InitialContext 오브젝트 작성
- 관리 대상 오브젝트 저장소에서 관리 대상 오브젝트 검색

프로시저

- InitialContext 오브젝트 작성에 대한 자세한 정보는 .NET 및 [InitialContext](#)의 특성에 대한 [InitialContext](#)를 참조하십시오.

XMS.NET InitialContext 특성

InitialContext 생성자의 매개변수에는 URI(Uniform Resource Indicator) 형식으로 제공되는 관리 대상 오브젝트 저장소 위치가 포함됩니다. 애플리케이션에서 저장소에 연결하려면 URI에 포함된 정보 외에 추가 정보를 제공해야 합니다.

JNDI 및 XMS의 .NET 구현에서 추가 정보는 환경 Hashtable에서 생성자에 제공됩니다.

관리 대상 오브젝트 저장소의 위치는 [XMSC_IC_URL](#) 특성에 정의되어 있습니다. 이 특성은 일반적으로 Create 호출에서 전달되지만 검색 전에 다른 이름 지정 디렉토리에 연결하기 위해 수정될 수도 있습니다. FileSystem 또는 LDAP 컨텍스트에서는 이 특성이 디렉토리의 주소를 정의합니다. COS 이름 지정의 경우 이 특성은 JNDI 디렉토리에 연결하기 위해 이들 특성을 사용하는 웹 서비스의 주소입니다.

다음 특성은 JNDI 디렉토리에 연결하는 데 사용할 웹 서비스에 수정되지 않은 채로 전달된다.

- [XMSC_IC_PROVIDER_URL](#)
- [XMSC_IC_SECURITY_CREDENTIALS](#)
- [XMSC_IC_SECURITY_AUTHENTICATION](#)
- [XMSC_IC_SECURITY_PRINCIPAL](#)
- [XMSC_IC_SECURITY_PROTOCOL](#)

XMS 초기 컨텍스트의 URI 형식

관리 대상 오브젝트의 저장소 위치는 URI(Uniform Resource Indicator)로 제공됩니다. URI의 형식은 컨텍스트 유형에 따라 다릅니다.

FileSystem 컨텍스트

FileSystem 컨텍스트의 경우 URL은 파일 시스템 기반 디렉토리의 위치를 제공합니다. URL의 구조는 RFC 1738, *URL(Uniform Resource Locator)*로 정의됩니다. URL에는 `file://` 접두부가 있으며 이 접두부 다음에 나오는 구문은 XMS가 실행 중인 시스템에서 열릴 수 있는 유효한 파일 정의입니다.

이 구문은 플랫폼에 따라 달라질 수 있으며 '/' 구분 기호 또는 '\' 구분 기호를 사용할 수 있습니다. '\'를 사용하는 경우 각 구분 기호는 '\'를 하나 더 사용하여 이스케이프되어야 합니다. 그러면 .NET 프레임워크가 다음과 같은 이 유로 구분 기호를 이스케이프 문자로서 해석하지 않게 됩니다.

이 예는 다음 구문을 설명합니다.

```
file://myBindings
file:///admin/.bindings
file://\admin\bindings
file://c:/admin/.bindings
file://c:\admin\bindings
file://\\madison\shared\admin\bindings
file:///usr/admin/.bindings
```

LDAP 컨텍스트

LDAP 컨텍스트의 경우 URL의 기본 구조는 대소문자를 구분하지 않는 접두부 `ldap://`를 사용하는 RFC 2255, *LDAP URL* 형식으로 정의됩니다.

다음 예제에는 정확한 구문이 설명되어 있습니다.

```
LDAP://[Hostname][:Port]["/"[DistinguishedName]]
```

이 구문은 RFC로 정의되어 있으나 구문에 속성, 범위, 필터 또는 확장자에 대한 지원이 없습니다.

이 구문의 예제는 다음과 같습니다.

```
ldap://madison:389/cn=JMSData,dc=IBM,dc=UK
ldap://madison/cn=JMSData,dc=IBM,dc=UK
LDAP:///cn=JMSData,dc=IBM,dc=UK
```

WSS 컨텍스트

WSS 컨텍스트의 경우 URL은 `http://` 접두부가 있는 웹 서비스 엔드 포인트 양식입니다.

또는 `cosnaming://`이나 `wsvc://` 접두부를 사용할 수도 있습니다.

이 두 접두부는 `http`를 통해 액세스되는 URL이 포함된 WSS 컨텍스트가 사용 중인 것으로 해석됩니다. 이를 통해 URL에서 직접 초기 컨텍스트 유형을 쉽게 파생시킬 수 있습니다.

이 구문의 예를 들면 다음과 같습니다.

```
http://madison.ibm.com:9080/xmsjndi/services/JndiLookup
cosnaming://madison/jndilookup
```

XMS .NET의 JNDI 찾아보기 웹 서비스

XMS에서 COS 이름 지정 디렉토리에 액세스하려면 JNDI 검색 웹 서비스가 WebSphere Application Server service integration bus 서버에 배치되어야 합니다. 이 웹 서비스는 COS 이름 지정 서비스의 Java 정보를 XMS 애플리케이션이 읽을 수 있는 양식으로 변환합니다.

웹 서비스는 설치 디렉토리에 있는 엔터프라이즈 아카이브 파일 `SIBXJndiLookupEAR.ear`에서 제공됩니다. 현재 IBM MQ Message Service Client (XMS) for .NET 릴리스의 경우, `SIBXJndiLookupEAR.ear`은

`install_dir\java\lib` 디렉토리에 있습니다. 이는 관리 콘솔 또는 `wsaadmin` 스크립팅 도구를 사용하여 WebSphere Application Server service integration bus 서버에 설치될 수 있습니다. 웹 서비스 애플리케이션 배치에 대한 추가 정보는 제품 문서를 참조하십시오.

XMS 애플리케이션 내에서 웹 서비스를 정의하려면 `InitialContext` 오브젝트의 `XMSC_IC_URL` 특성을 웹 서비스 엔드포인트 URL로 설정해야 합니다. 예를 들어 웹 서비스가 `MyServer`라는 서버 호스트에 배치되면 웹 서비스 엔드 포인트 URL의 예제는 다음과 같습니다.

```
wsvc://MyHost:9080/SIBXJndiLookup/services/JndiLookup
```

`XMSC_IC_URL` 특성을 설정하면 `InitialContext` 검색 호출이 정의된 엔드 포인트에서 웹 서비스를 호출할 수 있습니다. 그러면 이는 다시 COS 이름 지정 서비스에서 필요한 관리 대상 오브젝트를 검색합니다.

.NET 애플리케이션은 웹 서비스를 사용할 수 있습니다. 서버측 배치는 XMS C, /C++ 및 XMS .NET에서 동일합니다. XMS .NET는 Microsoft .NET Framework를 통해 직접 웹 서비스를 호출합니다.

관리 대상 오브젝트의 XMS .NET 검색

XMS는 `InitialContext` 오브젝트가 작성될 때 제공된 주소 또는 `InitialContext` 특성의 주소를 사용하여 저장소에서 관리 대상 오브젝트를 검색합니다.

검색될 오브젝트의 이름 유형은 다음과 같습니다.

- Destination 오브젝트를 설명하는 단순한 이름. 큐 목적지 이름인 `SalesOrders`를 예로 들 수 있습니다.
- SubContext로 구성될 수 있는 콤비네이션 이름. '/'로 구분되며 오브젝트 이름으로 끝나야 합니다. "Warehouse/PickLists/DispatchQueue2" 콤비네이션 이름을 예로 들 수 있습니다. 여기서 Warehouse와 Picklists는 이름 지정 디렉토리의 서브컨텍스트이며 DispatchQueue2는 Destination 오브젝트의 이름입니다.

애플리케이션에서 신규 XMS 버전 사용 금지

기본적으로 신규 XMS 버전이 설치되면, 이전 버전을 사용하는 애플리케이션은 다시 컴파일하지 않고 자동으로 신규 버전으로 전환됩니다. 그러나 애플리케이션 구성 파일에 있는 속성을 설정하면 애플리케이션이 신규 버전을 사용하지 못하도록 금지할 수 있습니다.

이 태스크 정보

다중 버전 공존 기능을 사용하면 신규 XMS 버전이 설치되어도 이전 XMS 버전을 덮어쓰지 않습니다. 대신 유사한 XMS .NET의 여러 인스턴스 어셈블리들이 GAC(Global Assembly Cache)에 공존하며 이들의 버전 번호는 다릅니다. 내부적으로 GAC는 정책 파일을 사용하여 애플리케이션 호출을 최신 버전의 XMS로 라우팅합니다. 다시 컴파일할 필요 없이 애플리케이션이 실행되고 신규 XMS .NET 버전에서 새 기능을 사용할 수 있습니다.

프로시저

- 이전 XMS .NET 버전을 사용하는 데 애플리케이션이 필요한 경우 애플리케이션 구성 파일에서 `publisherpolicy` 속성을 `no(으)`로 설정하십시오.

참고: 애플리케이션 구성 파일은 접미어가 `.config`이고 파일이 관련되는 실행 프로그램 이름으로 구성되는 파일입니다. 예를 들어 `text.exe`에 대한 애플리케이션 구성 파일의 이름은 `text.exe.config`가 됩니다.

그러나 언제든지 시스템의 모든 애플리케이션은 동일한 버전의 XMS .NET를 사용합니다.

XMS 애플리케이션의 통신 보안 설정

이 절에서는 XMS 애플리케이션이 SSL(Secure Sockets Layer)을 통해 WebSphere Application Server service integration bus 메시징 엔진 또는 IBM MQ 큐 관리자에 연결할 수 있도록 하는 보안 통신 설정에 대한 정보를 제공합니다.

이 태스크 정보

이 절에는 다음 주제가 포함되어 있습니다.

- [628 페이지의 『IBM MQ 큐 관리자에 대한 보안 연결』](#)

- 628 페이지의 『CipherSuite CipherSpec 큐 IBM MQ 매니저에 대한 연결을 XMS 위한 이름 매핑』
- 629 페이지의 『WebSphere Application Server service integration bus 메시징 엔진에 대한 보안 연결』
- 630 페이지의 『WebSphere Application Server service integration bus에 대한 연결에 필요한 CipherSuite 및 CipherSpec 이름 매핑』

IBM MQ 큐 관리자에 대한 보안 연결

XMS .NET 애플리케이션이 IBM MQ 큐 관리자에 대한 보안 연결을 작성할 수 있도록 하려면 ConnectionFactory 오브젝트에서 관련 특성을 정의해야 합니다.

암호화 조정에 사용되는 프로토콜은 ConnectionFactory 오브젝트에 지정되는 CipherSuite에 따라 SSL(Secure Sockets Layer) 또는 TLS(Transport Layer Security)가 될 수도 있습니다.

ConnectionFactory SSL 를 사용하여 큐 IBM MQ 매니저에 연결할 때의 속성과 간단한 설명은 다음 표에 나와 있습니다:

표 95. SSL 또는 TLS 을 통해 큐 관리자(IBM MQ queue manager)에 연결할 때의 ConnectionFactory 속성	
특성 이름	설명
XMSC_WMQ_SSL_CERT_STORES	큐 관리자에 대한 SSL(Secure Socket Layer) 연결에 사용되는 인증서 폐기 목록(CRL)이 있는 서버의 위치입니다.
XMSC_WMQ_SSL_CIPHER_SPEC	큐 관리자에 대한 보안 연결에 사용할 CipherSpec의 이름.
XMSC_WMQ_SSL_CIPHER_SUITE	큐 관리자에 대한 TLS 연결에 사용될 CipherSuite의 이름입니다. 보안 연결 협상에 사용되는 프로토콜은 지정된 CipherSuite에 따라 달라집니다.
XMSC_WMQ_SSL_CRYPTO_HW	클라이언트 시스템에 연결된 암호화 하드웨어의 구성 세부사항입니다.
XMSC_WMQ_SSL_FIPS_REQUIRED	이 속성의 값에 따라 애플리케이션이 비준수 FIPS 암호 모음을 사용할 수 있는지 여부가 결정됩니다. 이 속성이 true로 설정된 경우, 클라이언트-서버 연결에는 해당 알고리즘만 FIPS 사용됩니다.
XMSC_WMQ_SSL_키 V 10.0.0 저장소1	키 및 인증서가 저장되는 키 데이터베이스 파일의 위치입니다.
XMSC_WMQ_SSL_KEY_RESETCOUNT	KeyResetCount는 비밀 키가 재협상되기 전에 SSL 통신에서 송신되고 수신된 암호화되지 않은 총 바이트 수를 나타냅니다.
XMSC_WMQ_SSL_PEER_NAME	큐 관리자에 대한 SSL(Secure Socket Layer) 연결에 사용되는 피어 이름입니다.
V 10.0.0 X MSC_WMQ_SSL_키 저장소 암호 1	핸드셰이크 전에 TLS 인증서를 열 때 필요한 비밀번호를 클라이언트가 설정할 수 있는 속성 이름입니다.
V 10.0.0 X MSC_WMQ 암호화 키1	클라이언트가 암호화 키 파일이 저장된 위치를 설정하기 위한 속성 이름입니다.

참고: V 10.0.0 부터 IBM MQ 10.0.0, XMS.NET 애플리케이션은 사용자 지정 경로를 제공함으로써 TLS 연결에 대해 별도의 인증서 경로를 사용할 수 있습니다. Dotnet Password Encryptor(rundotnetcred) 도구를 사용하여 암호화된 암호를 제공하거나 일반 텍스트 암호로도 제공할 수 있습니다. 자세한 내용은 [TLS 연결 시 일반 텍스트 또는 암호화된 암호를 사용하는 경우 XMS.NET의 인증서 사용자 지정 경로를 참조하십시오.](#)

CipherSuite CipherSpec 큐 IBM MQ 매니저에 대한 연결을 XMS 위한 이름 매핑

InitialContext는 JMSAdmin Connection Factory 특성 SSLCIPHERSUITE와 XMS와 거의 동일한 XMSC_WMQ_SSL_CIPHER_SPEC 사이에서 변환됩니다. XMSC_WMQ_SSL_CIPHER_SUITE 값은 지정하지만 XMSC_WMQ_SSL_CIPHER_SPEC 값은 생략하는 경우 유사한 변환이 필요합니다.

629 페이지의 표 96에서는 사용 가능한 CipherSpecs 및 해당하는 동등한 JSSE CipherSuite를 나열합니다.

표 96. 사용 가능한 CipherSpec 및 해당하는 동등한 JSSE CipherSuite	
CipherSpec	동등한 JSSE CipherSuite
TLS_RSA_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA

참고: **Deprecated** TLS_RSA_WITH_3DES_EDE_CBC_SHA는 더 이상 사용되지 않습니다. 그러나 AMQ9288 오류로 인해 연결이 종료되기 전까지는 이 CipherSpec을 사용하여 최대 32GB의 데이터를 전송할 수 있습니다. 이 오류를 방지하려면 3중 DES를 사용하지 않거나 이 CipherSpec을 사용할 때 비밀 키 재설정을 사용 가능하게 하십시오.

WebSphere Application Server service integration bus 메시징 엔진에 대한 보안 연결

XMS .NET 애플리케이션이 WebSphere Application Server service integration bus 메시징 엔진에 대한 보안 연결을 작성할 수 있도록 하려면 ConnectionFactory 오브젝트에서 관련 특성을 정의해야 합니다.

XMS는 WebSphere Application Server service integration bus에 연결하는 데 필요한 SSL(Secure Socket Layer) 및 HTTPS 지원을 제공합니다. SSL 및 HTTPS는 인증 및 기밀성을 위해 보안 연결을 제공합니다.

WebSphere 보안과 유사하게 XMS 보안은 JSSE 보안 표준 및 이름 지정 규칙을 준수하도록 구성되며, 여기에는 보안 연결을 조정할 때 사용되는 알고리즘을 지정하는 CipherSuite 사용이 포함됩니다. 암호화 조정에 사용되는 프로토콜은 ConnectionFactory 오브젝트에 지정되는 CipherSuite에 따라 SSL 또는 TLS가 될 수 있습니다.

629 페이지의 표 97에는 ConnectionFactory 오브젝트에 정의되어야 하는 특성이 나열되어 있습니다.

표 97. WebSphere Application Server service integration bus 메시징 엔진에 대한 보안 연결의 ConnectionFactory 특성	
특성 이름	설명
<u>XMSC_WPM_SSL_CIPHER_SUITE</u>	메시징 엔진에 WebSphere Application Server service integration bus 대한 TLS 연결에 사용될 CipherSuite 의 이름입니다. 보안 연결 협상에 사용되는 프로토콜은 지정된 CipherSuite에 따라 달라집니다.
<u>XMSC_WPM_SSL_KEYRING_LABEL</u>	서버 인증 시 사용되는 인증서입니다.

다음은 WebSphere Application Server service integration bus 메시징 엔진에 대한 보안 연결의 ConnectionFactory 특성 예제입니다.

```
cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS, host_name:port_number:chain_name);
cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY, key_repository_pathname);
cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN, transport_chain);
cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE, cipher_suite);
cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE, stash_file_pathname);
```

여기서 chain_name은 BootstrapTunneledSecureMessaging 또는 BootstrapSecureMessaging 중 하나로 설정되어야 하고, port_number는 부트스트랩 서버가 수신 요청을 청취하는 포트 번호입니다.

다음은 샘플 값이 삽입된 WebSphere Application Server service integration bus 메시징 엔진에 대한 보안 연결의 ConnectionFactory 특성 예제입니다.

```
/* CF properties needed for an SSL connection */
cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS, "localhost:7286:BootstrapSecureMessaging");
cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN, "InboundSecureMessaging");
cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY, "C:\\Program Files\\IBM\\gsk7\\bin\\
XMSkey.kdb");
cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE, "C:\\Program Files\\IBM\\gsk7\\bin\\
```

```
\XMSkey.sth");
cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE, "SSL_RSA_EXPORT_WITH_RC4_40_MD5");
```

WebSphere Application Server service integration bus에 대한 연결에 필요한 CipherSuite 및 CipherSpec 이름 매핑

왜냐하면 IBM Global Security Kit (GSKit) CipherSuite 대신 CipherSpecs 사용하는 경우 CipherSuites, XMSC_WPM_SSL_CIPHER_SUITE 속성에 지정된 JSSE 스타일 CipherSuite 이름은 CipherSpecs에 매핑되어야 합니다. GSKit -스타일 CipherSpec 이름입니다.

630 페이지의 표 98에서는 인식된 각 CipherSuite에 대해 동등한 CipherSpec을 나열합니다.

표 98. 사용 가능한 CipherSuite 및 해당하는 동등한 CipherSpec	
CipherSuite	동등한 CipherSpec
TLS_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA

참고: Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA는 더 이상 사용되지 않습니다. 그러나 AMQ9288 오류로 인해 연결이 종료되기 전까지는 이 CipherSpec을 사용하여 최대 32GB의 데이터를 전송할 수 있습니다. 이 오류를 방지하려면 3중 DES를 사용하지 않거나 이 CipherSpec을 사용할 때 비밀 키 재설정을 사용 가능하게 하십시오.

TLS 연결 시 XMS .NET 평문 또는 암호화된 비밀번호 사용을 위한 인증서 사용자 지정 경로

애플리케이션에서 IBM MQ 10.0.0XMS .NET, 키스토어에 접근하기 위해 평문 또는 암호화된 비밀번호를 사용하여 TLS 연결을 수행할 인증서를 .NET 가져오도록 파일 시스템에서 사용자 지정 경로를 설정할 수 있습니다.

TLS 구성에 사용되는 속성

IBM MQ 10.0.0설정이전에, 다음 예시와 같이 속성을 **XMSC.WMQ_SSL_KEY_REPOSITORY** 사용하여 키 저장소 위치를 또는 *USER*SYSTEM로 설정할 수 있습니다:

```
properties.Add(XMSC.WMQ_SSL_KEY_REPOSITORY, "*SYSTEM")
```

또는

```
properties.Add(XMSC.WMQ_SSL_KEY_REPOSITORY, "*USER")
```

*SYSTEM은 해당 위치가 로컬 컴퓨터임을 나타내고, *USER는 해당 위치가 현재 사용자임을 나타냅니다.

TLS 연결을 설정하기 위한 인증서의 사용자 지정 경로를 지정하려면 속성과 **XMSC.WMQ_SSL_KEY_REPOSITORY** 함께 다음 두 속성을 추가로 사용할 수 있습니다 IBM MQ 10.0.0:

- 키스토어(.p12 파일)의 경로를. **XMSC.WMQ_SSL_KEY_REPOSITORY**을 사용하여 지정합니다.
- 키스토어 접근 비밀번호를 사용하여 **XMSC.WMQ_SSL_KEY_REPOSITORY_PASSWORD**, 평문 또는 암호화된 형태로.
- 암호화된 비밀번호를 **XMSC.WMQ_ENCRYPTION_KEY** 복호화하기 위해 선택적으로 사용할 수 있는 암호화 키 파일 경로.

다음 예시는 사용자 지정 경로 기능을 사용할 때 및 **XMSC.WMQ_SSL_KEY_REPOSITORY_PASSWORD** **XMSC.WMQ_SSL_KEY_REPOSITORY** 속성을 사용하여 일반 텍스트 비밀번호를 설정하는 방법을 보여줍니다:

- ▶ **Windows** Windows의 경우:

```
properties.Add(XMSC.WMQ_SSL_KEY_REPOSITORY, "C:\\CustomPath_Certificates\\myapp.p12")
properties.Add(XMSC.WMQ_SSL_KEY_REPOSITORY_PASSWORD, "123456")
```

- ▶ **Linux** Linux의 경우:

```
properties.Add(XMSC.WMQ_SSL_KEY_REPOSITORY, "/root/Certs/myapp.p12");
properties.Add(XMSC.WMQ_SSL_KEY_REPOSITORY_PASSWORD, "123456");
```

where *myapp.p12* 는 인증서를 포함하는 키 저장소의 이름입니다. 지정하는 경로에 이 키스토어 이름을 포함해야 합니다.

TLS 연결에 암호화된 암호를 사용하려면, 클라이언트 .NET 애플리케이션에 암호화 키를 지정하기 위해 추가 속성인 **XMSC.WMQ_ENCRYPTION_KEY** `encryptionKey` 를 제공해야 합니다. 다음 예시는 사용자 지정 경로 기능을 사용할 때, **XMSC.WMQ_SSL_KEY_REPOSITORY_PASSWORD**, **XMSC.WMQ_SSL_KEY_REPOSITORY**, 및 **XMSC.WMQ_ENCRYPTION_KEY** 속성을 사용하여 암호화된 암호를 설정하는 방법을 보여줍니다:

- ▶ **Windows** Windows의 경우:

```
properties.Add(XMSC.WMQ_SSL_KEY_REPOSITORY, "C:\\CustomPath_Certificates\\myapp.p12")
properties.Add(XMSC.WMQ_SSL_KEY_REPOSITORY_PASSWORD, "<DOTNET>2!KJL4YP2sZ/
Xu8cf6DgWOF1h+inu8nnYUzo8K1KRJH4U=!YMSXlqtjEtcinz8P")
properties.Add(XMSC.WMQ_ENCRYPTION_KEY, "C:\\CustomPath_Certificates\\EncryptionKey\\
\\mykey.txt")
```

- ▶ **Linux** Linux의 경우:

```
properties.Add(XMSC.WMQ_SSL_KEY_REPOSITORY, "/root/Certs/myapp.p12");
properties.Add(XMSC.WMQ_SSL_KEY_REPOSITORY_PASSWORD, "<DOTNET>2!KJL4YP2sZ/
Xu8cf6DgWOF1h+inu8nnYUzo8K1KRJH4U=!YMSXlqtjEtcinz8P");
properties.Add(XMSC.WMQ_ENCRYPTION_KEY, "C:\\CustomPath_Certificates\\EncryptionKey\\
\\mykey.txt")
```

설명:

- 첫 번째 코드 줄은 키스토어가 포함된 위치를 필요로 하는 현재 재사용되는 속성입니다. 두 번째 코드 줄은 새로운 비밀번호 속성입니다. 세 번째 코드 줄은 키가 포함된 위치를 필요로 하는 새로운 암호화 키 속성입니다.
- myapp.p12* 경로에 존재하는 키스토어 이름(인증서를 포함)이며, 경로와 함께 제공되어야 합니다.

참고: 클라이언트 .NET 애플리케이션에서 암호화 키 속성이 설정되지 않은 경우, 라이브러리는 비밀번호 속성에 설정된 값이 평문인지 또는 Base64 인코딩 형식인지 확인한 후 애플리케이션은 TLS 연결을 진행합니다:

- 값이 일반 텍스트로 되어 있다면, 이를 일반 텍스트 비밀번호로 처리합니다.
- 값이 Base64 로 인코딩된 경우, 기본 키를 사용하여 해당 값을 복호화하고 TLS 연결을 구축합니다.

닷넷 암호화 도구 (rundotnetcred)

이 **rundotnetcred** 도구를 사용하면 제공된 암호화 키로 일반 텍스트 키스토어 비밀번호를 처리하여 암호화된 비밀번호를 생성할 수 있습니다. 이 암호화된 비밀번호는 TLS 연결의 보안을 강화하는 데 사용할 수 있습니다.



주의: 이 **rundotnetcred** 도구는 평문 비밀번호를 암호화하기 위해 고급 암호화 표준(AES) 알고리즘을 사용하므로, 128비트 또는 256비트 키를 입력으로 제공해야만 AES-128-CBC 또는 AES-256-CBC 암호화를 활성화할 수 있습니다.

명령줄을 사용하여 도구를 **rundotnetcred** 실행하십시오:

```
<Program-name> -p <PT-pass> -sf <Encryptionkey-filepath>
```

여기서,

- **-p** 일반 텍스트 비밀번호입니다.
- **-sf** 암호화 키의 파일 경로입니다.

참고: **-sf** 필수 매개변수가 아니며, 설정되지 않은 경우 도구는 평문 비밀번호를 암호화하기 위해 기본 암호화 키를 사용합니다. 그러나 자체 암호화 키를 사용하는 것이 권장됩니다. 애플리케이션의 `.NET 'encryption' ENCRYPTION_KEY_PROPERTY.NET` 속성에 (에서 사용된 것과 **rundotnetcred** 동일한) 암호화 키를 반드시 사용하십시오.

암호화된 문자열이 `SSL_CERT_STORE_PASSWORD_PROPERTY`이며, 키 파일 경로는 `ENCRYPTION_KEY_PROPERTY`에 설정됩니다. 암호화 키 값은 주로 평문 비밀번호를 암호화하는 데 필요한 암호화 키를 포함하는 파일의 위치입니다.

다음 예제는 도구를 **rundotnetcred** 사용하는 방법을 보여줍니다:

```
rundotnetcred -p APlaintextPassword -sf C:\temp\Key\EncryptionKey.txt
```

설명:

- **rundotnetcred** 응용 프로그램의 이름입니다.
- `APlaintextPassword` TLS 연결을 위해 암호화해야 하는 일반 텍스트 비밀번호입니다.
- `C:\temp\Key\EncryptionKey.txt` AES 암호화에 사용되는 암호화 키의 경로입니다.

이 유틸리티의 출력 형식은 다음과 같습니다:

```
<DOTNET>2!iS9ccKTjsZ35SbqHQzQ3yKfodnNeId5UnWreJ99v+/0=!SGzPyIJnWaFacu5h
```

다양한 문자열의 조합입니다:

- 눈길을 끄는 것
- 알고리즘 유형
- 분리자
- Base64-sfncoded 암호화된 비밀번호
- 분리자
- 벡터 (IV)

오류 처리 및 추적

다음 MQRC 오류 코드는 사용자 지정 경로 기능과 관련된 모든 오류를 처리하는 데 사용됩니다:

- MQRC 2393 - MQRC_SSL_초기화_오류
- MQRC 2381 - MQRC_SSL_KEYREPOSITORY_ERROR
- MQRC 2392 - MQRC_SSL_CONFIG_ERROR

자세한 내용은 XMS의 [.NET 클라이언트 라이브러리에서 발생하는 일반적인 SSL 오류 코드](#)를 참조하십시오.

다음과 같은 시나리오에서 예외가 반환됩니다:

- 암호화 키 파일 경로에 암호화 키가 포함된 파일이 존재하지 않으면, 예외(`FileNotFoundException`)가 반환됩니다.
- 암호화 키가 포함된 파일에 null 값이 존재하는 경우, `ArgumentNullException`가 반환됩니다.

유틸리티에 암호화 키 속성을 제공하지 않을 경우, 암호화를 수행하기 위해 미리 정의된 기본 키가 대신 사용됩니다. 이 기본 키는 내부적으로 정의되며 128비트 또는 256비트를 사용합니다. 오류가 발생하지 않습니다.

XMS 메시지

이 절에서는 XMS 메시지의 구조 및 콘텐츠, 애플리케이션이 XMS 메시지를 처리하는 방법을 설명합니다.

이 절에는 다음 주제가 포함되어 있습니다.

- [633 페이지의 『XMS 메시지 부분』](#)
- [633 페이지의 『XMS 메시지의 헤더 필드』](#)
- [634 페이지의 『XMS 메시지의 특성』](#)
- [636 페이지의 『XMS 메시지의 본문』](#)

- 639 페이지의 『메시지 선택자』
- 640 페이지의 『XMS 메시지를 IBM MQ 메시지로 맵핑』

XMS 메시지 부분

XMS 메시지는 헤더, 특성 세트 및 본문으로 구성됩니다.

헤더

메시지의 헤더에는 필드가 있으며 모든 메시지에는 동일한 세트의 헤더 필드가 있습니다. XMS 및 애플리케이션은 헤더 필드 값을 사용하여 메시지를 식별하고 라우팅합니다. 헤더 필드에 대한 자세한 정보는 633 페이지의 『XMS 메시지의 헤더 필드』의 내용을 참조하십시오.

특성 세트

메시지 특성은 메시지에 대한 추가 정보를 지정합니다. 모든 메시지에 동일한 헤더 필드 세트가 있긴 하지만 모든 메시지의 특성 세트는 서로 다를 수 있습니다. 자세한 정보는 634 페이지의 『XMS 메시지의 특성』의 내용을 참조하십시오.

본문

메시지 본문에는 애플리케이션 데이터가 들어 있습니다. 자세한 정보는 636 페이지의 『XMS 메시지의 본문』의 내용을 참조하십시오.

애플리케이션에서 수신할 메시지를 선택할 수 있습니다. 선택 기준을 지정하는 메시지 선택자를 사용합니다. 이 기준은 특정 헤더 필드 값 및 메시지의 특성 값에 기반할 수 있습니다. 메시지 선택자에 대한 자세한 정보는 639 페이지의 『메시지 선택자』의 내용을 참조하십시오.

XMS 메시지의 헤더 필드

XMS 애플리케이션이 WebSphere JMS 애플리케이션과 메시지를 교환할 수 있도록 하기 위해 XMS 메시지 헤더에는 JMS 메시지 헤더 필드가 있습니다.

이러한 헤더 필드의 이름은 JMS 접두부로 시작됩니다. JMS 메시지 헤더 필드에 대한 설명은 *Java Message Service* 스펙을 참조하십시오.

XMS에서는 Message 오브젝트의 속성으로서 JMS 메시지 헤더 필드를 구현합니다. 각 헤더 필드에는 해당 값을 설정하고 가져오는 데 필요한 자체 메소드가 있습니다. 이러한 메소드에 대한 설명은 *IMessage*를 참조하십시오. 헤더 필드는 항상 읽기/쓰기가 가능합니다.

633 페이지의 표 99에서는 JMS 메시지 헤더 필드를 나열하고 전송된 메시지에 대해 각 필드 값이 설정되는 방법을 표시합니다. 필드 중 일부는 애플리케이션이 메시지를 전송할 때 또는 JMSRedelivered의 경우 애플리케이션이 메시지를 수신할 때 XMS에 의해 자동으로 설정됩니다.

표 99. JMS 메시지 헤더 필드.]	
JMS 메시지 헤더 필드 이름	전송된 메시지에 값을 설정하는 방법(<i>method [class]</i> 형식)
JMSCorrelationID	Set JMSCorrelationID [Message]
JMSDeliveryMode	Send [MessageProducer]
JMSDestination	Send [MessageProducer]
JMSExpiration	Send [MessageProducer]
JMSMessageID	Send [MessageProducer]
JMSPriority	Send [MessageProducer]
JMSRedelivered	Receive [MessageConsumer]
JMSReplyTo	Set JMSReplyTo [Message]
JMSTimestamp	Send [MessageProducer]
JMSType	Set JMSType [Message]

XMS 메시지의 특성

XMS에서는 세 가지 종류의 메시지 특성 즉, JMS 정의 특성, IBM 정의 특성 및 애플리케이션 정의 특성을 지원합니다.

XMS 는 Message 오브젝트의 다음 사전 정의된 특성을 지원하므로 XMS 애플리케이션은 WebSphere JMS 애플리케이션과 메시지를 교환할 수 있습니다.

- WebSphere JMS 에서 지원하는 동일한 JMS정의 특성입니다. 이러한 특성의 이름은 JMSX 접두부로 시작합니다.
- WebSphere JMS 에서 지원하는 동일한 IBM정의 특성입니다. 이러한 특성의 이름은 JMS_IBM_ 접두부로 시작합니다.

각 사전 정의된 특성의 이름은 다음과 같이 두 개입니다.

- JMS 이름(JMS 정의 특성의 경우) 또는 WebSphere JMS 이름(IBM 정의 특성의 경우).

이는 JMS 또는 WebSphere JMS에 특성이 알려진 이름이며 이 특성을 가진 메시지와 함께 제공되는 이름이기도 합니다. XMS 애플리케이션은 이 이름을 사용하여 메시지 선택자 표현식에서 특성을 식별합니다.

- 메시지 선택자 표현식을 제외한 모든 상황에서 특성을 식별하는 XMS 이름. 각 XMS 이름은 IBM.XMS.XMSC 클래스에서 이름 지정된 상수로서 정의됩니다. 이름 지정된 상수 값은 해당하는 JMS 또는 WebSphere JMS 이름입니다.

사전 정의된 특성 외에도 XMS 애플리케이션은 자체 메시지 특성 세트를 작성하고 사용할 수 있습니다. 이러한 특성을 애플리케이션 정의 특성이라고 합니다.

애플리케이션이 메시지를 작성한 후에 메시지의 특성을 읽고 쓸 수 있습니다. 애플리케이션이 메시지를 전송한 후에도 특성은 계속 읽고 쓸 수 있습니다. 애플리케이션이 메시지를 수신할 경우 메시지 특성은 읽기 전용입니다. 메시지의 특성이 읽기 전용일 때 애플리케이션이 메시지 클래스의 Clear Properties 메소드를 호출하는 경우, 특성은 읽기 및 쓰기가 가능하게 됩니다. 또한 이 메소드가 특성도 지웁니다.

메시지 특성을 제거한 후 전달될 때 수신되는 메시지는 메시지 특성이 제거된 표준 .NET용 WMQ XMS BytesMessage를 전달하는 것과 동일한 방식으로 작동합니다.

그러나 이는 다음 특성이 유실되므로 권장하지 않습니다.

- JMS_IBM_Encoding 특성 값 - 메시지 데이터가 의미 있게 디코딩될 수 없음을 의미합니다.
- JMS_IBM_Format 특성 값 - (MQMD 또는 새 MQRFH2) 메시지 헤더와 기존 헤더 사이의 헤더 체인이 끊어짐을 의미합니다.

메시지의 모든 특성 값을 판별하기 위해 애플리케이션이 Message 클래스의 Get Properties 메소드를 호출할 수 있습니다. 이 메소드는 Property 오브젝트의 목록을 캡슐화하는 반복기를 작성합니다. 각 Property 오브젝트는 메시지의 특성을 나타냅니다. 그런 다음 애플리케이션은 Iterator 클래스 메소드를 사용하여 각 Property 오브젝트를 차례로 검색할 수 있고 Property 클래스 메소드를 사용하여 각 특성의 이름, 데이터 유형 및 값을 검색할 수 있습니다.

메시지의 JMS 정의 특성

여러 JMS 정의 메시지 특성은 XMS 및 WebSphere JMS 모두에서 지원됩니다.

634 페이지의 표 100에서는 XMS 및 WebSphere JMS 모두에서 지원되는 JMS 정의 메시지 특성을 나열합니다. JMS 정의 특성에 대한 설명은 *Java Message Service* 스펙을 참조하십시오. JMS 정의 특성은 브로커에 대한 실시간 연결의 경우 올바르지 않습니다.

이 표는 각 특성의 데이터 유형을 지정하고 전송되는 메시지에 대해 특성 값을 설정하는 방법을 나타냅니다. 특성 중 일부는 애플리케이션이 메시지를 전송할 때 또는 JMSXDeliveryCount의 경우 애플리케이션이 메시지를 수신할 때 XMS에 의해 자동으로 설정됩니다.

표 100. 메시지의 JMS 정의 특성			
JMS 정의 특성의 XMS 이름	JMS 이름	데이터 유형	전송된 메시지에 값을 설정하는 방법(method [class] 형식)
JMSX_APPID	JMSXAppID	System.String	Send [MessageProducer]

표 100. 메시지의 JMS 정의 특성 (계속)			
JMS 정의 특성의 XMS 이름	JMS 이름	데이터 유형	전송된 메시지에 값을 설정하는 방법(method [class] 형식)
JMSX_DELIVERY_COUNT	JMSXDeliveryCount	System.Int32	Receive [MessageConsumer]
JMSX_GROUPID	JMSXGroupID	System.String	Set String Property [PropertyContext]
JMSX_GROUPSEQ	JMSXGroupSeq	System.Int32	Set Integer Property [PropertyContext]
JMSX_USERID	JMSXUserID	System.String	Send [MessageProducer]

메시지의 IBM 정의 특성

메시지에 대한 여러 IBM 정의 특성은 XMS 및 WebSphere JMS에서 지원됩니다.

635 페이지의 표 101에는 XMS 및 WebSphere JMS모두에서 지원되는 메시지의 IBM 정의 특성이 나열되어 있습니다. IBM 정의 특성에 대한 자세한 정보는 IBM MQ 또는 WebSphere Application Server 제품 문서를 참조하십시오.

이 표는 각 특성의 데이터 유형을 지정하고 전송되는 메시지에 대해 특성 값을 설정하는 방법을 나타냅니다. 특성 중 일부는 애플리케이션이 메시지를 전송할 때 XMS에 의해 자동으로 설정됩니다.

표 101. IBM 정의 메시지 특성			
IBM 정의 특성의 XMS 이름	WebSphere JMS 이름	데이터 유형	전송된 메시지에 값을 설정하는 방법(method [class] 형식)
JMS_IBM_CHARACTER_SET	JMS_IBM_Character_Set	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_ENCODING	JMS_IBM_Encoding	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_EXCEPTIONMESSAGE	JMS_IBM_ExceptionMessage	System.String	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONREASON	JMS_IBM_ExceptionReason	System.Int32	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONTIMESTAMP	JMS_IBM_ExceptionTimestamp	System.Int64	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONPROBLEMDESTINATION	JMS_IBM_ExceptionProblemDestination	System.String	Receive [MessageConsumer]
JMS_IBM_FEEDBACK	JMS_IBM_Feedback	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_FORMAT	JMS_IBM_Format	System.String	Set String Property [PropertyContext]
JMS_IBM_LASTMSGINGROUP	JMS_IBM_Last_Msg_In_Group	System.Boolean	Set Integer Property [PropertyContext]
JMS_IBM_MSGTYPE	JMS_IBM_MsgType	System.Int32	Set Integer Property [PropertyContext]

표 101. IBM 정의 메시지 특성 (계속)			
IBM 정의 특성의 XMS 이름	WebSphere JMS 이름	데이터 유형	전송된 메시지에 값을 설정하는 방법(method [class] 형식)
JMS_IBM_PUTAPPLTYPE	JMS_IBM_PutApplType	System.Int32	Send [MessageProducer]
JMS_IBM_PUTDATE	JMS_IBM_PutDate	System.String	Send [MessageProducer]
JMS_IBM_PUTTIME	JMS_IBM_PutTime	System.String	Send [MessageProducer]
JMS_IBM_REPORT_COA	JMS_IBM_Report_COA	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_COD	JMS_IBM_Report_COD	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_DISCARD_MSG	JMS_IBM_Report_Discard_Msg	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_EXCEPTION	JMS_IBM_Report_Exception	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_EXPIRATION	JMS_IBM_Report_Expiration	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_NAN	JMS_IBM_Report_NAN	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_PAN	JMS_IBM_Report_PAN	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_PASS_CORREL_ID	JMS_IBM_Report_Pass_Correl_ID	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_PASS_MSG_ID	JMS_IBM_Report_Pass_Msg_ID	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_SYSTEM_MESSAGEID	JMS_IBM_System_MessageID	System.String	Send [MessageProducer]

애플리케이션 정의 메시지 특성

XMS 애플리케이션은 자체 메시지 특성 세트를 작성하여 사용할 수 있습니다. 애플리케이션이 메시지를 전송할 때 해당 특성도 메시지와 함께 전송됩니다. 수신 애플리케이션은 메시지 선택자를 사용하여 이러한 특성 값에 기반하여 수신할 메시지를 선택할 수 있습니다.

WebSphere JMS 애플리케이션이 XMS 애플리케이션이 전송한 메시지를 선택하여 처리할 수 있도록 하려면 애플리케이션 정의 특성의 이름이 메시지 선택자 표현식의 ID 구성 규칙을 준수해야 합니다. 자세한 정보는 140 페이지의 『JMS의 메시지 선택자』의 내용을 참조하십시오. 애플리케이션 정의 특성 값의 데이터 유형은 System.Boolean, System.SByte, System.Int16, System.Int32, System.Int64, System.Float, System.Double 또는 System.String 데이터 유형 중 하나여야 합니다.

XMS 메시지의 본문

메시지 본문에는 애플리케이션 데이터가 들어 있습니다. 그러나 메시지에는 본문이 포함될 수 없으며 헤더 필드 및 특성만으로 구성됩니다.

XMS는 다섯 가지 유형의 메시지 본문을 지원합니다.

바이트

본문에 바이트 스트림이 포함됩니다. 이 유형의 본문이 있는 메시지를 바이트 메시지라고 합니다. `IBytesMessage` 인터페이스에는 바이트 메시지의 본문을 처리하는 메소드가 포함되어 있습니다.

맵

본문은 이름-값 쌍 세트를 포함합니다. 이 때 각 값은 연관된 데이터 유형을 갖습니다. 이 유형의 본문이 있는 메시지를 맵 메시지라고 합니다. `IMapMessage` 인터페이스에는 맵 메시지 본문을 처리하는 메소드가 포함되어 있습니다.

오브젝트

본문에는 직렬화된 Java 또는 .NET 오브젝트가 포함되어 있습니다. 이 유형의 본문이 있는 메시지를 오브젝트 메시지라고 합니다. `IOBJECTMessage` 인터페이스에는 오브젝트 메시지의 본문을 처리하는 메소드가 있습니다.

스트림

본문은 값 스트림을 포함합니다. 이 때 각 값은 연관된 데이터 유형을 갖습니다. 이 유형의 본문이 있는 메시지를 스트림 메시지라고 합니다. `IStreamMessage` 인터페이스에는 스트림 메시지의 본문을 처리하는 메소드가 포함되어 있습니다.

텍스트

본문에 문자열이 포함됩니다. 이 유형의 본문이 있는 메시지를 텍스트 메시지라고 합니다. `ITextMessage` 인터페이스에는 텍스트 메시지의 본문을 처리하는 메소드가 포함되어 있습니다.

`IMessage` 인터페이스는 모든 메시지 오브젝트의 상위이며 메시징 기능에서 XMS 메시지 유형을 나타내는 데 사용될 수 있습니다.

이러한 데이터 유형 각각의 크기, 최대값 및 최소값에 대한 정보는 [615 페이지의 표 87](#)의 내용을 참조하십시오.

바이트 메시지

바이트 메시지의 본문은 바이트 스트림을 포함합니다. 본문에는 실제 데이터만 있으며 이 데이터를 해석하는 애플리케이션을 전송하고 수신할 책임이 있습니다.

XMS 애플리케이션이 XMS 또는 JMS API(Application Programming Interface)를 사용하지 않는 애플리케이션과 메시지를 교환해야 하는 경우 바이트 메시지가 유용합니다.

애플리케이션이 바이트 메시지를 작성한 후에는 메시지 본문이 쓰기 전용이 됩니다. 애플리케이션은 .NET에 대해 `IBytesMessage` 인터페이스의 해당 쓰기 메소드를 호출하여 애플리케이션 데이터를 본문에 어셈블합니다. 애플리케이션이 바이트 메시지 스트림에 값을 쓸 때마다 그 값이 애플리케이션에 의해 작성된 이전 값 바로 뒤에 어셈블됩니다. XMS는 내부 커서를 유지보수하여 어셈블된 마지막 바이트의 위치를 기억합니다.

애플리케이션이 메시지를 전송할 때 메시지의 본문은 읽기 전용이 됩니다. 이 모드에서는 애플리케이션이 반복적으로 메시지를 전송할 수 있습니다.

애플리케이션이 바이트 메시지를 수신할 때 메시지 본문은 읽기 전용입니다. 애플리케이션은 `IBytesMessage` 인터페이스의 해당 읽기 메소드를 사용하여 바이트 메시지 스트림의 콘텐츠를 읽을 수 있습니다. 애플리케이션은 바이트를 순서대로 읽고 XMS는 내부 커서를 유지보수하여 읽은 마지막 바이트의 위치를 기억합니다.

바이트 메시지 본문이 쓰기 가능한 경우 애플리케이션이 `IBytesMessage` 인터페이스의 `Reset` 메소드를 호출하면 본문은 읽기 전용이 됩니다. 또한 메소드가 커서를 바이트 메시지 스트림의 시작으로 옮깁니다.

바이트 메시지의 본문이 읽기 전용 상태일 때 .NET 애플리케이션이 `IMessage` 인터페이스의 `Clear Body` 메소드를 호출하면, 해당 본문이 쓰기 가능 상태로 변경됩니다. 또한 메소드가 본문을 지웁니다.

맵 메시지

맵 메시지의 본문에는 이름-값 쌍 세트가 포함됩니다. 이 때 각 값에는 연관된 데이터 유형이 있습니다.

각 이름-값 쌍에서 이름은 값을 식별하는 문자열이며, 값은 에 나열된 데이터 XMS 유형 중 하나를 가지는 [639 페이지의 표 102](#) 애플리케이션 데이터의 요소입니다. 이름-값 쌍의 순서는 정의되지 않습니다. `MapMessage` 클래스에는 이름-값 쌍을 설정하고 가져오는 메소드가 포함됩니다.

애플리케이션은 이름을 지정하여 이름-값 쌍에 임의로 액세스할 수 있습니다.

.NET 애플리케이션은 `MapNames` 특성을 사용하여 맵 메시지 본문에 있는 이름 나열을 가져올 수 있습니다.

애플리케이션이 이름-값 쌍의 값을 가져올 때 값은 XMS에 의해 다른 데이터 유형으로 변환될 수 있습니다. 예를 들어 맵 메시지 본문에서 정수를 가져오기 위해 애플리케이션이 `MapMessage` 클래스의 `GetString` 메소드를 호출할 수 있습니다. 이는 정수를 문자열로서 리턴합니다. 지원되는 변환은 XMS가 특성 값을 한 데이터 유형에서

다른 유형으로 변환할 때 지원되는 것과 동일합니다. 지원되는 변환에 대한 자세한 정보는 [615 페이지의 『한 데이터 유형에서 다른 데이터 유형으로 특성 값의 암시적 변환』](#)의 내용을 참조하십시오.

애플리케이션이 맵 메시지를 작성한 후에는 메시지 본문이 읽기 및 쓰기가 가능하게 됩니다. 애플리케이션이 메시지를 전송한 후에도 본문은 읽기 및 쓰기가 가능합니다. 애플리케이션이 맵 메시지를 수신할 때 메시지 본문은 읽기 전용입니다. 맵 메시지 본문이 읽기 전용일 때 애플리케이션이 `Message` 클래스의 `Clear Body` 메소드를 호출하면 본문에서 읽기 및 쓰기가 가능하게 됩니다. 또한 메소드가 본문을 지웁니다.

오브젝트 메시지

오브젝트 메시지 본문에는 직렬화된 Java 또는 .NET 오브젝트가 포함되어 있습니다.

XMS 애플리케이션은 오브젝트 메시지를 수신하고, 이의 헤더 필드 및 특성을 변경한 후 이를 다른 목적지로 전송할 수 있습니다. 또한 애플리케이션은 오브젝트 메시지의 본문을 복사하여 다른 오브젝트 메시지를 구성하는 데 사용할 수 있습니다. XMS는 오브젝트 메시지의 본문을 바이트 배열로 취급합니다.

애플리케이션이 오브젝트 메시지를 작성한 후에는 메시지 본문이 읽기 및 쓰기가 가능하게 됩니다. 애플리케이션이 메시지를 전송한 후에도 본문은 읽기 및 쓰기가 가능합니다. 애플리케이션이 오브젝트 메시지를 수신할 때 메시지 본문은 읽기 전용입니다. 객체 메시지의 본문이 읽기 전용일 때 .NET 애플리케이션이 `IMessage` 인터페이스의 `Clear Body` 메서드를 호출하면, 본문이 읽기 및 쓰기 가능 상태가 됩니다. 또한 메소드가 본문을 지웁니다.

스트림 메시지

스트림 메시지의 본문은 값 스트림을 포함합니다. 이 때 각 값은 연관된 데이터 유형을 갖습니다.

값의 데이터 유형은 예 나열된 데이터 XMS 유형 [639 페이지의 표 102](#) 중 하나입니다.

애플리케이션이 스트림 메시지를 작성한 후 메시지 본문은 쓰기 가능합니다. 애플리케이션은 .NET `IStreamMessage` 인터페이스의 해당 쓰기 메소드를 호출하여 애플리케이션 데이터를 본문에 어셈블합니다. 애플리케이션이 메시지 스트림에 값을 쓸 때마다 그 값과 해당 유형은 애플리케이션에 의해 작성된 이전 값 바로 뒤에 어셈블됩니다. XMS는 내부 커서를 유지보수하여 어셈블된 마지막 값의 위치를 기억합니다.

애플리케이션이 메시지를 전송할 때 메시지의 본문은 읽기 전용이 됩니다. 애플리케이션은 이 모드에서 메시지를 여러 번 전송할 수 있습니다.

애플리케이션이 스트림 메시지를 수신할 때 메시지 본문은 읽기 전용입니다. 애플리케이션은 .NET `IStreamMessage` 인터페이스의 해당 읽기 메소드를 사용하여 메시지 스트림의 콘텐츠를 읽을 수 있습니다. 애플리케이션은 값을 순서대로 읽고 XMS는 내부 커서를 유지보수하여 읽은 마지막 값의 위치를 기억합니다.

애플리케이션이 메시지 스트림에서 값을 읽을 때 값은 XMS에 의해 다른 데이터 유형으로 변환될 수 있습니다. 예를 들어 메시지 스트림에서 정수를 읽으려면 애플리케이션은 문자열로 정수를 리턴하는 `ReadString` 메소드를 호출할 수 있습니다. 지원되는 변환은 XMS가 특성 값을 한 데이터 유형에서 다른 유형으로 변환할 때 지원되는 것과 동일합니다. 지원되는 변환에 대한 자세한 정보는 [615 페이지의 『한 데이터 유형에서 다른 데이터 유형으로 특성 값의 암시적 변환』](#)의 내용을 참조하십시오.

애플리케이션이 메시지 스트림에서 값을 읽으려고 시도할 때 오류가 발생하면 커서가 진행되지 않습니다. 애플리케이션은 값을 다른 데이터 유형으로 읽으려고 시도하여 오류를 복구할 수 있습니다.

스트림 메시지 본문이 쓰기 전용인 경우 애플리케이션이 XMS `IStreamMessage` 인터페이스의 `Reset` 메소드를 호출하면 본문은 읽기 전용이 됩니다. 또한 메소드가 커서를 메시지 스트림의 시작으로 옮깁니다.

스트림 메시지의 본문이 읽기 전용일 때 XMS 애플리케이션이 `IMessage` 인터페이스의 `Clear Body` 메서드를 호출하면, 본문이 쓰기 전용으로 변경됩니다. 또한 메소드가 본문을 지웁니다.

텍스트 메시지

텍스트 메시지의 본문은 문자열을 포함합니다.

애플리케이션이 텍스트 메시지를 작성한 후에는 메시지 본문이 읽기 및 쓰기가 가능하게 됩니다. 애플리케이션이 메시지를 전송한 후에도 본문은 읽기 및 쓰기가 가능합니다. 애플리케이션이 텍스트 메시지를 수신할 때 메시지 본문은 읽기 전용입니다. 텍스트 메시지 본문이 읽기 전용인 경우 애플리케이션이 .NET `IMessage` 인터페이스의 `Clear Body` 메소드를 호출하면 본문은 읽기 가능 및 쓰기 가능이 됩니다. 또한 메소드가 본문을 지웁니다.

애플리케이션 데이터 요소의 데이터 유형

XMS 애플리케이션이 IBM MQ classes for JMS 애플리케이션과 메시지를 교환할 수 있도록 하려면 두 애플리케이션이 메시지 본문에 있는 애플리케이션 데이터를 동일한 방식으로 해석할 수 있어야 합니다.

이러한 이유로 XMS 애플리케이션이 메시지 본문에 기록한 애플리케이션 데이터 요소의 데이터 유형은 639 페이지의 표 102에 나열된 데이터 유형 중 하나여야 합니다. 각 데이터 유형에 대해 표에서는 호환 가능한 Java 데이터 유형을 표시합니다. XMS는 이러한 데이터 유형만으로 애플리케이션 데이터 요소를 작성할 수 있는 메소드를 제공합니다.

표 102. XMS 호환 가능한 Java 데이터 유형		
XMS 데이터 유형	의미	호환 가능 Java 데이터 유형
System.Boolean	Boolean 값 True 또는 False	부울
System.Char16	2바이트 문자	char
System.SByte	부호가 있는 8비트 정수	byte
System.Int16	부호가 있는 16비트 정수	short
System.Int32	부호 있는 32비트 정수	int
System.Int64	부호가 있는 64비트 정수	long
System.Float	부호가 있는 부동 소수점 수	float
System.Double	부호가 있는 배정밀도 부동 소수점 수	double
System.String	문자열	문자열

이러한 데이터 유형 각각의 크기, 최대값 및 최소값에 대한 정보는 614 페이지의 『XMS 기본 유형』의 내용을 참조하십시오.

메시지 선택자

XMS 애플리케이션은 메시지 선택자를 사용하여 수신하려는 메시지를 선택합니다.

애플리케이션이 메시지 이용자를 작성할 때 메시지 선택자 표현식과 이용자를 연관시킬 수 있습니다. 메시지 선택자 표현식이 선택 기준을 지정합니다.

애플리케이션이 IBM WebSphere MQ 7.0 큐 관리자에 연결되면 메시지 선택은 큐 관리자측에서 수행됩니다. XMS는 아무 것도 선택하지 않고 단순히 큐 관리자로부터 받은 메시지를 전달만 하므로 성능이 개선됩니다.

애플리케이션이 둘 이상의 메시지 이용자를 작성할 수 있고, 각각에는 자체 메시지 선택자 표현식이 있습니다. 수신 메시지가 두 개 이상의 메시지 이용자 선택 기준을 충족시킬 경우 XMS는 해당 이용자 각각에게 메시지를 전달합니다.

메시지 선택자 표현식은 다음과 같은 메시지 특성을 참조할 수 있습니다.

- JMS 정의 특성
- IBM 정의 특성
- 애플리케이션 정의 특성

또한 다음과 같은 메시지 헤더 필드도 참조할 수 있습니다.

- JMSCorrelationID
- JMSDeliveryMode
- JMSMessageID
- JMSPriority
- JMSTimestamp
- JMSType

그러나 메시지 선택자 표현식이 메시지 본문의 데이터는 참조할 수 없습니다.

다음은 메시지 선택자 표현식의 한 예입니다.

```
JMSPriority > 3 AND manufacturer = 'Jaguar' AND model in ('xj6','xj12')
```

XMS 메시지의 우선 순위가 '3'보다 큰 경우에만 이 메시지 선택자 표현식을 사용하여 메시지 소비자에게 메시지를 전달하고, 'Jaguar'; '값'을 가진 애플리케이션 정의 속성인 제조업체와 'xj6' 또는 'xj12.' 값을 가진 다른 애플리케이션 정의 속성인 모델을 사용합니다

XMS의 메시지 선택자 표현식 형성 구문 규칙은 IBM MQ classes for JMS의 구문 규칙과 동일합니다. 메시지 선택자 표현식을 구성하는 방법에 대한 정보는 IBM MQ 제품 문서를 참조하십시오. 메시지 선택자 표현식에서는 JMS 정의 특성의 이름이 JMS 이름이고 IBM 정의 특성의 이름은 IBM MQ classes for JMS 이름이어야 함에 유의하십시오. 메시지 선택자 표현식에서 XMS 이름을 사용할 수 없습니다.

XMS 메시지를 IBM MQ 메시지로 맵핑

XMS 메시지의 JMS 헤더 필드 및 특성은 IBM MQ 메시지의 헤더 구조에 있는 필드로 맵핑됩니다.

XMS 애플리케이션이 IBM MQ 큐 관리자에 연결되면 큐 관리자에 송신된 메시지는 유사한 상황에서 IBM MQ classes for JMS 메시지가 IBM MQ 메시지에 맵핑되는 것과 동일한 방식으로 IBM MQ 메시지에 맵핑됩니다.

Destination 오브젝트의 `XMSC_WMQ_TARGET_CLIENT` 특성이 `XMSC_WMQ_TARGET_DEST_JMS`로 설정되는 경우 목적지로 전송된 메시지의 특성 및 JMS 헤더 필드는 IBM MQ 메시지의 MQMD 및 MQRFH2 헤더 구조에 있는 필드로 맵핑됩니다. 이런 방법으로 `XMSC_WMQ_TARGET_CLIENT` 특성을 설정할 경우 메시지를 수신하는 애플리케이션이 MQRFH2 헤더를 처리할 수 있다고 가정합니다. 그러므로 수신 애플리케이션은 다른 XMS 애플리케이션, IBM MQ classes for JMS 애플리케이션 또는 MQRFH2 헤더를 처리하도록 디자인된 고유 IBM MQ 애플리케이션이 될 수 있습니다.

Destination 오브젝트의 `XMSC_WMQ_TARGET_CLIENT` 특성이 대신 `XMSC_WMQ_TARGET_DEST_MQ`로 설정되는 경우에는 목적지로 전송된 메시지의 특성 및 JMS 헤더 필드가 IBM MQ 메시지의 MQMD 헤더 구조에 있는 필드로 맵핑됩니다. 이 메시지에는 MQRFH2 헤더가 없으며, MQMD 헤더 구조의 필드로 맵핑할 수 없는 모든 JMS 헤더 필드 및 특성은 무시됩니다. 그러므로 메시지를 수신하는 애플리케이션은 MQRFH2 헤더를 처리하도록 디자인되지 않은 고유 IBM MQ일 수 있습니다.

큐 관리자로부터 수신된 IBM MQ 메시지는 유사한 상황에서 IBM MQ 메시지가 IBM MQ classes for JMS 메시지에 맵핑되는 것과 동일한 방식으로 XMS 메시지에 맵핑됩니다.

수신 IBM MQ 메시지에 MQRFH2 헤더가 있는 경우, 결과 XMS 메시지에는 MQRFH2 헤더의 mcd 폴더에 포함된 **Msd** 특성의 값에 의해 유형이 판별되는 본문이 있습니다. **Msd** 특성이 MQRFH2 헤더에 없거나 IBM MQ 메시지에 MQRFH2 헤더에 없는 경우, 결과적으로 XMS 메시지에는 MQMD 헤더의 *Format* 필드 값으로 유형이 판별되는 본문이 있습니다. *Format* 필드가 MQFMT_STRING으로 설정된 경우 XMS 메시지는 텍스트 메시지입니다. 그렇지 않은 경우에는 XMS 메시지가 바이트 메시지입니다. IBM MQ 메시지에 MQRFH2 헤더가 없을 경우 MQMD 헤더의 필드에서 파생할 수 있는 JMS 헤더 필드 및 특성만 설정됩니다.

IBM MQ classes for JMS 메시지를 IBM MQ 메시지로 맵핑에 대한 자세한 정보는 [145 페이지의 『JMS 메시지를 IBM MQ 메시지로 맵핑』](#)의 내용을 참조하십시오.

IBM MQ Message Service Client (XMS) for .NET 애플리케이션에서 메시지 디스크립터 읽기 및 쓰기

StrucId와 Version을 제외한 IBM MQ 메시지의 모든 메시지 디스크립터(MQMD) 필드에 액세스할 수 있습니다. BackoutCount는 읽을 수 있지만 쓸 수는 없습니다.

IBM MQ Message Service Client (XMS) for .NET에서 제공하는 메시지 속성을 사용하면 XMS 애플리케이션이 MQMD 필드를 설정하고 IBM WebSphere MQ 애플리케이션을 구동하기도 쉬워집니다.

발행/구독 메시지를 사용할 경우 일부 제한사항이 적용됩니다. 예를 들어 MsgID 및 CorrelId와 같은 MQMD 필드는 무시됩니다.

해당 **PROVIDERVERSION** 속성이 6으로 설정된 경우에도 이 기능은 사용할 수 없습니다.

IBM MQ Message Service Client (XMS) for .NET 애플리케이션에서 IBM MQ 메시지 데이터 액세스

애플리케이션 IBM MQ Message Service Client (XMS) for .NET 내에서 MQRFH2 헤더(존재하는 경우) 및 기타 모든 IBM MQ 헤더(존재하는 경우)를 포함한 전체 IBM MQ 메시지 데이터에 JMSBytesMessage의 본문으로 접근할 수 있습니다.

이 주제에서 설명하는 기능은 IBM WebSphere MQ 7.0 이상의 큐 관리자에 연결하며 IBM MQ 메시징 제공자가 정상 모드인 경우에만 사용할 수 있습니다.

Destination 오브젝트 특성은 XMS 애플리케이션이 JMSBytesMessage의 본문으로서 IBM MQ 메시지 전체(있는 경우 MQRFH2 헤더 포함)에 액세스하는 방법을 판별합니다.

ALW

AMQP 클라이언트 애플리케이션 개발

AMQP API에 대한 IBM MQ 지원을 통해 IBM MQ 관리자가 AMQP 채널을 작성할 수 있습니다. 시작되면 이 채널은 AMQP 클라이언트 애플리케이션에서 연결을 승인하는 포트 번호를 정의합니다.

AMQP 채널은 시스템에 AIX, Linux, and Windows 설치할 수 있습니다. 또는 IBM i에서는 z/OS 사용할 수 없습니다.

AMQP 1.0 클라이언트 애플리케이션은 AMQP 채널을 사용하여 큐 관리자에 연결할 수 있습니다.

Apache Qpid JMS 라이브러리를 사용하는 애플리케이션 개발

소개

Apache Qpid JMS 라이브러리에서는 AMQP 1.0 프로토콜을 사용하여 JMS 2 스펙 구현을 제공합니다.

Apache Qpid JMS에서는 MQ Light 메시징 API와 다른 방식으로 AMQP 1.0 프로토콜의 몇몇 측면을 사용합니다. IBM MQ 9.2 AMQP IBM MQ 채널에 대한 지원을 추가하여 애플리케이션이 연결하고 IBM MQ 게시/구독 Apache QpidJMS 메시징을 수행할 수 있게 되었습니다. 여기에는 공유 구독 사용도 포함됩니다.

IBM MQ 9.3 AMQP IBM MQ 채널에 대한 추가 지원을 제공하여 애플리케이션이 연결하고 IBM MQ 지점 간 Apache QpidJMS 메시징을 수행할 수 있도록 합니다. [645 페이지의 『AMQP 채널에서의 포인트-투-포인트 지원』](#)의 내용을 참조하십시오.

IBM MQ 9.3.0 AMQP IBM MQ 채널에 대한 추가적인 큐 브라우징 지원을 추가하여, JMS 애플리케이션이 큐에 IBM MQ 연결하고 Apache Qpid 메시지를 브라우징할 수 있도록 했습니다. [645 페이지의 『AMQP 채널에서의 포인트-투-포인트 지원』](#)의 내용을 참조하십시오.

IBM MQ 9.3.0 AMQP 채널에 [TMPMODEL](#) 및 [TMPQPRFX](#)라는 두 개의 추가 채널 속성을 추가했습니다. 이러한 속성은 모델 큐와 임시 큐를 작성하는 동안 사용할 임시 큐 접두어용입니다.

다른 IBM MQ 애플리케이션과의 상호통신

Apache Qpid JMS 애플리케이션과 다른 IBM MQ 애플리케이션 간에는 메시지를 전송할 수 있습니다. 예를 들어, Apache Qpid 애플리케이션은 토픽에 메시지를 발행할 수 있으며 MQ Light 애플리케이션은 구독을 작성하여 이를 수신할 수 있습니다.

Apache Qpid JMS 애플리케이션은 기존 IBM MQ 애플리케이션에서 이용하는 메시지를 발행할 수도 있습니다 (예: MQSUB API 호출을 사용하여 동일한 토픽을 구독).

마찬가지로, Apache Qpid JMS 애플리케이션은 기존 IBM MQ 애플리케이션이 메시지를 발행하는 IBM MQ 토픽을 구독할 수 있습니다.

Apache Qpid JMS 애플리케이션과 MQ Light 애플리케이션이 동일한 공유 이름 및 토픽 패턴을 지정하는 경우에는 두 클라이언트가 구독을 공유할 수도 있습니다.

이를 수행하려면 Apache Qpid JMS 애플리케이션이 클라이언트 ID를 사용하여 연결해서는 안 된다는 점을 참고하십시오. 이를 통해 두 애플리케이션에서 사용하는 IBM MQ 구독 이름이 동일합니다.



주의: Apache Qpid JMS 애플리케이션은 IBM MQ JMS 애플리케이션과 구독을 공유할 수 없습니다.

Apache Qpid JMS 제한사항

다음 JMS 기능이 지원됩니다.

- 클라이언트 수신확인, 자동 수신확인 및 중복 허용 수신확인 모드(DUPS_OK_ACKNOWLEDGE)
 - 신임 정보를 사용하거나 사용하지 않고 연결
 - 토픽 대상에서 이용자 작성
 - 토픽 대상에서 지속 가능한 이용자 작성
 - 토픽 대상에서 공유 이용자 작성
 - 토픽 대상에서 지속 가능한 공유 이용자 작성
 - 클라이언트 수신확인 및 자동 수신확인 모드
 - 메시지 수신확인 및 세션 수신확인
 - 지속 가능한 구독의 구독 해제
 - 임시 큐 작성
 - 큐 또는 임시 큐 목적지에서 이용자 작성
 - JMS MessageListener
 - JMS Consumer가 객체를 수신함; JMS 2.0 메서드 호출됨 `Consumer.receiveBody()`
 - 다음 JMS 메시지 유형이 지원됩니다.
 - `BytesMessage`
 - `MapMessage`
 - `ObjectMessage`
 - `StreamMessage`
 - `TextMessage`
 - 큐에서 메시지 찾아보기

다음 JMS 기능은 AMQP 클라이언트에서 지원되지 않습니다.

- 트랜잭션된 세션 및 트랜잭션된 JMSContext 사용
 - 메시지 선택자 사용
 - **noLocal** 속성 사용
 - 트랜잭션 세션 사용
 - 전달 지연 사용
 - IBM MQ 9.3.0에서, 큐에서 메시지 찾아보기
 - 클라이언트 ID 및 토픽이 동일한 여러 지속 가능 구독 또는 이용자 작성
 - JMS 임시 토픽
 - AMQP 필터는 지원되지 않습니다.

샘플 AMQP 클라이언트 다운로드

IBM MQ에서는 AMQP 클라이언트를 배송하지 않지만 MQ Light 클라이언트를 다운로드하거나 Apache Qpid 라이브러리를 기반으로 오픈 소스 AMQP 클라이언트를 다운로드할 수 있습니다. 자세한 정보는 [IBM MQ Light](#) 및 [Apache Qpid](#)의 내용을 참조하십시오.

Apache Qpid 라이브러리를 기반으로 하는 다른 오픈 소스 AMQP 클라이언트를 다운로드할 수도 있습니다. 자세한 정보는 <https://qpid.apache.org/index.html>의 내용을 참조하십시오.



주의: IBM 지원팀은 해당 클라이언트 패키지에 대한 구성 또는 결함 지원을 제공할 수 없으며, 사용 관련 문의나 코드 결함 보고는 각 프로젝트에 직접 문의해야 합니다.

IBM MQ에 AMQP 클라이언트 배치

애플리케이션의 배치 준비가 되면 이는 다른 엔터프라이즈 애플리케이션의 모든 모니터링, 안정성 및 보안 기능을 필요로 합니다. 또한 다른 엔터프라이즈 애플리케이션과 데이터를 교환할 수도 있습니다.

AMQP 클라이언트를 배치한 경우 메시지를 IBM MQ 애플리케이션과 교환할 수 있습니다. 예를 들어, AMQP 클라이언트를 사용하여 JavaScript 문자열 메시지를 전송하는 경우, IBM MQ 애플리케이션은 MQ 메시지를 수신합니다. 여기서, MQMD의 형식 필드는 MQSTR로 설정됩니다.

AMQP 채널 관리

AMQP 채널은 다른 MQ 채널과 동일한 방법으로 관리될 수 있습니다. MQSC 명령, PCF 명령 메시지 또는 IBM MQ Explorer를 사용하여 채널을 정의, 시작, 중지 및 관리할 수 있습니다. [AMQP 채널 작성 및 사용](#)에서는 클라이언트를 큐 관리자에 대한 연결을 정의 및 시작하기 위해 예제 명령이 제공됩니다.

AMQP 채널이 시작되면 AMQP 1.0 클라이언트를 연결하여 테스트할 수 있습니다. (예: MQ Light, Apache Qpid Proton 또는 Apache Qpid JMS).

관련 태스크

[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

ALW

MQ Light Apache, Qpid JMS 및 AMQP(고급 메시지 큐잉 프로토콜)

MQ Light 클라이언트, Apache Qpid 클라이언트(Apache Proton, Apache 등) 및 Qpid JMS API는 OASIS 표준 AMQP(1.0) 유선 프로토콜을 기반으로 합니다. AMQP는 송신자 및 수신자 간에 메시지가 송신되는 방법을 지정합니다. 애플리케이션이 메시지 브로커(예: IBM MQ)에 메시지를 송신하는 경우 애플리케이션은 송신자로 작동합니다. IBM MQ가 AMQP 애플리케이션에 메시지를 송신하는 경우 송신자로 작동합니다.

AMQP의 일부 이점은 다음과 같습니다.

- 표준화된 오픈 프로토콜
- 기타 오픈 소스 AMQP 1.0 클라이언트와의 호환성
- 다수의 오픈 소스 클라이언트 구현 사용 가능

AMQP 1.0 클라이언트를 AMQP 채널에 연결할 수 있지만 일부 AMQP 기능은 지원되지 않습니다(예: 트랜잭션 또는 다중 세션).

자세한 정보는 [AMQP.org](#) 웹 사이트 및 [OASIS 표준 AMQP 1.0 PDF](#)를 참조하십시오.

MQ Light 및 Apache Qpid JMS API에는 다음 메시징 기능이 있습니다.

- 최대 한 번 메시지 전달
- 최소 한 번 메시지 전달
- 토픽 문자열 목적지 주소 지정
- 메시지 및 목적지 지속성
- 다중 구독자가 워크로드를 공유할 수 있도록 하는 공유된 목적지
- 정지된 클라이언트를 쉽게 해결하도록 클라이언트 인계
- 구성 가능한 메시지 미리 읽기
- 메시지의 구성 가능한 수신확인

Apache Qpid JMS API의 전체 문서는 [Qpid JMS](#)의 내용을 참조하십시오.

관련 태스크

[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

ALW

AMQP 1.0 지원

AMQP 채널은 AMQP 1.0 준수 애플리케이션의 지원 레벨을 제공합니다.

AMQP 채널은 AMQP 1.0 프로토콜의 서브세트를 지원합니다. AMQP 1.0 호환 클라이언트를 IBM MQ AMQP 채널에 연결할 수 있습니다. AMQP 채널이 지원하는 모든 메시징 기능을 사용하려면 특정 AMQP 1.0 필드의 값을 올바르게 설정해야 합니다.

이 정보는 AMQP 필드를 형식화해야 하는 방법을 아웃라인하고 AMQP 채널이 지원하지 않는 AMQP 1.0 스펙의 기능을 나열합니다.

AMQP 1.0 스펙의 다음 기능이 지원되지 않거나 사용이 제한됩니다.

ATTACH 프레임

AMQP 채널에서는 ATTACH 프레임의 기능에 다음 중 하나가 포함됩니다.

topic
temporary queue
queue
shared

기능은 오브젝트 유형을 암시하며 다중 기능이 있는 경우 기능 선택 시 우선순위는 토픽, 임시 큐, 큐입니다.

기능에 예상 값이 포함되지 않은 경우, 기본 기능은 토픽입니다. 다른 기능은 무시됩니다.

참고: 일부 AMQP 클라이언트는 이러한 기능을 설정하지 않으며, publish/subscribe의 기본 IBM MQ 동작을 따릅니다. 예를 들어, 반응형 메시징 AMQP 1.0 Quarkus 커넥터는 버전 2.8.0CR1 부터 기능 설정을 수행합니다.

AMQP 채널은 ATTACH 프레임의 distribution-Mode이(가) 소스 또는 대상에 대해 다음 중 하나를 포함할 것으로 예상합니다.

- move
- copy

여기서 move는 파괴적 가져오기를 나타내고 copy는 브라우저를 나타냅니다.

참고: distribution-Mode이(가) 설정되지 않았거나 복사 이외의 다른 값으로 설정되면 이동으로 간주합니다.

링크 이름

AMQP 채널은 AMQP 링크의 이름이 다음 형식 중 하나를 따를 것을 기대합니다:

- 일반 토픽(발행 및 구독의 경우)
 - 메시지 발행: 일반 토픽 문자열(예: `/sports/football`)의 링크 이름을 사용하면 `/sports/football` 토픽에 메시지가 발행됩니다.
 - 메시지를 수신하기 위해 토픽 구독: 일반 토픽 문자열(예: 링크 이름을 `/sports/football`(을)로 사용하면 `/sports/football` 토픽에 구독이 정의됨).
- 개인용 상세 토픽(구독의 경우)
 - `"private:topic string"`(예: `"private:/sports/football"`) 양식의 개인 구독을 설명하는 상세 토픽 문자열입니다. 동작은 일반 토픽 문자열과 동일합니다. `private` 선언은 클라이언트 간에 공유되는 구독과 특정 AMQP 클라이언트에 고유한 구독을 구별합니다.
- 공유 상세 토픽(구독의 경우)
 - `"share:share name:topic string"`(예: `"share:bbc:/sports/football"`) 양식의 공유 구독을 설명하는 상세 토픽 문자열입니다.
- 큐(생성자와 이용자를 위한 포인트-투-포인트 메시징)
 - 메시지를 전송하는 생성자, 큐 이름 문자열을 사용하여 생성자가 큐에서 메시지를 보낼 수 있습니다.
 - 메시지를 수신하는 이용자, 큐 이름 문자열을 사용하여 이용자가 큐에서 메시지를 수신할 수 있습니다.
- 공백(임시 큐에서의 포인트-투-포인트 메시징)
 - 임시 큐에서 메시지를 전송하는 생성자, 공백을 사용하여 생성자가 임시 큐에서 메시지를 보낼 수 있습니다.

- 임시 큐에서 메시지를 수신하는 이용자, 공백을 사용하여 이용자가 임시 큐에서 메시지를 수신할 수 있습니다.

AMQP 메시지가 메시지와 IBM MQ 어떻게 매핑되는지에 대한 자세한 내용은 [을 참조하십시오 649 페이지의 『AMQP 필드를 IBM MQ 필드\(수신되는 메시지\)에 매핑』](#).

토픽 문자열, 공유 이름, 클라이언트 ID의 최대 길이

토픽 문자열, 공유 이름, 클라이언트 ID는 10237바이트 내에 포함되어야 합니다. 또한 최대 클라이언트 ID 길이는 256자입니다.

최대 길이는 다음 중 하나를 가질 수 있음을 의미합니다.

- 매우 긴 토픽 문자열(공유 이름이 짧은 경우)
- 긴 공유 이름, 짧은 토픽 문자열

컨테이너 ID

AMQP 채널은 AMQP Open 수행의 Container-id가 고유한 AMQP 클라이언트 ID를 포함할 것으로 예상합니다. AMQP 클라이언트 ID의 최대 길이는 256자이며, ID는 영숫자 문자, 퍼센트 부호(%), 슬래시(/), 마침표(.) 및 밑줄(_)을 포함할 수 있습니다.

세션

AMQP 채널은 단일 AMQP 세션만 지원합니다. 둘 이상의 AMQP 세션을 작성하려고 시도하는 AMQP 클라이언트가 오류 메시지를 수신하고 채널에서 연결이 끊어집니다.

트랜잭션

AMQP 채널은 AMQP 트랜잭션을 지원하지 않습니다. 새 트랜잭션을 통합하려고 시도하는 AMQP 접속 프레임 또는 새 트랜잭션을 선언하려고 시도하는 AMQP 전송 프레임이 오류 메시지와 함께 거부됩니다.

전달 상태

AMQP 채널은 전달 상태가 허용됨, 릴리스됨 또는 수정됨인 처리 프레임만 지원합니다. 수정됨 상태를 사용하는 경우 AMQP 채널은 현재 전달할 수 없음 옵션을 지원하지 않습니다.

관련 태스크

[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

ALW AMQP 채널에서의 포인트-투-포인트 지원

IBM MQ AMQP 채널은 큐에 메시지를 전송하고 큐로부터 메시지를 수신하는 데 대한 지원을 제공합니다.

Apache Qpid™ JMS 라이브러리와 같은 AMQP 클라이언트는 AMQP 접속 프레임을 전송할 때 queue 또는 temporary-queue 기능을 요청합니다. 기능은 AMQP 채널이 오브젝트를 큐, 임시 큐 또는 토픽으로 식별할 수 있게 해 줍니다. queue 또는 temporary-queue 기능이 없는 경우, 또는 어떠한 기능도 없는 경우 요청은 토픽에 대한 것으로 간주됩니다.

IBM MQ AMQP 채널은 다음 항목에 대해 큐 유형 지원을 제공합니다.

큐 수신 및 전송

메시지를 큐에 전송하거나 큐로부터 수신하여 이용할 수 있습니다. 메시지 이용에 대해서는 동기 및 비동기 모드가 모두 지원됩니다.

큐 메시지 찾아보기

메시지를 큐에 넣고 큐에서 메시지를 가져올 뿐만 아니라 큐에서 메시지를 찾아볼 수도 있습니다.

임시 큐 지원

메시지를 임시 큐에 전송하거나 임시 큐로부터 수신하여 이용할 수 있습니다. 임시 큐를 작성하는 데 사용된 임시 큐 오브젝트가 임시 큐를 삭제하는 데도 사용되는 경우에는 임시 큐 삭제가 지원된다는 점을 참고하십시오.

임시 큐를 작성할 때는 SYSTEM.DEFAULT.MODEL.QUEUE가 사용되며, 임시 큐에 대한 접두부는 AMQP.*입니다.

SYSTEM.DEFAULT.MODEL.QUEUE는 기본적으로 임시 동적 큐지만, SYSTEM.DEFAULT.MODEL.QUEUE 큐의 **Definition type** 특성을 사용하여 이 큐를 영구적 동적 큐로 변경할 수 있습니다.

영구적 동적 큐

Apache Qpid JMS 라이브러리와 같은 AMQP 클라이언트가 ``closed attribute`` 속성이 ``true`` 로 설정된 프레임을 detach 포함한 요청을 전송하면 영구 동적 큐가 삭제됩니다.

중요사항:

Qpid JMS 작동:

큐를 사용한 후에는 Qpid JMS API 명령(예: `javax.jms.Queue.delete()` 메소드)을 호출하여 큐를 영구 삭제해야 하며, 이 프로세스는 큐에 있는 메시지도 지웁니다.

이러한 명령을 실행하지 않으면 연결이 닫힐 때 큐가 포함된 메시지와 함께 남아있게 됩니다.

-

임시 동적 큐

임시 동적 큐는 AMQP 클라이언트가 연결을 닫을 때 삭제됩니다.

중요사항:

Qpid JMS 작동:

Qpid JMS API 명령(예: `javax.jms.Queue.delete()` 메소드)을 호출하거나 JMS 연결을 닫거나 연결이 끊어지면 큐가 삭제되고 모든 메시지가 유실됩니다.

세션을 종료하는 JMS 행위 자체로는 임시 큐가 삭제되지 않습니다. 비록 해당 임시 큐가 해당 `javax.jms.Session.createTemporaryQueue()` 메서드를 사용하여 생성되었을지라도 말입니다.

-

관련 태스크

[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

ALW AMQP 및 IBM MQ 메시지 필드 �핑

AMQP 메시지는 헤더, 전달 어노테이션, 메시지 어노테이션, 특성, 애플리케이션 특성, 본문 및 푸터로 구성됩니다.

AMQP 메시지는 다음 부분으로 구성됩니다.

헤더

선택적 헤더에는 메시지의 5가지 고정 속성이 포함됩니다.

- **durable** - 지속성 요구사항을 지정합니다.
- **priority** - 상대적인 메시지 우선순위
- **ttl** - 생존 시간(밀리초)
- **first-acquirer** - 이 값이 True인 경우 메시지가 다른 링크에 의해 확보되지 않습니다.
- **delivery-count** - 이전의 성공하지 못한 전달 시도 수입니다.

전달 어노테이션

선택사항. 다른 의도된 대상에 대해 메시지의 비표준 헤더 속성을 지정합니다. 전달 어노테이션은 송신 피어에서 수신 피어로 정보를 송신합니다.

메시지 어노테이션

선택사항. 다른 의도된 대상에 대해 메시지의 비표준 헤더 속성을 지정합니다. 메시지 어노테이션 섹션은 인프라를 목표로 하는 메시지의 특성에 대해 사용되며, 모든 전달 단계에 대해 전파되어야 합니다.

특성

선택사항. 이 부분은 MQ 메시지 디스크립터와 등가입니다. 여기에는 다음 고정된 필드가 포함됩니다.

- **message-id** - 애플리케이션 메시지 ID
- **user-id** - 작성 사용자의 ID
- **to** - 메시지가 지정된 노드의 주소
- **subject** - 메시지의 제목
- **reply-to** - 송신이 응답하는 노드
- **correlation-id** - 애플리케이션 상관 ID
- **content-type** - MIME 콘텐츠 유형
- **content-encoding** - MIME 콘텐츠 유형. 콘텐츠 유형에 대한 수정자로 사용됩니다.
- **absolute-expiry-time** - 이 메시지가 만료된 것으로 고려되는 시간
- **creation-time** - 이 메시지가 작성된 시간
- **group-id** - 이 메시지가 속한 그룹
- **group-sequence** - 해당 그룹 내에서 이 메시지의 순서 번호
- **reply-to-group-id** - 응답 메시지가 속한 그룹

애플리케이션 특성

MQ 메시지 특성과 동일합니다.

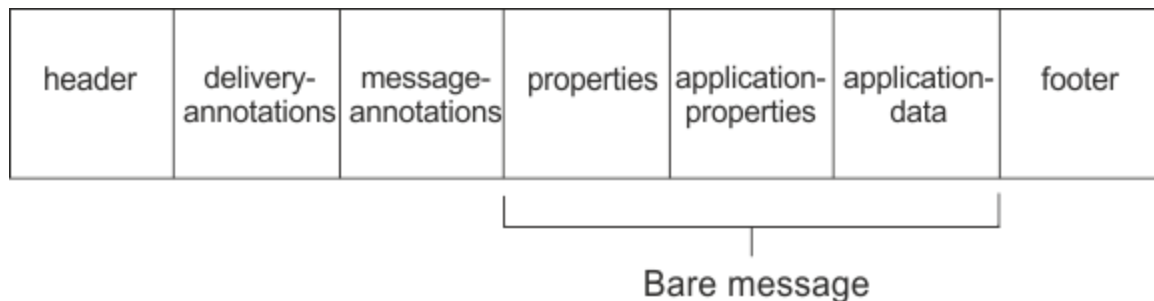
본문

MQ 사용자 페이로드와 동일합니다.

푸터

선택사항. 푸터는 전체 기본 메시지가 구성되거나 표시된 후에만(예: 메시지 해시, HMAC, 서명 및 암호화 세부사항) 계산 또는 평가될 수 있는 메시지 또는 전달에 대한 자세한 내용에 사용됩니다.

AMQP 메시지 형식에 대해서는 다음 그림에 설명되어 있습니다.



특성, 애플리케이션 특성 및 애플리케이션 데이터 부분을 "기본 메시지"라 합니다. 이는 송신자에 의해 송신된 메시지이며 변경할 수 없습니다. 수신자는 헤더, 푸터, 전달 어노테이션 및 메시지 어노테이션을 포함한 전체 메시지를 봅니다.

AMQP 1.0 메시지 형식에 대한 전체 설명은 OASIS Standard(<https://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf>)를 참조하십시오.

관련 태스크

[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

▶ ALW AMQP 필드에 IBM MQ 필드 맵핑(보내는 메시지)

IBM MQ 메시지가 발행되고 IBM MQ가 이를 AMQP 이용자에게 송신하면 IBM MQ 메시지의 일부 속성을 등가의 AMQP 메시지 속성에 전파합니다.

헤더

헤더는 헤더의 5개 필드 중 하나에 비기본 값이 포함된 경우에만 포함됩니다. 비기본 값이 있는 필드만 헤더에 포함됩니다. 5개의 헤더 필드는 등가의 `mq_amqp.Hdr` 특성에서 초기에 파생되며, 설정된 경우 다음 표에 표시된 대로 수정됩니다.

표 103. 헤더 필드 맵핑		
필드	기본값	값
지속 가능	false	MQMD.Persistence가 MQPER_PERSISTENT로 설정된 경우 True이며, 그렇지 않으면 False입니다.
priority	4	설정된 경우 mq_amqp.Hdr.Pri에서의 값이며, 그렇지 않으면 설정된 경우 MQMD.Priority에서의 값입니다. 둘 다 설정되지 않은 경우 4로 설정됩니다.
ttl	해당사항 없음	MQMD.Expiry(밀리초)입니다. MQMD.Expiry의 값이 MQEI_UNLIMITED인 경우, AMQP ttl 필드에 대한 최대값으로 설정됩니다.
first-acquirer	false	설정된 경우 mq_amqp.Hdr.Fac에서의 값이며, 그렇지 않으면 False입니다.
delivery-count	0	설정된 경우 mq_amqp.Hdr.Dct에서의 값이며 그렇지 않으면 0입니다.

전달 어노테이션

AMQP 채널별로 필요에 따라 설정됩니다.

메시지 어노테이션

포함되지 않습니다.

특성

특성은 설정된 경우 등가의 mq_amqp.Prp 특성에서 수정되지 않습니다. 메시지가 원래 AMQP 메시지가 아닌 경우(즉, PutApplType이 MQAT_AMQP가 아닌 경우), 다음 표에 설명된 대로 특성 섹션이 생성됩니다.

표 104. 특성 필드 맵핑	
이름	값
message-id	MQMD.MsgId가 2진으로 설정됩니다.
사용자 ID	MQMD.UserIdentifier의 UTF-8 양식이 네트워크 바이트 순서에서 2진으로 설정됩니다.
-	메시지를 얻은 큐 또는 발행용 토픽 문자열입니다.
subject	설정되지 않음.
reply-to	공백이 아닌 경우 MQMD.ReplyToQ이고, 그렇지 않으면 설정되지 않습니다.
correlation-id	공백이 아닌 경우 MQMD.CorrelId가 2진으로 설정되며, 그렇지 않으면 설정되지 않습니다.
content-type	설정되지 않음.
content-encoding	설정되지 않음.
absolute-expiry-time	설정되지 않음.
creation-time	시간소인 생성을 위해 MQMD.PutDate 및 MQMD.PutTime 필드가 사용됩니다.
group-id	설정되지 않음.
group-sequence	설정되지 않음.

표 104. 특성 필드 맵핑 (계속)	
이름	값
reply-to-group-id	설정되지 않음.

애플리케이션 특성

"usr" 그룹의 모든 IBM MQ 속성은 **application-properties** 로 추가됩니다.

본문

AMQP 채널은 IBM MQ 페이로드를 UTF-8로 변환하기 위해 변환과 함께 가져오기를 수행합니다.

IBM MQ 페이로드에 AMQP 메시지가 포함되지 않은 경우, IBM MQ 페이로드가 형식 MQFMT_STRING에 대한 단일 문자열 데이터 섹션으로 본문에서 설정되며(UTF-8에 대해 제공된 변환에 성공), 그렇지 않으면 단일 2진 데이터 섹션으로 설정됩니다.

AMQP 형식 메시지가 포함된 경우, 이는 본문으로 설정됩니다. AMQP 메시지에 선행하는 임의의 IBM MQ 헤더(메시지 핸들에서 리턴된 메시지 특성을 포함하지 않음)는 본문이 AMQP 순서인 경우 2진 값으로 앞에 추가됩니다. 그렇지 않으면 IBM MQ 헤더는 제거됩니다.

푸터

푸터가 포함되지 않습니다.

관련 태스크

[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

관련 참조

[MQMD - 메시지 디스크립터](#)

ALW AMQP 필드를 IBM MQ 필드(수신되는 메시지)에 맵핑

AMQP 채널이 메시지를 수신하고 이를 IBM MQ에 넣으면, 이는 AMQP 메시지의 일부 속성을 등가의 IBM MQ 메시지 속성에 전파합니다.

다음 제한사항은 수신되는 AMQP 메시지를 맵핑할 때 적용됩니다.

- 특성 부분의 message-id 또는 correlation-id 필드가 uuid 또는 ulong인 경우, 메시지가 거부됩니다.
- 메시지가 거부되는 원인이 될 message-annotations 수 있습니다.
- delivery-annotations 섹션은 footer 허용되지만 메시지에는 IBM MQ 전파되지 않습니다.

다음 서브 섹션은 AMQP 메시지의 IBM MQ 표현식을 표시합니다.

메시지 디스크립터

표 105. AMQP 메시지	
필드	값
StrucId	MQMD_STRUC_ID
버전	MQMD_VERSION_1
보고서	MQRO_NONE
MsgType	MQMT_DATAGRAM
만기	AMQP 메시지 헤더에서 ttl 필드로부터 사용된 값
Feedback	MQFB_NONE

표 105. AMQP 메시지 (계속)	
필드	값
Encoding	MQENC_NORMAL
CodedCharSetId	1208 (UTF-8)
형식	페이로드 참조
Priority	AMQP 메시지 헤더에서 <code>priority</code> 필드로부터 사용된 값. 설정한 경우 최대값 9로 제한됩니다. 설정하지 않으면 기본값 4가 사용됩니다.
Persistence	AMQP 메시지 헤더의 <code>durable</code> 필드가 True로 설정된 경우, <code>MQPER_PERSISTENT</code> 로 설정하십시오. 그렇지 않으면 <code>MQPER_NOT_PERSISTENT</code> 로 설정하십시오.
MsgId	큐 관리자는 고유 24바이트 <code>MsgId</code> 를 할당합니다.
CorrelId	설정된 경우 AMQP 특성의 <code>correlation-id</code> 필드에서 사용된 값입니다. 24바이트 2진 값으로 설정하십시오. 그렇지 않으면 <code>MQCI_NONE</code> 으로 설정됩니다.
BackoutCount	0
ReplyToQ	설정된 경우 AMQP 특성의 <code>reply-to</code> 필드에서 사용된 값입니다. 그렇지 않으면 ""로 설정합니다.
ReplyToQMgr	""
보고서	AMQP 애플리케이션 특성에 설정된 JMS IBM 보고서 특성에서 파생된 값입니다.
UserIdentifier	AMQP 채널에 연결된 인증된 사용자의 ID로 설정됨
AccountingToken	MQACT_NONE
ApplIdentityData	16진수 문자열. AMQP 채널의 MQ 연결 ID의 마지막 8바이트로 설정됩니다.
PutApplType	MQAT_AMQP
PutApplName	
PutDate	설정된 경우 AMQP 특성의 <code>creation-time</code> 필드에서 사용된 값입니다. 그렇지 않으면 현재 날짜로 설정됩니다.
PutTime	설정된 경우 AMQP 특성의 <code>creation-time</code> 필드에서 사용된 값입니다. 그렇지 않으면 현재 시간으로 설정됩니다.
ApplOriginData	""

메시지 특성

메시지 특성 설정에는 두 가지 이유가 있습니다.

- 메시지의 페이로드에 영향을 주지 않고 AMQP 메시지의 부분이 큐 관리자를 통해 플로우되도록 허용합니다.
- `.application-properties`선택을 허용하기 위해

다음 표에서는 AMQP 메시지에서 설정된 특성을 보여줍니다.

표 106. AMQP 메시지 특성

특성 이름	MQRFH2 이름	유형	설명
AMQPListener	mq_amqp.Lis	MQTYPE_STRING	AMQP 채널의 식별 문자열입니다. 관심 있는 부서에서 메시지를 넣는 버전을 알려줄 수 있도록 메시지를 생성하는 데 사용됩니다(예를 들어 문제점을 진단할 경우 서비스 팀). 값은 큐 관리자에 의해 유효성 검증되지 않으며, 외부적으로 문서화되지 않아야 합니다.
AMQPVersion	mq_amqp.Ver	MQTYPE_STRING	AMQP 메시지의 버전입니다. 없는 경우 "1.0"이 가정됩니다. 값은 큐 관리자에 의해 유효성 검증되지 않습니다.
AMQPClient	mq_amqp.Cli	MQTYPE_STRING	API에 대한 식별 문자열입니다. 관심 있는 부서에서 메시지를 넣는 버전을 알려줄 수 있도록 AMQP 메시지를 채널에 송신하는 데 사용됩니다(예를 들어 문제점을 진단할 경우 서비스 팀). 값은 큐 관리자에 의해 유효성 검증되지 않으며, 외부적으로 문서화되지 않아야 합니다.
AMQPDurable	mq_amqp.Hdr.Dur	MQTYPE_BOOLEAN	설정된 경우 AMQP 메시지 헤더의 durable 필드의 값입니다.
AMQPPriority	mq_amqp.Hdr.Pri	MQTYPE_INT32	설정된 경우 AMQP 메시지 헤더의 priority 필드의 값입니다.
AMQPTtl	mq_amqp.Hdr.Ttl	MQTYPE_INT64	설정된 경우 AMQP 메시지 헤더의 ttl 필드의 값입니다.
AMQPFirstAcquirer	mq_amqp.Hdr.Fac	MQTYPE_BOOLEAN	설정된 경우 AMQP 메시지 헤더의 first-acquirer 필드의 값입니다.
AMQPDeliveryCount	mq_amqp.Hdr.Dct	MQTYPE_INT64	설정된 경우 AMQP 메시지 헤더의 delivery-count 필드의 값입니다.
AMQPMsgId	mq_amqp.Prp.Mid	MQTYPE_STRING	문자열로 설정된 경우 AMQP 특성에서 message-id 필드의 값입니다.
		MQTYPE_BYTE_STRING	바이트 문자열로 설정된 경우 AMQP 특성에서 message-id 필드의 값입니다.
AMQPUserId	mq_amqp.Prp.Uid	MQTYPE_BYTE_STRING	설정된 경우 AMQP 특성의 user-id 필드의 값입니다.
AMQPTo	mq_amqp.Prp.To	MQTYPE_STRING	설정된 경우 AMQP 특성의 to 필드의 값입니다.
AMQPSubject	mq_amqp.Prp.Sub	MQTYPE_STRING	설정된 경우 AMQP 특성의 subject 필드의 값입니다.
AMQPReplyTo	mq_amqp.Prp.Rto	MQTYPE_STRING	설정된 경우 AMQP 특성의 reply-to 필드의 값입니다.
AMQPCorrelationId	mq_amqp.Prp.Cid	MQTYPE_STRING	문자열로 설정된 경우 AMQP 특성에서 correlation-id 필드의 값입니다.

표 106. AMQP 메시지 특성 (계속)			
특성 이름	MQRFH2 이름	유형	설명
		MQTYPE_BYTE_STRING	바이트 문자열로 설정된 경우 AMQP 특성에서 correlation-id 필드의 값입니다.
AMQPContentType	mq_amqp.Prp.Cnt	MQTYPE_STRING	설정된 경우 AMQP 특성의 content-type 필드의 값입니다.
AMQPContentEncoding	mq_amqp.Prp.Cne	MQTYPE_STRING	설정된 경우 AMQP 특성의 content-encoding 필드의 값입니다.
AMQPAbsoluteExpiryTime	mq_amqp.Prp.Aet	MQTYPE_STRING	설정된 경우 AMQP 특성의 absolute-expiry-time 필드의 값입니다.
AMQPCreationTime	mq_amqp.Prp.Crt	MQTYPE_STRING	설정된 경우 AMQP 특성의 creation-time 필드의 값입니다.
AMQPGroupId	mq_amqp.Prp.Gid	MQTYPE_STRING	설정된 경우 AMQP 특성의 group-id 필드의 값입니다.
AMQPGroupSequence	mq_amqp.Prp.Gsq	MQTYPE_INT64	설정된 경우 AMQP 특성의 group-sequence 필드의 값입니다.
AMQPReplyToGroupId	mq_amqp.Prp.Rtg	MQTYPE_STRING	설정된 경우 AMQP 특성의 reply-to-group-id 필드의 값입니다.

AMQP 메시지의 각 애플리케이션 특성은 IBM MQ 메시지 특성으로 설정됩니다. 해당 application-properties 섹션은 바이트 단위로 동일하게 재구성되어야 하며, 따라서 다음과 같은 제한 사항이 적용됩니다:

- 애플리케이션 특성이 MQSETMP 유효성 검증 코드에 의해 거부되는 경우, 거부될 메시지입니다. 예를 들면, 다음과 같습니다.
 - 특성 이름은 MQ_MAX_PROPERTY_NAME_LENGTH로 길이가 제한됩니다.
 - 속성 이름은 식별자에 Java 대한 언어 Java 사양에서 정의한 규칙을 따라야 합니다.
 - 특성 이름은 설정할 수 있는 문서화된 JMS 특성을 제외하고 JMS 또는 usr.JMS로 시작하지 않아야 합니다.
 - 특성 이름은 SQL 키워드가 아니어야 합니다.
- 유니코드 문자 U+002E(".")가 포함된 애플리케이션 특성은 메시지가 거부되도록 합니다. 특성은 JMS에서 사용된 특성의 "usr" 그룹에서 표현 가능해야 합니다.
- null, boolean, byte, short, int, long, float, double, binary 및 string 특성만 지원됩니다. 다른 유형을 가진 애플리케이션 특성은 메시지가 거부되도록 유발합니다.

application-properties를 사용하여 다음 JMS 특성을 설정할 수 있습니다.

- [JMS_IBM_REPORT_EXCEPTION](#)
- [JMS_IBM_REPORT_EXPIRATION](#)
- [JMS_IBM_REPORT_COA](#)
- [JMS_IBM_REPORT_COD](#)
- [JMS_IBM_REPORT_PAN](#)
- [JMS_IBM_REPORT_NAN](#)
- [JMS_IBM_REPORT_PASS_MSG_ID](#)
- [JMS_IBM_REPORT_PASS_CORREL_ID](#)
- [JMS_IBM_REPORT_DISCARD_MSG](#)

특성 이름 및 값은 해당하는 155 페이지의 『JMS 제공자 특정 필드 매핑』 세부사항과 일치하며 유효하지 않은 값은 무시됩니다.

페이로드

- 단일 바이너리 데이터 섹션을 가진 body AMQP의 경우, 바이너리 데이터(AMQP 비트 제외)는 MQFMT_NONE 형식으로 IBM MQ 페이로드에 배치됩니다.
- 단일 문자열 데이터 섹션을 가진 body AMQP의 경우, 문자열 데이터(AMQP 비트 제외)는 MQFMT_STRING 형식으로 IBM MQ 페이로드에 배치됩니다.
- 그렇지 않으면 AMQP는 페이로드를 body 그대로 구성하며, 형식은 MQFMT_AMQP로 지정됩니다.

관련 태스크

[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

ALW 메시지 전달 안정성

이 섹션에서는 API와 MQ LightApache Qpid의 신뢰성 JMS기능을 비교합니다.

관련 태스크

[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

ALW MQ Light 메시지 신뢰도

MQ Light API의 네 가지 기능을 사용하여 AMQP 애플리케이션과의 메시지 전달 신뢰도를 제어할 수 있습니다.

이러한 항목은 다음과 같습니다.

- 653 페이지의 『메시지 서비스 품질(QoS)』
- 654 페이지의 『구독자 자동 확인』
- 654 페이지의 『구독 잔존 시간』
- 654 페이지의 『메시지 지속성』

메시지 서비스 품질(QoS)

MQ Light API에서는 두 가지 서비스 품질(QoS)을 제공합니다.

- 최대 한 번
- 최소 한 번

발행자와 구독자가 사용하게 할 서비스 품질(QoS)을 선택할 수 있습니다.

MQ Light 클라이언트를 사용하는 경우 클라이언트 또는 구독 **qos** 옵션을 *QOS_AT_MOST_ONCE* 또는 *QOS_AT_LEAST_ONCE*로 설정하십시오.

다른 AMQP 클라이언트를 사용하는 경우, 달성하려는 서비스 품질에 따라 전송 프레임(발행인의 경우) 또는 처리 프레임(구독자의 경우)의 **settled** 속성을 *true* 또는 *false*로 설정하십시오.

서비스 품질은 대화의 어느 sending 쪽에서 메시지가 폐기되는지를 결정합니다.

발행

- 발행자가 **qos 0**(최대 한 번)을 선택하면 발행자가 큐 관리자의 수신확인을 기다리지 않고 메시지의 사본을 버립니다. 송신이 완료되기 전에 큐 관리자에 연결하는 데 실패하는 경우에는 구독자가 메시지를 받지 못합니다.
- 발행자가 **qos 1**(최소 한 번)을 선택하면 발행자가 메시지의 사본을 버리기 전에 큐 관리자가 구독자 큐에 메시지가 기록되었음을 확인할 때까지 대기합니다. 송신 중에 큐 관리자에 연결하는 데 실패하는 경우에는 발행자가 큐 관리자에 다시 연결되면 메시지를 다시 송신합니다.

구독

- 구독자가 **QOS 0**을 선택하면 큐 관리자가 구독자의 수신확인을 기다리지 않고 메시지의 사본을 버립니다. 구독자가 메시지를 수신하기 전에 구독자에 연결하는 데 실패하면 해당 메시지가 손실될 수 있습니다.
- 구독자가 **QOS 1**을 선택하면 큐 관리자가 메시지의 사본을 버리기 전에 구독자의 수신확인을 대기합니다. 성능 향상을 위해 수신 IBM MQ 9.3.3 확인된 메시지는 일괄적으로 제거됩니다. 자세한 정보는 [656 페이지](#)의 『큐에서 수신 확인된 AMQP 메시지를 일괄적으로 제거하기』의 내용을 참조하십시오.

구독자가 메시지를 수신하기 전에 구독자에 연결하는 데 실패하면 큐 관리자가 메시지를 보관합니다. 큐 관리자가 다시 연결되면 구독자에게 메시지를 다시 보내거나 구독이 공유되는 경우에는 다른 구독자에게 메시지를 보냅니다.

구독자 자동 확인

구독자가 **QOS 1**(최소 한 번)을 선택하면 큐 관리자가 사본을 버리기 전에 각 메시지의 수신을 확인해야 합니다. 구독자는 메시지를 수신확인하는 시기를 결정할 수 있습니다.

auto-confirm을(를) *true*로 설정한 경우 클라이언트가 네트워크를 통해 메시지를 성공적으로 수신하면 MQ Light 클라이언트가 각 메시지의 전달을 자동으로 수신확인합니다.

이와 같이 설정하면 네트워크 장애가 있는 경우 메시지가 애플리케이션에 다시 전달됩니다. 그러나 MQ Light 클라이언트가 메시지를 수신확인하고 애플리케이션이 이를 처리하는 사이에 애플리케이션이 실패하는 경우 애플리케이션에서 메시지가 손실될 가능성이 여전히 있습니다.

auto-confirm을(를) *false*로 설정하면 MQ Light 클라이언트가 메시지 전달을 자동으로 수신확인하지 않고, 애플리케이션에서 수신확인해야 하는 시기를 결정하도록 둡니다.

이와 같이 설정하면 메시지가 처리되었으며 메시지를 제거할 수 있음을 큐 관리자에게 수신확인하기 전에 애플리케이션이 데이터베이스 또는 파일과 같은 외부 자원에 업데이트할 수 있습니다.

구독 잔존 시간

구독할 때 애플리케이션은 구독 및 해당 구독에 대한 메시지가 저장되는 목적지가 애플리케이션의 연결이 끊어진 후에도 계속 존재할지 여부를 선택합니다.

MQ Light 구독 옵션 **ttl**은(는) 애플리케이션 연결이 끊어진 후에도 구독이 계속 존재하는 시간(밀리초)을 지정하는 데 사용됩니다. 이 시간 전에 애플리케이션이 다시 연결되면 구독이 재개되고 애플리케이션은 해당 구독의 메시지를 계속 이용할 수 있습니다.

애플리케이션이 다시 연결되지 않은 채 잔존 시간 기간이 지나면 구독은 제거되고 지속 메시지를 포함하여 목적지에 저장되어 있는 모든 메시지가 손실됩니다.

메시지가 손실되지 않는 것이 중요한 경우 가동 중단 동안 메시지가 손실되지 않도록 충분히 큰 잔존 시간 값을 애플리케이션에 지정해야 합니다.

메시지 지속성

발행 애플리케이션, 구독 애플리케이션, IBM MQ 토픽 오브젝트에서 메시지의 지속성을 제어합니다.

AMQP 구독자가 **QOS 0**(최대 한 번)을 사용하고 지속 불가능한 구독을 작성하는 경우 AMQP 채널은 다음 텍스트에 설명된 기타 옵션에 관계없이 항상 비지속 메시지를 구독자 큐에 넣습니다.

큐 관리자가 중지되면 구독과 메시지 모두 손실됩니다.

AMQP 구독자가 AMQP **durable** 헤더를 *true*로 설정하면 AMQP 채널은 지속 메시지를 구독자 큐에 넣습니다.

큐 관리자가 어떤 이유로든 중지되는 경우 큐 관리자를 재시작하면 구독자가 여전히 메시지를 사용할 수 있습니다.

durable 헤더가 설정되지 않은 경우, AMQP 채널은 관련 IBM MQ 토픽 오브젝트의 **DEFPSIST** 속성을 기반으로 게시된 메시지의 지속성을 선택합니다.

기본적으로 **DEFPSIST** 속성을 *NO*(비지속)로 사용하는 **SYSTEM.BASE.TOPIC**입니다.



주의: MQ Light 클라이언트의 이후 버전은 AMQP 지속 가능 헤더 설정을 지원하지 않습니다.

관련 태스크

[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

Apache Qpid JMS 메시지 안정성

Apache Qpid™ JMS 라이브러리에는 AMQP 애플리케이션과 주고 받는 메시지 전달의 신뢰성을 제어할 수 있는 네 가지 기능이 있습니다.

다음에 대한 항목입니다.

- [655 페이지의 『발행』](#) 포인트-투-포인트 메시징을 위한 생성자
 - 메시지 만기
 - 메시지 지속성
- [655 페이지의 『구독』](#)
 - 구독 지속성
 - 세션 확인 모드 (소비자 간 지점 간 메시징에도 적용 가능)

발행

메시지 만기

JMS 생성자의 잔존 시간 값을 설정하면 메시지 생성자에서 발행한 메시지에 주어진 만기 시간에 영향을 줍니다.

JMS 생성자의 잔존 시간 값이 만기 전에 메시지를 이용하기에 충분한지 확인하십시오.

또는 잔존 시간 값을 설정하지 않으면 구독 큐에서 메시지 만기를 방지합니다.

메시지 지속성

JMS 메시지 생성자의 전달 모드를 설정하면 지정된 토픽에 발행된 IBM MQ 메시지의 지속성을 설정합니다.

큐 관리자가 종료되거나 가동 중단된 경우 보존해야 하는 메시지에 대해 **DeliveryMode.PERSISTENT**를 사용하는지 확인하십시오.

구독

구독 지속성

AMQP 채널은 JMS 사용자 작성 메소드의 지속 가능한 버전을 사용하여 지속 가능한 구독 작성을 지원합니다.

- **createDurableConsumer()**
- **createSharedDurableConsumer()**

세션 수신확인 모드

구독 IBM MQ 대기열에서 메시지가 제거되기 전에 소비된 메시지가 완전히 처리되었음을 보장하려면,.

SessionCLIENT_ACKNOWLEDGE 모드를 사용하여 세션을 JMS 생성하고 이 메시지와 해당 세션에서 이전에 수신된 모든 메시지를 확인하기 위해 **message.acknowledge()** 메서드를 사용하십시오.

관련 개념

[AMQP 클라이언트 애플리케이션 개발](#)

AMQP API에 대한 IBM MQ 지원을 통해 IBM MQ 관리자가 AMQP 채널을 작성할 수 있습니다. 시작되면 이 채널은 AMQP 클라이언트 애플리케이션에서 연결을 승인하는 포트 번호를 정의합니다.

큐에서 수신 확인된 AMQP 메시지를 일괄적으로 제거하기

AMQP 애플리케이션이 QOS_AT_LEAST_ONCE(1) 메시지 전달 방식을 사용하는 경우, AMQP 서비스는 애플리케이션에 메시지를 전송한 후 해당 애플리케이션으로부터 수신 확인을 받기 전까지 애플리케이션에 보낸 메시지의 사본을 유지합니다. 확인된 IBM MQ 9.3.3 메시지는 개별적으로가 아닌 일괄적으로 큐에서 제거되어 성능이 향상됩니다.

이 태스크 정보

이전에는 IBM MQ 9.4.0 각 메시지가 개별적으로 큐에서 제거되었습니다.

다음 두 시스템 속성 **com.ibm.mq.AMQP.BATCHSZ** 와 **com.ibm.mq.AMQP.BATCHINT** IBM MQ 9.4.0를 사용하여 배치 처리 시 확인 응답 처리를 미세 조정하여 성능을 향상시킬 수 있습니다:

com.ibm.mq.AMQP.BATCHSZ

이 속성은 AMQP 서비스가 메시지를 제거하기 전에 수신해야 하는 최대 확인 응답 수를 정의합니다. 숫자는 1 - 9999 범위일 수 있습니다. 잘못된 숫자가 설정되었거나 지정된 숫자가 범위를 벗어날 경우, 기본값인 50이 사용됩니다.

배치 크기는 메시지 전송 방식에 영향을 미치지 않습니다. 메시지는 항상 개별적으로 전송되지만, AMQP 서비스가 확인 응답을 수신한 후 일괄적으로 제거됩니다. 배치의 실제 크기는 지정된 값보다 작을

com.ibm.mq.AMQP.BATCHINT 수 있습니다. 예를 들어, 배치 작업은 **com.ibm.mq.AMQP.BATCHINT** attribute 로 설정된 기간이 만료되면 완료됩니다.

com.ibm.mq.AMQP.BATCHINT

이 속성은 AMQP 서비스가 큐에 확인된 메시지를 유지하는 시간을 밀리초 단위로 정의합니다. 배치가 가득 차지 않은 경우, 이 기간이 지나면 배치가 정리됩니다. 1에서 999,999,999까지 원하는 밀리초 수를 지정할 수 있습니다. 기본값은 50입니다. 이 속성에 값을 지정하지 않으면 기본값인 50이 사용됩니다.

참고:

1. AMQP 서비스가 메시지를 폐기하기 전에 확인 응답을 기다리는지 여부는 애플리케이션이 메시지 전달에 사용하는 다음 두 가지 서비스 품질 중 어느 것을 사용하는지에 따라 달라집니다:

- QOS_AT_MOST_ONCE(0)에 대한 QOS

AMQP 애플리케이션이 이 품질 수준을 사용하는 경우 메시지를 확인하지 않으므로, AMQP 서비스는 애플리케이션에 메시지를 전송한 후 확인을 기다리지 않고 메시지를 폐기합니다.

- QOS_AT_LEAST_ONCE(1)

AMQP 애플리케이션이 이 품질 수준을 사용하는 경우 메시지에 대한 확인을 수행하므로, AMQP 서비스는 애플리케이션에 메시지를 전송한 후 애플리케이션으로부터 확인을 받을 때까지 각 메시지의 사본을 보관합니다. 응용 프로그램이 메시지를 수신 확인하기 전에 연결이 끊어지거나 연결을 잃는 경우, 해당 메시지는 다른 응용 프로그램에 제공됩니다. AMQP 서비스는 메시지가 수신 확인될 때까지 큐에서 메시지를 제거하지 않습니다.

2.  및 **com.ibm.mq.AMQP.BATCHINT** **com.ibm.mq.AMQP.BATCHSZ** 시스템 속성은 에 적용되지 IBM MQ Appliance 않습니다. 기본값 50이 사용됩니다 IBM MQ Appliance.

프로시저

배치 단위의 확인 응답 처리를 미세 조정하려면 **com.ibm.mq.AMQP.BATCHINT** 시스템

com.ibm.mq.AMQP.BATCHSZ 속성을 사용하십시오.

큐 관리자가 생성될 때 IBM MQ 9.3.3, 파일에는 `amqp_java.properties` 시스템 속성에 대한 다음과 같은 기본값이 포함됩니다:

```
-Dcom.ibm.mq.AMQP.BATCHSIZE=50
-Dcom.ibm.mq.AMQP.BATCHINT=50
```

소비되는 메시지 속도에 따라, 성능 향상을 위해 일괄 처리된 확인 응답 처리를 미세 조정할 수 있습니다. 이동된 큐 관리자는 해당 `amqp_java.properties` 파일 내에 이러한 속성을 가지고 있지 않습니다. 따라서 마이그레이

이전된 큐 관리자나 속성이 설정되지 않은 경우 기본값이 사용됩니다. 최적화된 성능을 위해 값을 미세 조정할 수 있도록 이러한 속성을 추가할 수 있습니다.

다음 조건 중 하나가 충족될 때 확인된 메시지는 일괄적으로 제거됩니다:

- 확인된 메시지 수가 `com.ibm.mq.AMQP.BATCHSZ`에 도달합니다.
- `com.ibm.mq.AMQP.BATCHINT` 배치 시작 후 초과됩니다.
- 응용 프로그램은 이전 두 조건이 충족되기 전에 큐 또는 토픽을 분리하거나 닫습니다.

ALW IBM MQ가 있는 AMQP 클라이언트의 토폴로지

IBM MQ와 함께 작업하도록 AMQP 클라이언트를 개발하는 데 도움이 되는 토폴로지 예입니다.

관련 태스크

[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

ALW IBM MQ를 통해 통신하는 AMQP 클라이언트

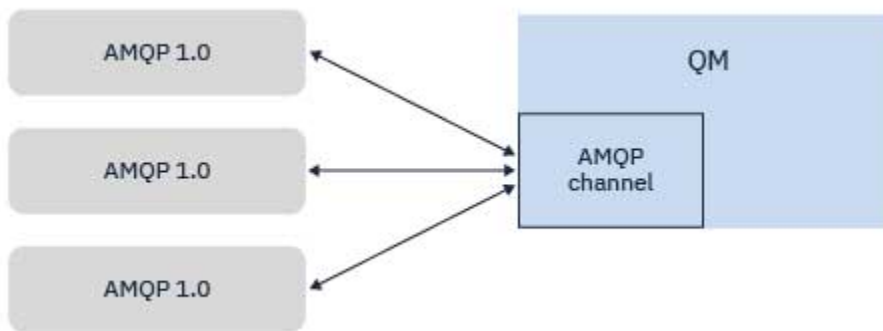
AMQP 1.0을 준수하는 모든 애플리케이션의 메시징 제공자로 IBM MQ를(를) 사용할 수 있습니다. AMQP 1.0 클라이언트를 AMQP 채널에 연결할 수 있지만 일부 AMQP 기능은 지원되지 않습니다(예: 트랜잭션 또는 다중 세션).

하나 이상의 AMQP 채널을 정의하여 AMQP 1.0 클라이언트가 큐 관리자에 연결하고 토픽 문자열에 메시지를 송신할 수 있습니다. 클라이언트는 토픽 패턴을 구독하여 패턴과 일치하는 메시지를 수신할 수 있습니다.

다음 시나리오에서 메시지를 보내고 받는 유일한 애플리케이션은 AMQP 1.0 애플리케이션뿐입니다.

애플리케이션에서 일시적으로 큐 관리자에 대한 연결이 끊어지는 경우 메시지가 손실되지 않도록 토픽 문자열을 구독하여 작성된 목적지가 지속되는지 여부를 애플리케이션이 선택할 수 있습니다.

애플리케이션은 목적지에서 제거되기 전에 메시지가 보관되는 기간을 선택할 수도 있습니다.



관련 태스크

[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

ALW IBM MQ 애플리케이션과 메시지를 교환하는 AMQP 클라이언트

AMQP 1.0 애플리케이션은 AMQP 채널을 정의하고 시작하여 기존 MQ 애플리케이션에서 수신한 메시지를 발행할 수 있습니다. AMQP 채널을 통해 발행된 메시지는 모두 MQ 큐가 아니라 MQ 토픽에 송신됩니다. MQ 애플리케이션이 사용하는 토픽 오브젝트 또는 토픽 문자열이 AMQP 클라이언트가 발행하는 토픽 문자열과 일치하는 경우 MQSUB API 호출을 사용하여 구독을 작성한 MQ 애플리케이션은 AMQP 1.0 애플리케이션이 발행한 메시지를 수신합니다.

AMQP 메시지 데이터, 속성, 특성은 MQ 애플리케이션이 수신하는 MQ 메시지에서 설정됩니다. AMQP에서 MQ 메시지 매핑에 대한 자세한 내용은 [참조하십시오 649 페이지의 『AMQP 필드를 IBM MQ 필드\(수신되는 메시지\)에 매핑』](#).

MQ 애플리케이션이 지속 가능한 구독을 작성한 경우 AMQP 애플리케이션이 발행하는 메시지는 구독을 지원하는 큐에 저장됩니다. 애플리케이션이 해당 구독을 재개하는 경우 MQ 애플리케이션이 메시지를 수신합니다. AMQP 애플리케이션이 메시지 수명을 지정하고 MQ 애플리케이션이 수명 내에 다시 연결되지 않는 경우 메시지가 큐에서 만기됩니다.

AMQP 1.0 애플리케이션은 기존 MQ 애플리케이션에서 발행되는 메시지를 이용할 수도 있습니다. 애플리케이션이 발행된 토픽 문자열과 일치하는 토픽 패턴으로 구독된 경우 토픽 문자열 또는 MQ 토픽에 대해 MQ 애플리케이션이 발행하는 메시지가 AMQP 1.0 애플리케이션에 의해 수신됩니다.

AMQP 1.0 애플리케이션이 구독에 대해 수명 값을 지정하고 AMQP 애플리케이션이 수명보다 긴 시간동안 연결이 끊어지는 경우 구독이 큐 관리자에서 만기되고 구독 큐에서 저장된 메시지가 손실됩니다.

MQMD 필드, 메시지 특성, 애플리케이션 데이터는 AMQP 애플리케이션이 수신한 AMQP 메시지에서 설정됩니다. MQ에서 AMQP 메시지 매핑에 대한 자세한 내용은 [을 참조하십시오 647 페이지의 『AMQP 필드에 IBM MQ 필드 매핑\(보내는 메시지\)』](#).

관련 태스크

[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

▶ ALW IBM MQ 큐에서 애플리케이션과 직접 상호작용하도록 AMQP 클라이언트 구성

AMQP IBM MQ 구현은 퍼블리시/서브스크라이브 및 포인트 투 포인트를 지원합니다. 포인트 투 포인트를 지원하지 않는 모든 AMQP 클라이언트의 경우, 다음 단계를 따라 큐에 메시지를 전송하거나 큐에서 메시지를 수신하십시오.

개요

예를 들어, 입력 큐 IN_QUEUE에서 메시지를 가져오고 해당 메시지를 출력 큐 OUT_QUEUE에 넣는 애플리케이션이 있다고 가정합니다. AMQP 클라이언트가 IN_QUEUE에 메시지를 넣고 OUT_QUEUE에서 메시지를 가져올 수 있습니다.

참고: 애플리케이션 자체를 변경하지 않아도 됩니다.



큐에 메시지를 넣는 AMQP 발행자의 경우 의도한 큐의 목적지를 사용하여 AMQP 클라이언트가 발행하는 토픽 문자열에 대한 관리 구독을 작성해야 합니다([658 페이지의 『애플리케이션에 메시지 전송』](#) 참조).

큐에서 메시지를 가져오는 AMQP 구독자의 경우 AMQP 클라이언트가 구독하는 토픽 문자열을 나타내는 토픽 오브젝트의 대상을 사용하여 동일한 이름의 알리어스 큐로 큐를 바꾸어야 합니다([659 페이지의 『애플리케이션에서 메시지 가져오기』](#) 참조).

애플리케이션에 메시지 전송

애플리케이션이 이미 IN_QUEUE에서 메시지를 선택하고 있으며 애플리케이션에서 처리하게 이 큐로 이동하도록 AMQP 클라이언트가 메시지를 발행할 수 있게 합니다.

이를 위해 새 관리 구독을 작성합니다. 이때 이 구독이 메시지를 수신하는 토픽 문자열은 AMQP 클라이언트가 발생하는 토픽 문자열입니다. 이 구독의 목적지 큐는 IN_QUEUE 애플리케이션의 입력 큐입니다.

해당 관리 구독에 대해 정의된 토픽 문자열에 발행되는 모든 메시지는 정의된 목적지로 라우팅됩니다(이 경우 IN_QUEUE).

AMQP 클라이언트가 토픽 문자열 /application/in에 발행한다고 가정하면 다음 MQSC 명령을 사용하여 관리 구독 APP_IN을(를) 작성할 수 있습니다.

```
DEF SUB(APP_IN) TOPICSTR('/application/in') DEST('IN_QUEUE')
```

이 오브젝트를 정의하면 /application/in에 발행된 모든 메시지가 목적지 IN_QUEUE(으)로 라우팅됩니다. 여기서 다른 애플리케이션이 이 큐에 넣은 다른 메시지와 동일한 방식으로 애플리케이션이 선택합니다.

애플리케이션에서 메시지 가져오기

애플리케이션이 메시지를 OUT_QUEUE에 넣습니다. 여기서 다른 클라이언트가 이 메시지를 선택하고 처리할 수 있습니다.

그러나 이 경우 AMQP 클라이언트로 대신 메시지를 전달하려고 하지만 AMQP 클라이언트가 발행/구독만 사용하는 경우 큐에서 직접 메시지를 선택할 수 없습니다.

이전에 메시지를 수신하는 클라이언트를 구독 AMQP 클라이언트로 바꾸려면 알리어스 큐 및 AMQP 클라이언트가 구독하는 토픽 문자열에 대해 토픽 오브젝트를 작성해야 합니다.



주의: 알리어스 큐를 정의한 후 AMQP 클라이언트에서 구독하기 전에 생성 애플리케이션을 시작한 경우 생성 애플리케이션이 "큐"로 전송하는 메시지(현재 토픽)는 구독자가 없으므로 유실됩니다.

이 텍스트에서 설명하는 변경사항은 이전에 메시지를 수신하는 클라이언트를 구독 AMQP 클라이언트로만 바꿉니다. AMQP와 다른 클라이언트의 조합을 사용하여 메시지를 가져오려면 보다 광범위한 변경이 필요합니다.

AMQP 클라이언트가 토픽 문자열 /application/out을(를) 구독한다고 가정하면 다음 MQSC 명령을 사용하여 APP_OUT 토픽 오브젝트를 정의할 수 있습니다.

```
DEF TOPIC(APP_OUT) TOPICSTR('/application/out')
```

이 토픽 오브젝트에 전달된 모든 메시지는 동일한 토픽 문자열로 구독하는 AMQP 클라이언트에 전달됩니다.

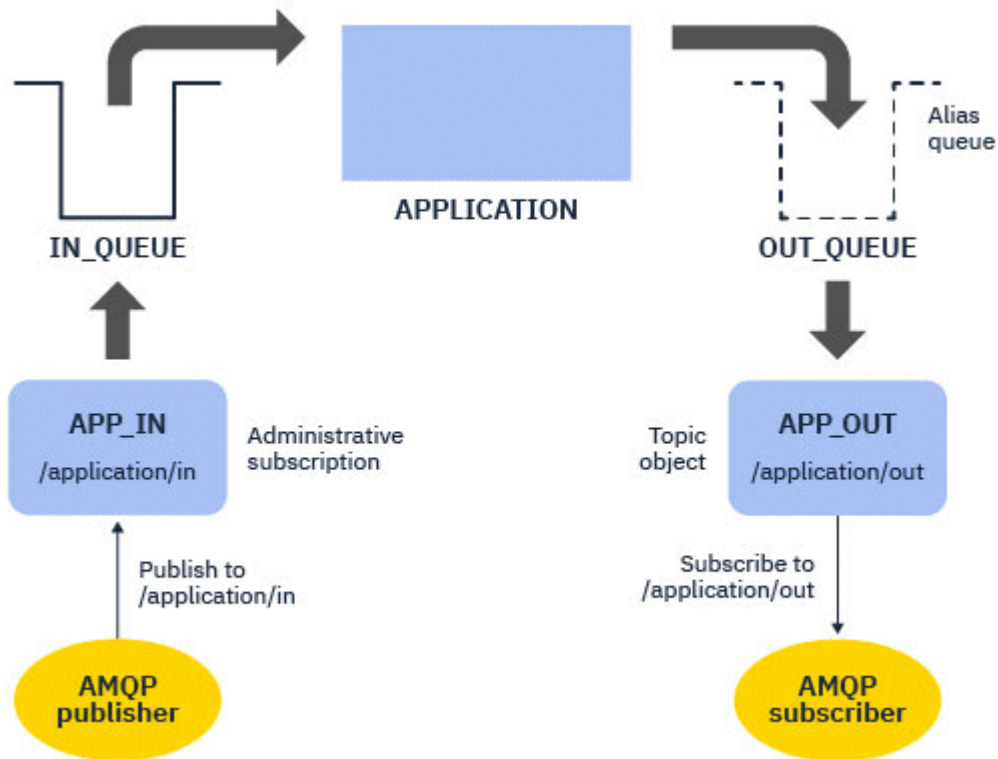
그런 다음 애플리케이션이 OUT_QUEUE에 넣은 메시지가 이 새 토픽 오브젝트로 전달되어 구독 클라이언트로 전송되는지 확인해야 합니다.

이 작업을 수행하려면 다음 MQSC 명령을 사용하여 방금 작성한 토픽 오브젝트의 대상 유형과 함께 기존 큐 OUT_QUEUE을(를) 동일한 이름의 알리어스 큐로 대체하십시오.

```
DEF QALIAS(OUT_QUEUE) TARGTYPE(TOPIC) TARGET(APP_OUT)
```

이제 애플리케이션이 OUT_QUEUE에 넣은 메시지는 큐가 선택될 때까지 대기하지 않습니다. 대신 이 알리어스 큐의 대상(즉, 새 토픽 오브젝트 APP_OUT)으로 전달됩니다.

이 토픽 오브젝트 /application/out(으)로 표시되는 토픽 문자열을 구독하는 AMQP 클라이언트는 알리어스 큐에서 이 토픽 오브젝트로 전송된 메시지를 수신합니다.



관련 태스크

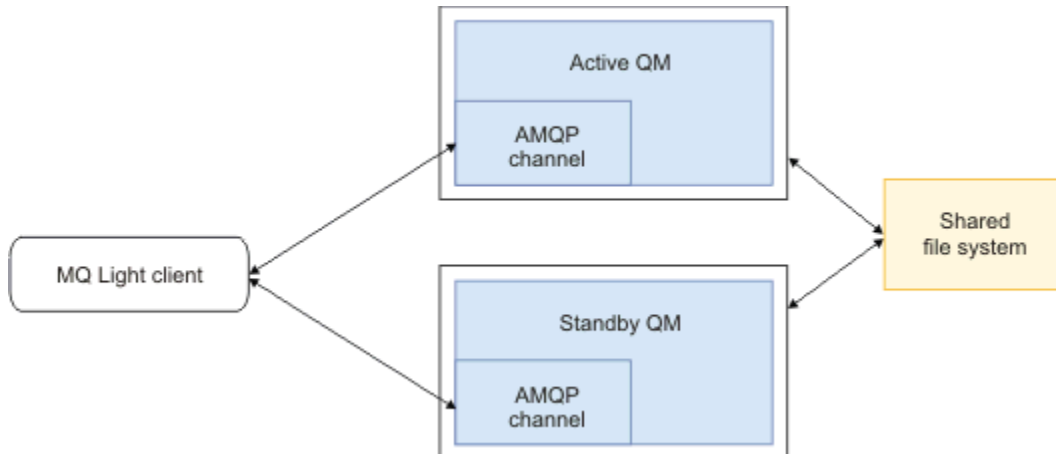
AMQP 채널 작성 및 사용

AMQP 클라이언트 보안 설정

ALW 고가용성을 위해 AMQP 클라이언트 구성

IBM MQ 다중 인스턴스 큐 관리자의 활성 인스턴스에 연결하고 고가용성(HA) 쌍에서 다중 인스턴스 큐 관리자의 대기 인스턴스로 장애 복구하도록 AMQP 1.0 애플리케이션을 구성할 수 있습니다. 이를 수행하기 위해 두 IP 주소 및 포트 쌍이 있는 AMQP 애플리케이션을 구성합니다.

클라이언트에서 서버에 대한 연결이 끊기면 호출되는 사용자 정의 함수를 사용하여 AMQP 클라이언트 API를 구성할 수 있습니다. 기능은 대체 IP 주소(예: 대기 IBM MQ 큐 관리자) 또는 원래 IP 주소에 연결할 수 있습니다. 기타 AMQP 클라이언트에 대해 클라이언트가 다중 연결 엔드포인트의 구성을 지원하는 경우 두 호스트 포트 쌍으로 애플리케이션을 구성하고 AMQP 라이브러리가 제공하는 재연결 기능을 사용하여 대기 큐 관리자로 전환하십시오.



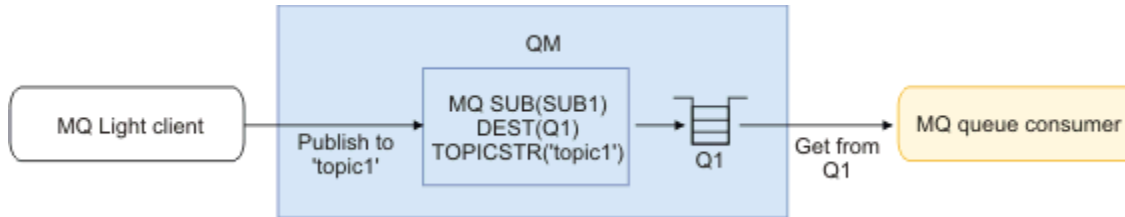
관련 태스크

AMQP 채널 작성 및 사용

▶ ALW AMQP 클라이언트에 대해 발행/구독 구성

AMQP 클라이언트는 기존 애플리케이션이 읽는 IBM MQ 큐에 메시지를 라우팅하는 IBM MQ가 있는 토픽에 발행할 수 있습니다. AMQP 1.0 애플리케이션이 큐에서 읽도록 구성된 기존 IBM MQ 애플리케이션에 메시지를 전송하게 하려면 큐 관리자에서 관리 IBM MQ 구독을 정의해야 합니다.

AMQP 애플리케이션이 사용하는 토픽 문자열과 일치하는 토픽 패턴을 사용하도록 구독을 구성하십시오. IBM MQ 애플리케이션이 메시지를 가져오거나 찾아보는 큐의 이름으로 구독 목적지를 설정하십시오.



관련 태스크

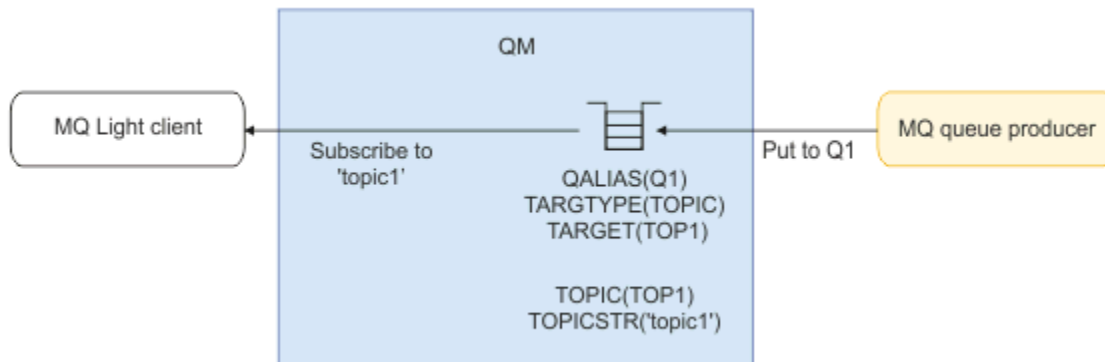
[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

▶ ALW IBM MQ 애플리케이션에서 메시지를 수신하기 위해 큐 알리언스를 사용하는 AMQP 클라이언트

AMQP 클라이언트는 토픽을 구독하고 IBM MQ 애플리케이션이 알리언스 큐에 넣는 메시지를 수신할 수 있습니다. AMQP 1.0 애플리케이션이 큐에 메시지를 넣도록 구성된 기존 IBM MQ 애플리케이션에서 메시지를 수신하게 하려면 큐 관리자에서 큐 알리언스(QALIAS)를 정의해야 합니다.

큐 알리언스에는 넣기를 위해 IBM MQ 애플리케이션이 여는 큐와 동일한 이름이 있어야 합니다. 큐 알리언스는 AMQP 애플리케이션이 구독하는 토픽 패턴과 일치하는 토픽 문자열이 있는 IBM MQ 토픽 오브젝트의 기본 오브젝트 및 TOPIC의 기본 유형을 지정해야 합니다.



관련 태스크

[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

▶ ALW 애플리케이션 서버에서 응답을 이용하고 요청을 제출하는 AMQP 클라이언트

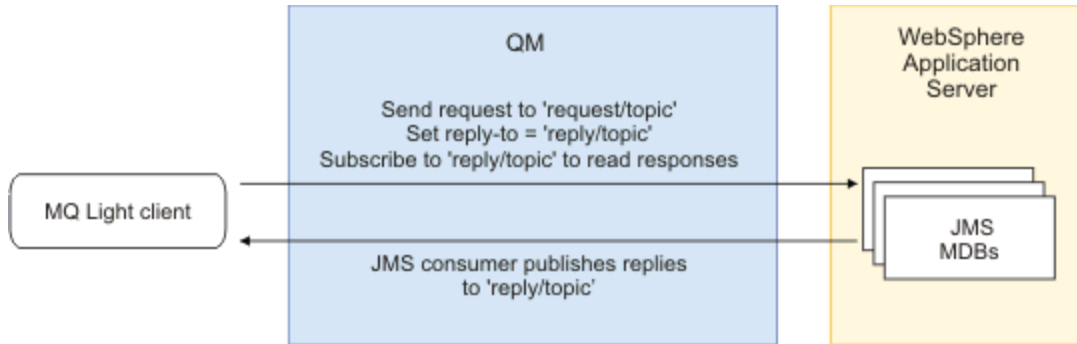
AMQP 클라이언트는 애플리케이션 서버에서 실행되는 메시지 구동 Bean에 요청을 제출하고 응답 토픽의 응답을 이용할 수 있습니다. IBM MQ는 IBM MQ가 발행하는 메시지에서 응답 대상 토픽을 설정하는 AMQP 1.0 애플리케이션을 지원합니다. AMQP 메시지가 응답 대상 속성 세트와 함께 발행되는 경우 응답 대상 필드의 값은 수신할 JMS 이용자의 JMS 특성으로 설정됩니다. 이 설정으로 JMS 이용자가 메시지에서 응답 대상 토픽을 읽고 AMQP 클라이언트에 응답 메시지를 다시 송신할 수 있습니다.

JMS 특성은 **JMSReplyTo**입니다. AMQP 응답 대상 문자열은 다음 유형 중 하나여야 합니다.

- 토픽 문자열입니다. 예: 'reply/topic'
- amqp://host:port/[topic-string] 양식의 AMQP 주소 URL. 예: amqp://localhost:5672/reply/topic

AMQP 주소 URL을 응답 대상 필드로 지정하면 **JMSReplyTo** 특성을 설정하기 전에 URL 끝에 있는 토픽 문자열을 제외한 모든 항목이 제거됩니다.

AMQP reply-to 주소에서 **JMSReplyTo** 속성으로의 매핑에 대한 자세한 내용은 다음을 참조하십시오. [649 페이지](#)의 『AMQP 필드를 IBM MQ 필드(수신되는 메시지)에 매핑』



관련 태스크

[AMQP 채널 작성 및 사용](#)

[AMQP 클라이언트 보안 설정](#)

ALW MQ Light 및 Apache Qpid JMS 애플리케이션 간의 상호 운용성

MQ Light 및 Apache Qpid JMS 애플리케이션은 동일한 방식으로 작업하며, 주제를 구독하는 경우 동일한 이름 지정 규칙을 따르는 IBM MQ 구독을 작성하십시오.

개인용 비공유 구독

애플리케이션에서 작성한 IBM MQ 구독의 이름은 :private:<clientid>:<topicstring>입니다.

다른 클라이언트 ID를 사용하는 애플리케이션은 다른 애플리케이션에서 작성한 구독에 액세스할 수 없습니다. 구독 이름이 자동으로 생성되며 AMQP 클라이언트 ID를 포함하기 때문입니다.

Apache Qpid JMS 및 MQ Light 애플리케이션은 개인용 구독에 대해 이 이름 지정 규칙을 사용합니다.

글로벌 공유 구독

AMQP 클라이언트가 작성한 글로벌로 공유하는 IBM MQ 구독의 이름은 :share:<sharename>:<topicstring>입니다.

다른 AMQP 클라이언트 ID 내 여러 애플리케이션이 동일한 공유 이름 및 토픽 문자열을 지정하는 경우 단일 구독을 공유하며, 함께 작업하여 해당 구독에 대한 메시지를 처리할 수 있습니다. 구독에서 메시지를 소모하는 작업자 애플리케이션 수를 조정하려는 경우 이 패턴을 사용할 수 있습니다.

Apache Qpid JMS 및 MQ Light 애플리케이션은 글로벌 공유 구독에 대해 이 이름 지정 규칙을 사용합니다. Apache Qpid JMS의 경우 JMS 연결에 지정된 클라이언트 ID가 없어야 합니다.

Apache Qpid JMS 라이브러리는 자동으로 AMQP 클라이언트 ID를 생성하지만, 이 클라이언트 ID는 IBM MQ 구독 이름 지정 목적으로 사용되지 않습니다.

참고: 글로벌 공유 구독은 여전히 개별 큐 관리자로 범위가 지정됩니다.

개인용 공유 구독

AMQP 클라이언트가 작성한 개인적으로 공유하는 IBM MQ 구독의 이름은 :privateshare:<clientid>:<sharename>:<topicstring>입니다.

단일 Apache Qpid JMS 애플리케이션의 여러 스레드가 동일한 공유 이름 및 토픽 문자열을 사용하고 JMS 연결에서 클라이언트 ID가 구성된 경우 해당 스레드는 동일한 IBM MQ 구독 오브젝트를 공유합니다.

그러나 다른 Apache Qpid JMS 연결은 구독을 공유할 수 없습니다. 다른 클라이언트 ID를 사용해야 하기 때문입니다.

MQ Light 클라이언트는 개인용 공유 구독의 개념을 지원하지 않으며, Apache Qpid JMS 애플리케이션에서 작성한 개인용 공유 구독에서 메시지를 이용할 수 없습니다.

IBM MQ JMS 구독

IBM MQ JMS 구독은 AMQP 채널과 다른 이름 지정 설계를 사용합니다. MQ Light 또는 Apache Qpid JMS 애플리케이션에서 IBM MQ JMS 애플리케이션과 구독을 공유할 수 없습니다.

관련 개념

AMQP 클라이언트 애플리케이션 개발

AMQP API에 대한 IBM MQ 지원을 통해 IBM MQ 관리자가 AMQP 채널을 작성할 수 있습니다. 시작되면 이 채널은 AMQP 클라이언트 애플리케이션에서 연결을 승인하는 포트 번호를 정의합니다.

ALW IBM MQ AMQP 리스너 제어 속성

멀티스레드 애플리케이션에서 더 나은 성능을 위해, AMQP 속성 파일에서 속성을 구성하여 AMQP 서비스가 사용해야 할 작업자 스레드 수를 조정할 수 있습니다.

다음 속성 파일에서 AMQP 리스너 서비스 속성을 구성할 수 있습니다:

- ▶ **Windows** 시스템에서 Windows: amqp_win.properties .
- ▶ **Linux** ▶ **AIX** 시스템에서 AIX and Linux: amqp_unix.properties .

구성할 수 있는 속성은 다음과 같습니다:

표 107. AMQP 리스너 서비스 속성	
특성	설명
com.ibm.mq.MQXR.Workers	AMQP 리스너 서비스가 생성하는 서버 작업자 스레드 수. 이 값이 지정되지 않으면 시스템의 논리 프로세서 수와 동일하게 기본값으로 설정됩니다.
MQIBindType	AMQP 서비스의 바인딩 유형으로, 다음 중 하나를 선택합니다: FASTPATH, SHARED 또는 ISOLATED. 기본값은 FASTPATH입니다.

AMQP 리스너 서비스는 여러 작업자 스레드에 걸쳐 클라이언트 연결 작업 부하를 분산합니다. AMQP 서비스가 사용해야 하는 작업자 스레드 수는 속성을 **com.ibm.mq.MQXR.Workers** 사용하여 지정할 수 있습니다.

큐 관리자 IBM MQ 관리자는 멀티스레드 애플리케이션에서 더 나은 성능을 위해 작업자 스레드 수를 조정할 수 있습니다. 일반적으로 작업자 스레드 수가 시스템의 논리 프로세서 수와 일치할 때 최상의 성능이 달성됩니다. 그러나 특정 시스템 구성 및 클라이언트 부하 특성에 따라 항상 그렇지 않을 수 있으므로, 작업자 스레드 수의 최적 값을 찾기 위해 일부 튜닝이 필요할 수 있습니다.

튜닝 전에 클라이언트 애플리케이션의 특성과 해당 애플리케이션의 워크로드를 철저히 이해해야 합니다. 다양한 스레드 수로 애플리케이션 성능을 측정하고 벤치마킹하면 작업자 스레드 수의 최적 값을 결정하는 데 도움이 될 것입니다.

참고: ▶ **MQ Appliance** 이러한 속성은 적용되지 않습니다 IBM MQ Appliance. 기본값이 사용됩니다 IBM MQ Appliance. ▶ **10.0.0**에서 IBM MQ 10.0.0, 이러한 속성을 에서 사용자 IBM MQ Appliance 정의할 수 있으며 기본값을 사용할 필요가 없습니다.

관련 참조

setamqp (AMQP 서버 속성 설정)

IBM MQ를 사용하여 REST 애플리케이션 개발

메시지를 송신 및 수신하기 위한 REST 애플리케이션을 개발할 수 있습니다. IBM MQ는 플랫폼 및 기능에 따라 다양한 REST API를 지원합니다.

다음 옵션은 IBM MQ와 메시지를 수신 및 전송하기 위해 선택할 수 있는 IBM MQ 지원 옵션입니다.

- [IBM MQ messaging REST API](#)
- [IBM z/OS Connect EE](#)
- [IBM Integration Bus](#)
- [DataPower](#)

IBM MQ messaging REST API

messaging REST API를 사용하여 일반 텍스트 형식으로 IBM MQ 메시지를 전송 및 수신하고 찾아볼 수 있습니다. 기본적으로 messaging REST API는 사용 가능합니다.

일반적인 메시지 특성을 설정하는 데 사용할 수 있는 다양한 HTTP 헤더가 지원됩니다.

messaging REST API는 IBM MQ 보안과 완전히 통합되어 있습니다. messaging REST API를 사용하려면 사용자는 mqweb 서버에 대해 인증되어야 하며 MQWebUser 역할의 구성원이어야 합니다.

추가 정보는 665 페이지의 『REST API를 사용한 메시징』의 내용을 참조하십시오. 또한 IBM Developer에서 학습서: [IBM MQ 메시징 REST API 시작하기](#)를 참조하십시오. 여기에는 메시징 REST API 사용에 대한 Go 및 Node.js 예제를 포함합니다.

IBM z/OS Connect EE

IBM z/OS Connect EE 기존 z/OS 자산(예: 트랜잭션 CICS, 큐 IMS IBM MQ 및 토픽) 위에 REST API를 구축할 수 있게 해주는 z/OS 제품입니다. 기존 z/OS 자산은 사용자에게 표시되지 않습니다. 따라서 자산 또는 자산을 사용하는 기존 애플리케이션을 변경하지 않고 REST에서 자산을 사용하도록 할 수 있습니다.

IBM z/OS Connect EE REST API에서 사용하는 JSON 데이터와 많은 메인프레임 애플리케이션에서 기대하는 전통적인 언어 구조(예: COBOL) 간 변환을 위한 자동 데이터 변환 기능을 제공합니다.

Eclipse 기반 IBM z/OS Connect EE API 툴킷은 쿼리 매개변수 및 URL 경로 세그먼트를 활용하여 포괄적인 RESTful API를 구축하고, IBM z/OS Connect EE 런타임을 통과하는 JSON 형식을 조작하는 데 사용될 수 있습니다.

IBM z/OS Connect EE 서비스 IBM MQ 공급자를 통해 큐와 토픽을 RESTful IBM MQ API로 노출하는 데 사용할 수 있습니다. 두 가지 다른 서비스 유형이 지원됩니다.

- 단방향 서비스: 여기에서는 단일 IBM MQ 조작을 큐 또는 토픽에서 수행할 수 있도록 하는 REST API를 제공합니다. HTTP 요청의 정확한 구성에 따라 메시지가 토픽에 발행되거나 큐에 전송될 수 있습니다. 또는 HTTP 요청으로 인해 큐에서 메시지가 파괴적인 방식으로 수신될 수 있습니다.
- 양방향 서비스: 여기에서는 백엔드 요청-응답 스타일의 애플리케이션에서 사용하는 큐 쌍 외에도 REST API를 제공합니다. 호출자는 양방향 서비스에 대한 HTTP 요청을 발행합니다. HTTP 요청 페이로드는 JSON에서 기존 언어 구조로 변환되며, 백엔드 애플리케이션에서 처리될 때 요청 및 응답 큐에 배치된 응답에 배치됩니다. 이 응답은 서비스에서 검색되고, 기존 언어 구조에서 JSON으로 변환되며, POST 응답 본문으로 호출자로 다시 전송됩니다.

자세한 내용은 문서의 [IBM z/OS Connect 를 애플리케이션에서 z/OS RESTful API 호출하기](#) 참조하십시오.

IBM MQ 서비스 제공자에 대한 자세한 정보는 [IBM MQ 서비스 제공자 사용](#)을 참조하십시오.

IBM Integration Bus

IBM Integration Bus IBM의 선도적인 통합 기술로, 지원하는 메시지 형식 및 프로토콜에 관계없이 애플리케이션과 시스템을 서로 연결하는 데 사용할 수 있습니다.

IBM Integration Bus 항상 지원해 IBM MQ 왔으며, *HTTPInput* 및 *HTTPRequest* IBM MQ노드를 제공하여 데이터베이스 등 다양한 시스템 위에 RESTful 인터페이스를 구축할 수 있습니다.

IBM Integration Bus는 IBM MQ 위에 간단한 REST 인터페이스를 제공하는 것 이상을 수행하는 데 사용될 수 있습니다. 해당 기능은 고급 페이로드(payload) 조작, 페이로드 인리치먼트 그리고 REST API의 일부인 향상된 기타 많은 기능을 제공할 때 사용할 수 있습니다.

자세한 내용은 XML 페이로드를 기대하는 애플리케이션 IBM MQ 위에 JSON을 REST 인터페이스로 노출하는 [기술 샘플](#)을 참조하십시오.

DataPower

DataPower 게이트웨이는 IBM MQ를 포함한 다양한 시스템에 보안, 제어, 통합 및 최적화된 액세스를 제공하는 데 도움이 되는 단일 다중 채널 게이트웨이입니다. 이는 하드웨어 및 가상 폼 팩터로 제공됩니다.

DataPower가 제공하는 서비스 중 하나는 하나의 프로토콜에서 입력을 받아 다른 프로토콜에서 출력을 생성할 수 있는 다중 프로토콜 게이트웨이입니다. 특히 DataPower는 HTTP(S) 데이터를 받아들이고 IBM MQ 위에 REST 인터페이스를 구축하는 데 사용할 수 있는 클라이언트 연결을 통해 IBM MQ로 라우트하도록 구성할 수 있습니다. 변형과 같은 다른 DataPower 서비스를 사용하여 REST 인터페이스를 향상시킬 수도 있습니다.

추가적인 정보는 [멀티프로토콜 게이트웨이](#)를 참조하십시오.

REST API를 사용한 메시징

를 사용하여 간단한 지점 messaging REST API 간 및 게시 메시징을 수행할 수 있습니다. 토픽에 메시지를 게시하고, 큐에 메시지를 전송하며, 큐의 메시지를 열람하고, 큐에서 메시지를 파괴적으로 가져올 수 있습니다. 정보는 일반 텍스트 형식으로 messaging REST API로 송수신됩니다.

시작하기 전에

참고:

- 기본적으로 messaging REST API는 사용 가능합니다. messaging REST API를 사용 안함으로 설정하여 모든 메시징을 방지할 수 있습니다. messaging REST API 사용 또는 사용 안함에 대한 자세한 정보는 [messaging REST API 구성](#)을 참조하십시오.
- messaging REST API는 IBM MQ 보안과 통합되어 있습니다. messaging REST API를 사용하려면 사용자는 mqweb 서버에 대해 인증되어야 하며 MQWebUser 역할의 구성원이어야 합니다. 사용자는 지정된 큐 또는 토픽에 접근할 수 있는 권한도 부여받아야 합니다. REST API의 보안에 대한 자세한 정보는 [IBM MQ Console](#) 및 [REST API 보안](#)을 참조하십시오.
- Advanced Message Security(AMS)을(를) messaging REST API과(와) 함께 사용하는 경우에는, 메시지를 게시하는 사용자의 컨텍스트가 아닌 mqweb 서버의 컨텍스트를 사용하여 모든 메시지가 암호화됨에 유의하십시오.
- 메시지를 수신하거나 열람할 때, 또는 JMS TextMessage 형식의 IBM MQMQSTR 메시지만 지원됩니다. 결과적으로 모든 메시지는 동기점에서 파괴적으로 수신되고 처리되지 않은 메시지는 큐에 남아 있습니다. 이러한 변조 메시지를 대체 대상으로 이동하도록 IBM MQ 큐를 구성할 수 있습니다. 추가 정보는 [224 페이지의 『IBM MQ classes for JMS에서 변조 메시지 핸들링』](#)의 내용을 참조하십시오.
- messaging REST API는 트랜잭션 지원에서 메시지의 단 한 번만의 전달을 제공하지는 않습니다. HTTP POST 요청이 발행되고 클라이언트가 HTTP 응답을 수신하기 전에 연결이 실패하면, 클라이언트는 메시지가 지정된 큐로 전송되었는지 또는 지정된 토픽에 게시되었는지 즉시 판단할 수 없습니다. HTTP DELETE가 실행되고 클라이언트가 HTTP 응답을 수신하기 전에 연결이 실패하는 경우에는 메시지를 큐에서 파괴적으로 가져온 후 메시지가 유실되었을 수 있는데, 파괴적 가져오기는 롤백할 수 없기 때문입니다.
- HTTP POST 작업에서 수신 문자열의 줄바꿈이 더 IBM MQ 9.3.0 이상 제거되지 않습니다. 이전 버전을 사용하는 REST 애플리케이션은 REST API를 통해 전송되거나 게시되는 메시지에 개행 문자를 사용해서는 안 됩니다. 이는 해당 문자가 손실될 수 있기 때문입니다.

프로시저

- [666 페이지의 『messaging REST API 시작하기』](#)
- [668 페이지의 『messaging REST API 사용』](#)
- [REST API 오류 핸들링](#)

- [REST API 검색](#)
- [REST API 자국어 지원\(NLS\)](#)

관련 참조

[메시징 REST API 참조](#)

관련 정보

[학습서: IBM MQ 메시징 REST API 시작하기](#)

messaging REST API 시작하기

messaging REST API를 빠르게 시작하고 cURL을 사용하여 일부 예제 명령을 실행해 봅니다.

시작하기 전에

messaging REST API 사용을 시작하기 위해 이 태스크의 예에는 다음 요구사항이 있습니다.

- 예에서는 cURL을 사용하여 REST 요청을 전송함으로써 큐에서 메시지를 넣고 가져옵니다. 따라서 이 태스크를 완료하려면 cURL이 시스템에 설치되어 있어야 합니다.
- 예제에서는 큐 관리자 QM1을(를) 사용합니다. 동일한 이름으로 큐 관리자를 작성하거나 시스템의 기존 큐 관리자를 대체하십시오. 큐 관리자는 mqweb 서버와 같은 시스템에 있어야 합니다.
- 이 태스크를 완료하려면 **dspmqweb** 명령을 사용할 수 있도록 특정 권한이 있는 사용자여야 합니다.
 -  **z/OS** z/OS의 경우 **dspmqweb** 명령을 실행하는 권한 및 mqwebuser.xml 파일에 대한 쓰기 액세스 권한이 있어야 합니다.
 -  **Multi** 다른 모든 운영 체제에서는 [권한이 있는 사용자](#)여야 합니다.
 -  **IBM i** IBM i, 날짜에 명령어는 QSHELL에서 실행되어야 합니다.

프로시저

1. mqweb 서버가 messaging REST API에 대해 구성되었는지 확인하십시오.

- IBM MQ Consolemqweb 서버가, administrative REST APIadministrative REST API , messaging REST APIMFT 또는 에서 사용될 수 있도록 설정되었는지 확인하십시오.기본 레지스트리를 사용하여 mqweb 서버를 구성하는 방법에 대한 자세한 내용은 “[mqweb 서버의 기본 구성](#)”을 참조하십시오.
- mqweb 서버가 이미 구성된 경우, [mqweb 서버 기본 구성](#)의 5단계에서 메시징을 활성화하기 위해 적절한 사용자를 추가했는지 확인하십시오.
 - mqweb 서버 구성에서 true 가 로 **mqRestMessagingAdoptWebUserContext** 설정된 경우, 의 사용자는 MQWebUser 역할의 messaging REST API 구성원이어야 합니다. MQWebAdmin 및 MQWebAdminRO 역할은 messaging REST API에 적용할 수 없습니다. 사용자는 또한 OAM 또는 RACF®을 통해 메시징에 사용되는 큐 및 토픽에 접근할 수 있도록 권한이 부여되어야 합니다.
 - mqweb 서버 구성에서 false 가 로 **mqRestMessagingAdoptWebUserContext** 설정된 경우, mqweb 서버를 시작하는 데 사용되는 사용자 ID는 OAM 또는 를 통해 메시징에 사용되는 큐에 대한 RACF 액세스 권한이 부여되어야 합니다.

2.  **z/OS**

z/OS에서 WLP_USER_DIR 환경 변수를 설정하여 명령을 **dspmqweb** 사용할 수 있도록 하십시오. 다음 명령을 입력하여 mqweb 서버 구성을 가리키도록 변수를 설정하십시오.

```
export WLP_USER_DIR=WLP_user_directory
```

crtmqweb여기서 **WLP_user_directory** 는 에 전달되는 디렉터리의 이름입니다. 예:

```
export WLP_USER_DIR=/var/mqm/web/installation1
```

자세한 정보는 [mqweb 서버](#)를 참조하십시오.

3. 다음 명령을 입력하여 `URL`을 REST API 확인하십시오:

```
dspmqweb status
```

다음 단계의 예에서는 REST API URL이 기본 URL `https://localhost:9443/ibmmq/rest/v3/(이)`라고 가정합니다. URL이 기본값과 다른 경우 다음 단계에서 URL을 대체하십시오.

4. 큐 관리자 QM1에서 MSGQ 큐를 작성하십시오. 이 큐는 메시징에 사용됩니다. 다음 방법 중 하나를 사용하십시오.

- administrative REST API의 mqsc 자원에 대한 POST 요청을 사용하여 mqadmin 사용자로 인증하십시오.

```
curl -k https://localhost:9443/ibmmq/rest/v3/admin/action/qmgr/QM1/mqsc -X POST -u mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/json" --data '{"type": "runCommandJSON", "command": "define", "qualifier": "qlocal", "name": "MSGQ"}'
```

- MQSC 명령을 사용하십시오.

 z/OS에서는 **runmqsc** 명령 대신에 2CR 소스를 사용하십시오. 자세한 내용은 [MQSC 및 PCF 명령을 실행할 수 있는 소스에 관한 IBM MQ for z/OS내용을](#) 참조하십시오.

- a. 다음 명령을 입력하여 큐 관리자에 대해 **runmqsc**를 시작하십시오.

```
runmqsc QM1
```

- b. **DEFINE QLOCAL** MQSC 명령을 사용하여 큐를 작성하십시오.

```
DEFINE QLOCAL(MSGQ)
```

- c. 다음 명령을 입력하여 **runmqsc**를 종료하십시오.

```
end
```

5. mqweb 서버 기본 구성의 5단계에서 mqwebuser.xml에 추가한 사용자에게 MSGQ 큐에 액세스할 권한을 부여하십시오. myuser이(가) 사용되는 사용자를 대체하십시오.

-  z/OS의 경우:

- a. 큐에 대한 사용자 액세스 권한을 부여하십시오.

```
RDEFINE MQQUEUE hlq.MSGQ UACC(NONE)
PERMIT hlq.MSGQ CLASS(MQQUEUE) ID(MYUSER) ACCESS(UPDATE)
```

- b. 큐의 모든 컨텍스트를 설정하기 위한 mqweb 시작 태스크 사용자 ID 액세스 권한을 부여하십시오.

```
RDEFINE MQADMIN hlq.CONTEXT.MSGQ UACC(NONE)
PERMIT hlq.CONTEXT.MSGQ CLASS(MQADMIN) ID(mqwebStartedTaskID) ACCESS(CONTROL)
```

-  기타 모든 운영 체제에서는 사용자가 mqm 그룹에 속하면 권한이 이미 부여되어 있습니다. 그렇지 않으면 다음 명령을 입력하십시오.

- a. 다음 명령을 입력하여 큐 관리자에 대해 **runmqsc**를 시작하십시오.

```
runmqsc QM1
```

- b. **SET AUTHREC** MQSC 명령을 사용하여 사용자에게 큐에서 찾아보기, 조회, 가져오기 및 넣기 권한을 부여하십시오.

```
SET AUTHREC PROFILE(MSGQ) OBJTYPE(QUEUE) +
PRINCIPAL(myuser) AUTHADD(BROWSE, INQ, GET, PUT)
```

- c. 다음 명령을 입력하여 **runmqsc**를 종료하십시오.

```
end
```

6. message 자원에서 POST 요청을 사용하여 QM1 큐 관리자의 MSGQ 큐에 Hello World! 콘텐츠가 있는 메시지를 두십시오. myuser 및 mypassword의 mqwebuser.xml에서 사용자 ID와 비밀번호를 대체하십시오.

기본 인증이 사용되며 임의의 값이 있는 ibm-mq-rest-csrf-token HTTP 헤더가 cURL REST 요청에 설정됩니다. 이 추가 헤더는 POST, PATCH 및 DELETE 요청에 필요합니다.

```
curl -k https://localhost:9443/ibmmq/rest/v3/messaging/qmgr/QM1/queue/MSGQ/message -X POST
-u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: text/
plain;charset=utf-8" --data "Hello World!"
```

7. message 자원의 DELETE 요청을 사용하여 QM1 큐 관리자의 MSGQ 큐에 있는 Hello World! 큐에서 파괴적으로 메시지를 가져옵니다. myuser 및 mypassword의 mqwebuser.xml에서 사용자 ID와 비밀번호를 대체하십시오.

```
curl -k https://localhost:9443/ibmmq/rest/v3/messaging/qmgr/QM1/queue/MSGQ/message -X DELETE
-u myuser:mypassword -H "ibm-mq-rest-csrf-token: value"
```

Hello World! 메시지가 리턴됩니다.

다음에 수행할 작업

- 예에서는 기본 인증을 사용하여 요청을 보냅니다. 토큰 기반 인증 또는 클라이언트 기반 인증을 대신 사용할 수 있습니다. 자세한 정보는 [REST API 및 IBM MQ Console로 클라이언트 인증서 인증 사용 및 REST API로 토큰 기반 인증 사용을 참조하십시오](#).
- messaging REST API 사용과 조회 매개변수를 사용한 URL 구성에 대해 자세히 알아보십시오. [668 페이지의 『messaging REST API 사용』](#).
- messaging REST API를 사용할 경우 성능을 최적화하기 위해 큐 관리자에 대한 연결이 풀링됩니다. 최대 풀 크기, 풀에 있는 모든 연결이 사용 중일 때 수행할 조치를 구성할 수 있습니다([messaging REST API 구성](#)).
- 사용 가능한 messaging REST API 자원과 사용 가능한 모든 조회 매개변수에 대한 참조 정보를 찾아보십시오. [messaging REST API 참조](#).
- 발견하세요 administrative REST API, RESTful 인터페이스를 통한 IBM MQ 관리: [를 REST API 사용한 관리](#).
- 브라우저 기반 GUI인 IBM MQ Console 검색: [IBM MQ Console을 사용한 관리](#).

messaging REST API 사용

messaging REST API를 사용하는 경우, URL에 대한 HTTP 메소드를 호출하여 IBM MQ 메시지를 송신 및 수신합니다. HTTP 메소드(예: POST)는 URL이 나타내는 오브젝트에서 수행될 조치의 유형을 나타냅니다. 조치에 대한 자세한 내용은 조회 매개변수에 인코딩될 수 있습니다. 조치 수행 결과에 대한 정보는 HTTP 응답의 본문으로 리턴될 수 있습니다.

시작하기 전에

messaging REST API를 사용하기 전에 다음 사항을 고려하십시오.

- messaging REST API를 사용하려면 mqweb 서버를 사용하여 인증해야 합니다. HTTP 기본 인증, 클라이언트 인증서 인증 또는 토큰 기반 인증을 사용하여 인증할 수 있습니다. 이러한 인증 방법 사용 방법에 대한 자세한 내용은 [REST API 보안을 IBM MQ Console 참조하십시오](#).
- REST API에서는 대소문자가 구분됩니다. 예를 들어, 큐 관리자를 qmgr1(이)라고 하는 경우 다음 URL의 HTTP POST를 수행한 결과 오류가 발생합니다.

```
/ibmmq/rest/v3/messaging/qmgr/QMGR1/queue/Q1/message
```

- 원격 큐 매니저에 연결할 때는 messaging REST API 큐 매니저 이름 대신 큐 매니저 연결에 고유한 이름을 사용해야 합니다.
- IBM MQ 오브젝트 이름에 사용할 수 있는 모든 문자를 URL로 직접 인코딩할 수 있는 것은 아닙니다. 이러한 문자를 올바르게 인코딩하려면 적절한 URL 인코딩을 사용해야 합니다.
 - 슬래시는 %2F(으)로 인코딩되어야 합니다.

- 백분율 기호는 %25(으)로 인코딩되어야 합니다.
- 마침표는 %2E(으)로 인코딩되어야 합니다.
- 물음표는 %3F(으)로 인코딩되어야 합니다.
- 메시지를 수신하거나 열람할 때, 및 JMS TextMessage 형식의 IBM MQMQSTR 메시지만 지원됩니다. 결과적으로 모든 메시지는 동기점에서 파괴적으로 수신되고 처리되지 않은 메시지는 큐에 남아 있습니다. 이러한 변조 메시지를 대체 대상으로 이동하도록 IBM MQ 큐를 구성할 수 있습니다. 추가 정보는 [224 페이지의 『IBM MQ classes for JMS에서 변조 메시지 핸들링』](#)의 내용을 참조하십시오.

이 태스크 정보

REST API를 사용하여 IBM MQ 큐 오브젝트에 대해 메시징 조치를 수행하는 경우, 먼저 해당 오브젝트를 나타내는 URL을 구성해야 합니다. 각 URL은 요청을 전송할 포트 또는 호스트 이름을 설명하는 접두부로 시작됩니다. URL의 나머지 부분은 특정 오브젝트 또는 자원으로 알려진 해당 오브젝트에 대한 라우트를 설명합니다.

자원에서 수행할 메시징 조치는 URL에 조회 매개변수가 필요한지 여부를 정의합니다. 또한 사용되는 HTTP 메소드와 추가 정보가 URL로 전송되는지 또는 URL에서 리턴되는지 여부를 정의합니다. 추가 정보가 HTTP 요청의 일부를 구성하거나 HTTP 응답의 일부로 리턴될 수 있습니다.

URL을 구성한 후 HTTP 요청을 IBM MQ에 보낼 수 있습니다. 선택한 프로그래밍 언어로 기본 제공되는 HTTP 구현을 사용하여 요청을 전송할 수 있습니다. cURL과 같은 명령행 도구, 웹 브라우저 또는 웹 브라우저 추가 기능을 사용하여 요청을 보낼 수도 있습니다.

중요사항: 최소한 [669 페이지의 『1.a』](#) 및 [669 페이지의 『1.b』](#) 단계를 수행해야 합니다.

프로시저

1. URL을 구성하십시오.

- a) 다음 명령을 입력하여 접두부 URL을 판별하십시오.

```
dspmweb status
```

사용할 URL에 `/ibmmq/rest/` 문구가 포함되어 있습니다.

- b) URL 경로에 메시징에 사용할 큐 및 연관된 큐 관리자 자원을 추가하십시오.

메시징 참조에서 변수 세그먼트는 URL에서 이를 둘러싼 중괄호({ })로 식별할 수 있습니다. 추가 정보는 [/messaging/qmgr/{qmgrName}/queue/{queueName}/message](#)를 참조하십시오.

예를 들어, 큐 관리자 `QM1`과 연관된 `Q1` 큐와 상호작용하려면 `/qmgr` 및 `/queue`을(를) 접두어 URL에 추가하여 다음 URL을 작성하십시오.

```
https://localhost:9443/ibmmq/rest/v3/messaging/qmgr/QM1/queue/Q1/message
```

팁: 대기열 관리자가 원격 대기열 관리자일 경우, 대기열 관리자 이름 대신 해당 대기열 관리자의 고유 이름을 사용해야 합니다. 원격 큐 관리자는 사용하기 전에 반드시 구성되어야 messaging REST API입니다. 자세한 정보는 [670 페이지의 『원격 큐 관리자를 설정하여 사용하기 위한 messaging REST API』](#)의 내용을 참조하십시오.

- c) 옵션: 선택적 조회 매개변수를 URL에 추가하십시오.

물음표, ?, 쿼리 매개변수, 등호 = 및 값을 URL에 추가하십시오.

예를 들어, 다음 메시지를 사용할 수 있을 때까지 최대 30초 동안 대기하려면 다음 URL을 작성하십시오.

```
https://localhost:9443/ibmmq/rest/v3/messaging/qmgr/QM1/queue/Q1/message?wait=30000
```

- d) 옵션: 선택적 추가 조회 매개변수를 URL에 추가하십시오.

앰퍼샌드(&)를 URL에 추가한 다음 [1c](#) 단계를 반복하십시오.

2. URL에서 관련 HTTP 메소드를 호출하십시오. 선택적인 메시지 페이로드를 지정하고 인증할 적절한 보안 신임 정보를 제공하십시오. 예를 들면, 다음과 같습니다.

- 선택한 프로그래밍 언어의 HTTP/REST 구현을 사용하십시오.

- REST 클라이언트 브라우저 추가 기능 또는 cURL과(와) 같은 도구를 사용하십시오.

원격 큐 관리자를 설정하여 사용하기 위한 messaging REST API

원격 큐 관리자에 연결하여 messaging REST API 메시지를 수행할 수 있습니다. 원격 큐 관리자에 연결하기 전에 원격 큐 관리자 구성을 설정해야 합니다. 그런 다음 구성 정보에 정의된 고유 이름을 사용하여 원격 큐 관리자에 연결할 수 있습니다.

시작하기 전에

- IBM MQ Consolemqweb 서버가, administrative REST APIadministrative REST API , messaging REST APIMFT 또는 에서 사용될 수 있도록 설정되었는지 확인하십시오.기본 레지스트리를 사용하여 mqweb 서버를 구성하는 방법에 대한 자세한 내용은 “mqweb 서버의 기본 구성”을 참조하십시오.
- mqweb 서버가 이미 구성된 경우, mqweb 서버 기본 구성의 5단계에서 메시지를 활성화하기 위해 적절한 사용자를 추가했는지 확인하십시오. 사용자는 MQWebUser 역할의 messaging REST API 구성원이어야 합니다. MQWebAdmin 및 MQWebAdminRO 역할은 messaging REST API에 적용할 수 없습니다.
- 원격 큐 관리자에 연결하도록 messaging REST API 구성되었는지 확인하십시오. 자세한 내용은 [연결 모드 구성을 messaging REST API참조하십시오](#).

이 태스크 정보

원격 큐 관리자에 연결하려면 다음을 사용하십시오 messaging REST API. 원격 큐 관리자는 다른 시스템의 큐 관리자, 다른 설치 환경의 큐 관리자 또는 mqweb 서버와 동일한 설치 환경의 큐 관리자일 수 있습니다.

원격 큐 관리자에 연결하려면 다음 구성 단계를 완료해야 합니다:

- 서버 연결 채널과 리스너를 구성합니다.
- 적절한 사용자에게 큐 관리자에 대한 접근 권한을 부여하십시오.
- 큐 관리자에 대한 연결 정보가 포함된 CCDT 파일을 생성하십시오.
- 연결 정보를 **setmqweb remote** 명령어를 사용하여 messaging REST API 에 추가하십시오.

그런 다음 리소스 URL 에서 큐 관리자 이름 대신 고유 이름을 제공하여 원격 큐 관리자를 사용할 수 있습니다.

원격 큐 관리자를 큐 관리자 그룹의 일부로 구성할 수도 있습니다. 자세한 정보는 [673 페이지의 『메시징 REST API와 함께 사용할 큐 관리자 그룹 설정』](#)의 내용을 참조하십시오.

프로시저

1. 원격 큐 관리자에 서버 연결 채널을 생성하여 큐 관리자에 대한 원격 연결을 허용합니다. 명령줄에서 MQSC **DEFINE CHANNEL** 명령을 사용하여 서버 연결 채널을 생성할 수 있습니다.
예를 들어, 큐 관리자 QM1 에 대한 서버 QM1.SVRCONN 연결 채널을 생성하려면 다음 명령을 입력하십시오:

```
DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

DEFINE CHANNEL 및 사용 가능한 옵션에 대한 자세한 내용은 [DEFINE CHANNEL](#)을 참조하십시오.

2. 적절한 사용자가 큐 관리자에 접근할 수 있도록 권한을 부여하십시오. 사용자는 큐 관리자에 대한, inquire, alternate user, 및 set contextconnect 권한이 필요합니다. 명령줄에서 제어 UNIX, Linux, and Windows**setmqaut** 명령을 사용하십시오. RACF 프로파일을 정의하여 승인된 사용자가 큐 z/OS관리자에 접근할 수 있도록 합니다.
예를 들어, 사용자가 큐 관리자 QM1 에 UNIX, Linux, and Windows 접근하도록 권한을 부여하려면 exampleUser 다음 명령을 입력하십시오:

```
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
```

3. 메시징에 사용하는 모든 큐 또는 토픽에 대해 사용자에게 set context 권한을 부여하십시오. 이를 위해 UNIX, Linux, and Windows `setmqaut control` 명령을 사용하여 `RACF` 를 z/OS설정하십시오.

예를 들어, UNIX, Linux, and Windows에서 메시징에 EXAMPLEQ 큐를 사용하는 경우, 다음 명령을 입력하여 권한을 exampleUser에 context 부여하십시오:

```
setmqaut -m QM1 -t queue -p exampleUser -n EXAMPLEQ +setall
```

- mqweb 서버 구성에서 false 가로 설정되었거나 설정될 **mqRestMessagingAdoptWebUserContext** 경우, 2단계에서 정의한 사용자는 메시징에 사용할 큐 및 토픽에 대한 접근 권한도 부여받아야 합니다. 가 설정되었거나 **mqRestMessagingAdoptWebUserContext** 설정될 경우 true, MQWebUser 역할을 가진 사용자는 메시징에 사용되는 큐 및 토픽에 대한 접근 권한을 부여받아야 합니다. 예에서 UNIX, Linux, and Windows, 이 작업은 **setmqaut** 명령어를 사용하여 수행할 수 있습니다. 사용자에게 올바른 권한을 부여하려면 RACFz/OS 프로필을 사용하십시오.

어떤 사용자에게 권한을 부여해야 하는지에 대한 자세한 내용은 을 참조하십시오 [675 페이지의 『사용된 보안 주체를 확인하는 messaging REST API』](#).

5. ALW

원격 큐 관리자에서 리스너가 존재하지 않으면, 명령줄에서 MQSC **DEFINE LISTENER** 명령을 사용하여 들어오는 네트워크 연결을 수락할 리스너를 생성하십시오.
예를 들어, 리모트 큐 관리자 QM1에 대해 포트 1414에서 REMOTE.LISTENER 리스너를 작성하려면 다음 명령을 입력하십시오.

```
runmqsc QM1
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
end
```

- 명령줄에서 MQSC **START LISTENER** 명령을 사용하여 리스너가 실행 중인지 확인하십시오:

ALW

예를 들어, AIX, Linux, and Windows에서 큐 관리자 QM1에 대한 REMOTE.LISTENER 리스너를 시작하려면 다음 명령을 입력하십시오.

```
runmqsc QM1
START LISTENER(REMOTE.LISTENER)
end
```

z/OS

예를 들어 z/OS에서 리스너를 시작하려면 다음 명령을 입력하십시오.

```
runmqsc QM1
START LISTENER TRPTYPE(TCP) PORT(1414)
end
```

채널 시작자 주소 공간은 시작하기 전에 시작해야 z/OS 합니다.

- mqweb 서버가 messaging REST API 호스팅되는 시스템에서 큐 관리자 연결 정보를 포함하는 JSON CCDT 파일을 생성하거나 업데이트하십시오.

CCDT 파일에는 name, clientConnection 및 type 정보가 포함되어야 합니다. 선택적으로 transmissionSecurity 정보와 같은 추가 정보를 포함할 수 있습니다. 모든 CCDT 채널 속성 정의에 대한 자세한 정보는 [CCDT 채널 속성 정의의 전체 목록](#)을 참조하십시오.

다음 예는 리모트 큐 관리자 연결을 위한 기본 JSON CCDT 파일을 보여줍니다. 이것은 채널 이름을 단계에서 생성된 예시 서버 연결 [670 페이지의 『1』](#) 채널과 동일한 이름으로 설정합니다. 연결 포트는 리스너가 사용하는 포트와 동일한 값으로 설정됩니다. 연결 호스트는 원격 큐 관리자(QM1)가 실행 중인 시스템의 호스트명으로 설정됩니다.

```
{
  "channel": [{
    "name": "QM1.SVRCONN",
    "clientConnection": {
      "connection": [{
        "host": "example.com",
        "port": 1414
      }],
      "queueManager": "QM1"
    },
    "type": "clientConnection"
  }]
}
```

8. mqweb 서버를 호스팅하는 설치 환경에서 명령을 **setmqweb remote** 사용하여 원격 큐 messaging REST API 관리자 정보를 mqweb 서버 구성에 추가하십시오.

최소한 다음 매개변수를 지정해야 합니다:

- **-qmgrName** 여기서 큐 관리자 이름을 지정합니다.
- **-ccdtURL** 여기서 큐 관리자에 대한 CCDT URL 를 지정합니다.
- **-uniqueName** 여기서 큐 관리자에 대한 고유한 이름을 지정합니다. 고유 이름은 동일한 이름을 가질 수 있는 원격 큐 관리자를 구별하는 데 사용되며, 따라서 다른 큐 관리자를 식별하기 위해 존재해서는 안 됩니다.

원격 큐 관리자 연결에 사용할 사용자 이름 및 암호, 신뢰 저장소 및 키 저장소 세부 정보 등 여러 가지 다른 옵션을 지정할 수 있습니다. **setmqweb remote** 명령어로 지정할 수 있는 매개변수의 전체 목록은 **setmqweb remote**를 참조하십시오.

예를 들어, 원격 큐 관리자 QM1 를 추가하려면 예제 CCDT 파일을 사용하여 다음 명령을 입력하십시오:

```
setmqweb remote add -uniqueName "RemoteQM1" -qmgrName "QM1" -ccdtURL "c:\myccdt\ccdt.json"
```

결과

원격 큐 관리자는 리소스 `URL` 내의 고유 이름을 큐 관리자 이름 대신 사용하여 `messaging REST API`와 함께 사용할 수 있습니다.

예

다음 예제는 큐 관리자 QM1 에 대한 원격 큐 관리자 연결을 설정합니다. **exampleUser** 사용자에게 부여된 권한에 따라 큐 관리자를 관리할 권한이 IBM MQ Console 부여됩니다. 이 사용자의 자격 증명은 **setmqweb remote** 사용하여 큐 관리자 연결을 구성할 때 IBM MQ Console 에 제공됩니다.

1. 원격 큐 관리자 QM1 가 있는 시스템에서 서버 연결 채널과 리스너가 생성됩니다. 리스너가 시작되고, 사용자가 큐 매니저에 연결하고 메시징에 **exampleUser** 사용되는 큐에 접근할 수 있도록 권한이 부여됩니다:

```
runmqsc QM1
#Define the server connection channel that will accept connections from the Console
DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
# Define the listener to use for the connection from the Console
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
# Start the listener
START LISTENER(REMOTE.LISTENER)
end

#Set mq authorization for exampleUser to access the queue manager and a queue for messaging
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +setall +dsp
setmqaut -m QM1 -t queue -p exampleUser -n EXAMPLEQ +put +get +browse +inq +setall
```

2. mqweb 서버가 실행 중인 시스템에 다음 연결 정보가 포함된 파일이 QM1_ccdt.json 생성됩니다:

```
{
  "channel": [{
    "name": "QM1.SVRCONN",
    "clientConnection": {
      "connection": [{
        "host": "example.com",
        "port": 1414
      }],
      "queueManager": "QM1"
    },
    "type": "clientConnection"
  }]
}
```

3. mqweb 서버가 실행 중인 시스템에서 큐 관리자 QM1 의 연결 정보가 mqweb 서버에 추가됩니다. 의 자격 증명은 연결 **exampleUser** 정보에 포함되어 있습니다:

```
setmqweb remote add -uniqueName "MACHINEAQM1" -qmgrName "QM1" -ccdtURL
"c:\myccdt\QM1_ccdt.json" -username "exampleUser" -password "password"
```

4. 리소스 `URL`에서 큐 관리자 이름 대신 큐 관리자 연결의 고유 이름을 사용하여 원격 큐 관리자 `QM1`에 연결할 수 messaging REST API 있습니다:

```
curl -k https://localhost:9443/ibmmq/rest/v3/messaging/qmgr/MACHINEAQM1/queue/EXAMPLEQ/  
message -X POST -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type:  
text/plain;charset=utf-8" --data "Hello World!"
```

메시징 REST API와 함께 사용할 큐 관리자 그룹 설정

메시징을 위해 큐 관리자 그룹에 연결하려면 messaging REST API 를 사용할 수 있습니다. 큐 관리자 그룹에 연결하기 전에, 해당 그룹에 대한 원격 큐 관리자 구성을 설정해야 합니다. 그런 다음 구성 정보에 정의된 고유 이름을 사용하여 큐 관리자 그룹에 연결할 수 있습니다.

시작하기 전에

- IBM MQ Consolemqweb 서버가, administrative REST APIadministrative REST API , messaging REST APIMFT 또는 에서 사용될 수 있도록 설정되었는지 확인하십시오. 기본 레지스트리를 사용하여 mqweb 서버를 구성하는 방법에 대한 자세한 내용은 “mqweb 서버의 기본 구성”을 참조하십시오.
- mqweb 서버가 이미 구성된 경우, mqweb 서버 기본 구성의 5단계에서 메시징을 활성화하기 위해 적절한 사용자를 추가했는지 확인하십시오. 사용자는 MQWebUser 역할의 messaging REST API 구성원이어야 합니다. MQWebAdmin 및 MQWebAdminRO 역할은 messaging REST API에 적용할 수 없습니다.
- 원격 큐 관리자에 연결하도록 가 messaging REST API 구성되었는지 확인하십시오. 자세한 내용은 다음을 참조하십시오. [연결 모드 설정 messaging REST API](#)

이 태스크 정보

큐 관리자 그룹을 사용하면 애플리케이션을 그룹 내의 모든 큐 관리자에 연결할 수 있습니다. 그룹은 클라이언트 채널 정의 테이블(CCDT) 내의 연결 집합으로 정의됩니다. MQCONN 또는 MQCONNX 호출을 사용할 때, 큐 관리자 이름 앞에 별표(*)를 붙여 그룹을 참조합니다. 큐 매니저 그룹과 연관된 고유 이름을 messaging REST API 사용하여 그룹을 참조합니다. 고유 이름은 리소스 `URL`에 큐 관리자 이름 대신 포함됩니다. 큐 관리자 그룹에 대한 자세한 내용은 을 참조하십시오 926 페이지의 『CCDT의 큐 관리자 그룹』.

원격 큐 관리자를 개별적으로 구성할 수도 있습니다. 자세한 정보는 670 페이지의 『원격 큐 관리자를 설정하여 사용하기 위한 messaging REST API』의 내용을 참조하십시오.

프로시저

1. 그룹 내 각 원격 큐 관리자에서 큐 관리자에 대한 원격 연결을 허용하기 위해 서버 연결 채널을 생성하십시오. 명령줄에서 MQSC **DEFINE CHANNEL** 명령을 사용하여 서버 연결 채널을 생성할 수 있습니다. 예를 들어, 큐 관리자 QM1에 대한 서버 QM1.SVRCONN 연결 채널을 생성하려면 다음 명령을 입력하십시오:

```
DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

DEFINE CHANNEL 및 사용 가능한 옵션에 대한 자세한 내용은 [DEFINE CHANNEL](#)을 참조하십시오.

2. 그룹 내 각 원격 큐 관리자에서 적절한 사용자가 큐 관리자에 접근할 수 있도록 권한이 부여되었는지 확인하십시오. 사용자는 큐 관리자에 대한, inquire, alternate user, 및 set contextconnect 권한이 필요합니다. 명령줄에서 제어 UNIX, Linux, and Windows **setmqaut** 명령을 사용하십시오. RACF 프로파일을 정의하여 승인된 사용자가 큐 z/OS관리자에 접근할 수 있도록 합니다. 예를 들어, 다음 명령어를 입력하여 UNIX, Linux, and Windows 사용자 `user exampleUser`가 큐 관리자 `QM1: `에 접근할 수 있도록 권한을 부여합니다

```
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
```

3. 메시징에 사용하는 그룹 내 큐 관리자에 정의된 모든 큐 또는 토픽에 대해 사용자에게 set context 권한을 부여하십시오. 이를 수행하려면 에서 제어 **setmqaut** 명령을 UNIX, Linux, and Windows 사용하고, 에서 z/OSRACF 을 사용하면 됩니다.

예를 들어 UNIX, Linux, and Windows, QM1 큐 관리자의 EXAMPLEQ 큐가 메시징에 사용 중인 경우, 다음 명령을 입력하여 권한을 exampleUserSet context 부여하십시오:

```
setmqaut -m QM1 -t queue -p exampleUser -n EXAMPLEQ +setall
```

4. mqweb 서버 구성에서 false 가로 설정되었거나 설정될 **mqRestMessagingAdoptWebUserContext** 경우, 2단계에서 정의된 사용자는 메시징에 사용할 그룹 내 큐 관리자에 정의된 큐 및 토픽에 대한 접근 권한도 부여받아야 합니다. 가 설정되었거나 **mqRestMessagingAdoptWebUserContext** 설정될 경우 true, MQWebUser 역할을 가진 사용자는 메시징에 사용되는 큐 및 토픽에 대한 접근 권한을 부여받아야 합니다. 예를 들어 UNIX, Linux, and Windows, 이 작업은 **setmqaut** 명령어를 사용하여 수행할 수 있습니다. 사용자에게 올바른 권한을 부여하려면 RACFz/OS 프로필을 사용하십시오.

어떤 사용자에게 권한을 부여해야 하는지에 대한 자세한 내용은 을 참조하십시오 [675 페이지의 『사용된 보안 주체를 확인하는 messaging REST API』](#).

5. ALW

그룹 내 각 원격 큐 관리자에서 리스너가 존재하지 않는 경우, 들어오는 네트워크 연결을 수락할 리스너를 생성하십시오. 명령줄에서 MQSC **DEFINE LISTENER** 명령을 사용하여 리스너를 생성할 수 있습니다. 예를 들어, 리모트 큐 관리자 QM1에 대해 포트 1414에서 REMOTE.LISTENER 리스너를 작성하려면 다음 명령을 입력하십시오.

```
runmqsc QM1
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
end
```

6. 그룹 내 각 원격 큐 관리자에서 명령줄의 MQSC **START LISTENER** 명령을 사용하여 리스너가 실행 중인지 확인하십시오.

ALW 예를 들어, AIX, Linux, and Windows에서 큐 관리자 QM1에 대한 REMOTE.LISTENER 리스너를 시작하려면 다음 명령을 입력하십시오.

```
runmqsc QM1
START LISTENER(REMOTE.LISTENER)
end
```

z/OS 예를 들어 z/OS에서 리스너를 시작하려면 다음 명령을 입력하십시오.

```
runmqsc QM1
START LISTENER TRPTYPE(TCP) PORT(1414)
end
```

채널 시작자 주소 공간은 시작하기 전에 시작해야 z/OS 합니다.

7. mqweb 서버가 messaging REST API 실행 중인 시스템에서 JSON CCDT 파일을 생성하십시오. 이 JSON 파일에는 그룹 내 각 큐 관리자에 대한 연결 정보가 포함되어 있습니다.

CCDT 파일에는 각 큐 관리자 연결에 대한, clientConnection, 및 type 정보가 반드시 포함되어야 name 합니다. 선택적으로 transmissionSecurity 정보와 같은 추가 정보를 포함할 수 있습니다. 모든 CCDT 채널 속성 정의에 대한 자세한 정보는 [CCDT 채널 속성 정의의 전체 목록](#)을 참조하십시오.

다음 예시는 두 개의 큐 관리자 연결을 위한 기본 JSON CCDT 파일을 보여줍니다. 첫 번째 연결은 큐 관리자 (QM1)용입니다. 서버 연결 채널을 보유하고 있으며 QM1.SVRCONN, 포트에 리스너가 설정되어 있고 1414, 호스트에서 QM1.example.com실행됩니다. 두 번째 연결은 큐 관리자 QM2 용입니다. 서버 연결 채널을 보유하고 있으며 QM2.SVRCONN, 포트에 리스너가 설정되어 있고 1415, 호스트에서 QM2.example.com실행됩니다. 그러나 두 연결 모두 연결 관리자 그룹 QMGRP의 일부이므로, 두 연결의 해당 **queueManager** 필드는 연결 관리자 그룹의 이름으로 설정됩니다.

```
{
  "channel": [{
    "name": "QM1.SVRCONN",
    "clientConnection": {
      "connection": [{
        "host": "QM1.example.com",
        "port": 1414
      }],
      "queueManager": "QMGRP"
    }
  ]
}
```

```

    },
    "type": "clientConnection"
  }],
  "channel": [{
    "name": "QM2.SVRCONN",
    "clientConnection": {
      "connection": [{
        "host": "QM2.example.com",
        "port": 1415
      }],
      "queueManager": "QMGRP"
    }
  }],
  "type": "clientConnection"
}

```

8. mqweb 서버를 호스팅하는 설치 환경에서 명령을 **setmqweb remote** 사용하여 큐 관리자 messaging REST API 그룹을 mqweb 서버 구성에 추가하십시오.

최소한 다음 매개변수를 지정해야 합니다:

- **-qmgrName** 여기서 큐 관리자 그룹의 그룹 이름을 지정합니다.
- **-ccdtURL** 여기서 큐 관리자들에 대한 CCDT URL 를 지정합니다.
- **-uniqueName** 여기서 큐 관리자 그룹을 식별하기 위한 고유한 이름을 지정합니다.
- **-group** 큐 관리자 정보를 그룹용으로 설정하기 위해.

연결에 사용할 사용자 이름과 비밀번호, 또는 트러스트스토어 및 키스토어의 세부 정보 등 여러 가지 다른 옵션을 지정할 수 있습니다. **setmqweb remote** 명령어로 지정할 수 있는 매개변수의 전체 목록은 **setmqweb remote**를 참조하십시오.

예를 들어, 예제 CCDT 파일을 사용하여 큐 관리자 그룹 QMGRP를 추가하려면 다음 명령을 입력하십시오:

```

setmqweb remote add -uniqueName "MyQMGRP" -qmgrName "QMGRP" -ccdtURL
"c:\myccdt\group_ccdt.json" -group

```

결과

리소스 URL 에서 고유 이름을 사용하여 messaging REST API 원격 큐 관리자 그룹을 사용할 수 있습니다. 그룹 내 큐 매니저 중 하나가 요청을 완료하도록 선택되며, 응답 헤더에는 해당 요청을 완료한 큐 **ibm-mq-resolved-qmgr** 매니저에 대한 정보가 반환됩니다.

사용된 보안 주체를 확인하는 messaging REST API

메시징을 위해 연결하려는 큐 관리자, 큐 및 토픽에 접근하려면 적절한 사용자에게 권한이 부여되어야 합니다 messaging REST API. 승인해야 할 사용자는 mqweb 서버의 구성 방식과 messaging REST API를 사용하여 원격 큐 관리자를 사용하는지 여부에 따라 달라집니다.

기본적으로 큐 관리자에 대한 액세스 권한을 부여하는 데 사용되는 보안 주체는 messaging REST API net 프레임워크를 실행하는 mqweb 서버를 시작하는 사용자입니다. 큐 및 토픽에 대한 액세스 권한을 부여하는 데 사용되는 보안 주체는 시스템에 로그인한 messaging REST API 사용자입니다. 그러나 mqweb 서버 또는 원격 큐 관리자 연결이 다른 보안 주체가 사용되도록 구성되었을 수 있습니다.

큐 매니저에 연결하는 데 사용되는 보안 주체 결정

로컬 큐 매니저 연결의 경우, 큐 매니저에 연결하는 데 사용되는 보안 주체는 messaging REST API net 애플리케이션을 실행하는 mqweb 서버를 시작하는 사용자입니다. 원격 큐 관리자 연결의 경우, 큐 관리자에 대한 액세스 권한을 부여하기 messaging REST API 위해 다음 보안 주체가 우선순위 순으로 사용됩니다. 즉, 원격 큐 관리자 구성 내에서 사용자가 여러 방식으로 지정된 경우, 목록에서 첫 번째 항목이 인증에 사용됩니다.

1. 보안 주체는 보안 종료 지점에서 채택된 사용자 컨텍스트입니다.
2. 보안 주체는 원격 큐 관리자에 연결하는 데 사용되는 서버 연결 채널의 CHLAUTH 규칙에서 채택된 사용자 컨텍스트입니다.
3. 보안 주체는 원격 큐 관리자 구성에 포함된 messaging REST API 사용자 ID입니다. 이 사용자 ID는 **setmqweb remote** 명령어로 큐 관리자를 추가할 때 큐 관리자 연결 정보에 선택적으로 포함됩니다.

4. 보안 주체는. messaging REST API를 실행하는 mqweb 서버를 시작하는 사용자입니다. 원격 큐 관리자를 설정하여 사용하기 위한 자세한 내용은 [messaging REST API 참조하십시오 670 페이지의 『원격 큐 관리자를 설정하여 사용하기 위한 messaging REST API』](#).

큐 및 토픽에 연결하는 데 사용되는 보안 주체 결정

mqweb 서버 구성에서 속성을 설정하여. messaging REST API를 프레임워크를 사용할 때 큐 및 토픽에 대한 연결을 인증하는 데 사용되는 보안 주체를 결정할 수 있습니다. 이 속성은

mqRestMessagingAdoptWebUserContext 속성입니다. 이 속성이 어떻게 설정되어 있는지 확인하려면 명령어를 **dspmweb properties** 사용하면 됩니다.

- 가 true로 **mqRestMessagingAdoptWebUserContext** 설정된 경우, 는 인증을 위해 messaging REST API에 로그인한 사용자의 messaging REST API 사용자 ID를 사용합니다. 따라서, mqweb 서버 구성에 존재하는 사용자 ID 또는 사용자 ID들은 큐 및 토픽에 접근하기 위해 messaging REST API 권한 부여가 필요한 보안 주체들입니다.
- 가 false로 **mqRestMessagingAdoptWebUserContext** 설정된 경우, 는 해당 를 호스팅하는 mqweb 서버를 시작한 messaging REST API 사용자의 사용자 ID를 인증에 사용합니다. 따라서, 해당 큐와 토픽을 호스팅하는 mqweb 서버를 시작하는 사용자 ID와 동일한 사용자 messaging REST API ID가 큐와 토픽에 대한 접근 권한을 부여받아야 합니다.

큐와 토픽이 원격 큐 관리자에 있는 경우, 인증에 사용되는 보안 주체는 큐 관리자 구성 설정에서 결정될 수 있습니다. 다음 보안 주체는 우선순위 순으로 사용될 수 있습니다:

1. 보안 주체는 보안 종료 지점에서 채택된 사용자 컨텍스트입니다.
2. 보안 주체는 원격 큐 관리자에 연결하는 데 사용되는 서버 연결 채널의 CHLAUTH 규칙에서 채택된 사용자 컨텍스트입니다. 예를 들어, 서버 연결 채널에서 CHLAUTH 규칙을 구성하여 MCAUSER 매개변수를 사용할 수 있습니다. 그런 다음 모든 연결은 큐 관리자를 사용할 권한이 있는 사용자 ID에 매핑됩니다.
3. 보안 주체는 큐 관리자의 AUTHINFO에서 채택된 사용자 컨텍스트입니다. 대기열 관리자의 CONNAUTH 속성이 참조하는 AUTHINFO 객체가 **ADOPTCTX(yes)** 사용하도록 구성된 경우, 대기열 관리자에 대한 연결을 인증하는 데 사용되는 보안 주체는 대기열 및 토픽에 대한 인증에도 사용됩니다. 예를 들어, 이 보안 주체는 명령어의 **setmqweb remote** 일부로 원격 큐 관리자 연결 정보에 포함된 사용자 ID일 수 있습니다.

관련 정보

[CHLAUTH](#)

[CONNAUTH](#)

[dspmweb 속성](#)

IBM MQ를 사용하여 MQI 애플리케이션 개발

IBM MQ에서는 C, Visual Basic, COBOL, 어셈블러, RPG, pTAL 및 PL/I에 대한 지원을 제공합니다. 이러한 절차적 언어는 메시지 큐 인터페이스(MQI)를 사용하여 메시지 큐잉 서비스에 액세스합니다.

애플리케이션을 선택한 언어로 작성하는 방법에 대한 자세한 정보는 하위 주제를 참조하십시오.

프로시저 언어의 호출 인터페이스에 대한 개요는 호출 설명을 참조하십시오. 이 주제에는 MQI 호출 목록이 포함되어 있으며, 각 호출은 각 언어로 호출을 코딩하는 방법을 표시합니다.

IBM MQ는 애플리케이션 작성에 도움이 되는 데이터 정의 파일을 제공합니다. 전체 설명은 [677 페이지의 『IBM MQ 데이터 정의 파일』](#)의 내용을 참조하십시오.

프로그램을 코딩하는 프로시저 언어를 선택하는 데 도움을 주기 위해서는, 프로그램에서 처리하는 최대 메시지 길이를 고려해야 합니다. 프로그램에서 알려진 최대 메시지 길이만 처리하는 경우 지원되는 언어로 코드화할 수 있습니다. 프로그램에서 처리하는 최대 메시지 길이를 알지 못하는 경우에는 CICS, IMS 또는 배치 애플리케이션을 작성하는지에 따라 선택하는 언어가 달라집니다.

IMS 및 배치

임의의 메모리 양을 확보하고 릴리스하기 위해 이러한 언어에서 제공하는 기능을 사용하려면 프로그램을 C, PL/I 또는 어셈블러 언어로 코드화하십시오. 또는 프로그램을 COBOL로 코드화할 수 있지만 스토리지를 가져오거나 해제하려면 어셈블러 언어, PL/I 또는 C 서브루틴을 사용하십시오.

CICS

CICS에서 지원하는 언어로 프로그램을 코드화하십시오. EXEC CICS 인터페이스는 필요한 경우 메모리 관리를 위한 호출을 제공합니다.

관련 개념

15 페이지의 『객체 지향 애플리케이션』

IBM MQ은(는) JMS, Java, C++ 및 .NET에 대한 지원을 제공합니다. 이러한 언어 및 프레임워크는 IBM MQ 오브젝트 모델을 사용하며 IBM MQ 호출 및 구조와 동일한 기능을 제공하는 클래스를 제공합니다.

기술 개요

6 페이지의 『애플리케이션 개발 개념』

사용자는 원하는 절차적 또는 객체 지향 언어를 사용하여 IBM MQ 애플리케이션을 작성할 수 있습니다. IBM MQ 애플리케이션을 디자인하고 작성하기 전에 기본 IBM MQ 개념을 숙지하십시오.

관련 참조

[애플리케이션 참조 개발](#)

IBM MQ 데이터 정의 파일

IBM MQ는 애플리케이션 작성에 도움이 되는 데이터 정의 파일을 제공합니다.

데이터 정의 파일은 다음과 같이 알려지기도 합니다.

언어	데이터 정의
C	포함 파일 또는 헤더 파일
Visual Basic	모듈 파일(32비트 버전 전용)
COBOL	복사 파일
어셈블러	매크로
PL/I	포함 파일

채널 엑시트를 작성하는 데 도움이 되는 데이터 정의 파일은 680 페이지의 『IBM MQ COPY, 헤더, 포함 및 모듈 파일』에 설명되어 있습니다.

설치 가능 서비스 엑시트를 작성하는 데 도움을 주는 데이터 정의 파일은 687 페이지의 『사용자 엑시트, API 엑시트 및 IBM MQ 설치 가능 서비스』에 설명되어 있습니다.

C++에서 지원되는 데이터 정의 파일은 [C++ 사용](#)의 내용을 참조하십시오.

IBM i

RPG에서 지원되는 데이터 정의 파일은 [IBM i 애플리케이션 프로그래밍 참조서\(ILE/RPG\)](#)의 내용을 참조하십시오.

접두부 CMQ 및 접미부가 있는 데이터 정의 파일의 이름은 프로그래밍 언어에 의해 판별됩니다.

접미부	언어
a	어셈블러 언어
b	Visual Basic
c	C
l	COBOL(초기화 값이 없음)
p	PL/I
v	COBOL(기본값이 설정됨)

설치 라이브러리

▶ **z/OS** 이름 **thlqual**은 z/OS에 있는 설치 라이브러리의 상위 레벨 규정자입니다.

이 주제에서는 다음 표제 아래 IBM MQ 데이터 정의 파일을 소개합니다.

- 678 페이지의 『C 언어 포함 파일』
- 678 페이지의 『Visual Basic 모듈 파일』
- 678 페이지의 『COBOL 복사 파일』
- ▶ **z/OS** 679 페이지의 『System/390 어셈블러 언어 매크로』
- ▶ **z/OS** 680 페이지의 『PL/I 포함 파일』

C 언어 포함 파일

C 포함 IBM MQ 파일은, 680 페이지의 『C 헤더 파일』에 나열되어 있습니다. 이러한 파일은 다음 디렉토리 또는 라이브러리에 설치됩니다.

플랫폼	설치 디렉토리 또는 라이브러리
▶ IBM i IBM i	QMQM/H
▶ Linux	MQ_INSTALLATION_PATH/inc/
▶ AIX AIX and Linux	
▶ Windows Windows	MQ_INSTALLATION_PATH\Tools\c\include
▶ z/OS z/OS	thlqual.SCSQC370

여기서 `MQ_INSTALLATION_PATH`은(는) IBM MQ이(가) 설치되어 있는 상위 레벨 디렉토리를 나타냅니다.

참고: AIX and Linux의 경우 포함 파일은 상징적으로 `/usr/include`에 링크됩니다.

디렉토리 구조에 대한 자세한 정보는 [파일 시스템 지원 계획](#)을 참조하십시오.

Visual Basic 모듈 파일

IBM MQ for Windows는 네 개의 Visual Basic 모듈 파일을 제공합니다.

이러한 파일은 685 페이지의 『Visual Basic 모듈 파일』에 나열되고 다음 경로에 설치됩니다.

```
MQ_INSTALLATION_PATH\Tools\Samples\VB\Include
```

COBOL 복사 파일

COBOL의 경우, IBM MQ는 이름 지정된 상수를 포함하는 별도의 복사 파일 및 각 구조에 대한 두 개의 복사 파일을 제공합니다.

초기값과 함께 또는 초기값 없이 각 구조가 제공되기 때문에 각 구조당 두 개의 복사 파일이 있습니다.

- COBOL 프로그램의 WORKING-STORAGE SECTION에서 구조 필드를 기본값으로 초기화하는 파일을 사용하십시오. 이러한 구조는 접미부에 V(Value)자가 표시되는 이름을 가지는 복사 파일에 정의됩니다.
- COBOL 프로그램의 LINKAGE SECTION에서 초기값이 없는 구조를 사용하십시오. 이러한 구조는 접미부에 L(Linkage)자가 표시되는 이름을 가진 복사 파일에 정의됩니다.

▶ **IBM i** IBM i에 대한 데이터와 인터페이스 정의를 포함하는 복사 파일이 MQI에 대한 프로토타입 호출을 사용하는 ILE COBOL 프로그램에 대해 제공됩니다. 이 파일은 접미부에 L(초기값이 없는 구조의 경우) 또는 접미부에 V(초기값이 있는 구조의 경우)가 표시되는 멤버 이름으로 QMQM/QCBLLESRC에 존재합니다.

COBOL 복사 파일 IBM MQ 목록은, 681 페이지의 『COBOL COPY 파일』에 나열되어 있습니다. 이러한 파일은 다음 디렉토리에 설치됩니다.

플랫폼	설치 디렉토리 또는 라이브러리
 Linux and Linux	<code>MQ_INSTALLATION_PATH/inc/</code>
 IBM i	<code>QM/QM/IBM/</code>
 Windows	<code>MQ_INSTALLATION_PATH\Tools\cobol\copybook</code> (Micro Focus COBOL의 경우) <code>MQ_INSTALLATION_PATH\Tools\cobol\copybook\VAcobol</code> (IBM VisualAge® COBOL의 경우)
 z/OS	<code>thlqual.SCSQCOBC</code>

`MQ_INSTALLATION_PATH`은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

필요한 파일만 프로그램에 포함시키십시오. 레벨-01 선언 후에 하나 이상의 COPY 명령문으로 이 작업을 수행하십시오. 이는 필요에 따라 여러 버전의 구조를 프로그램에 포함시킬 수 있음을 의미합니다. CMQV는 대형 파일이라는 점에 유의하십시오.

다음은 CMQMDV 사본 파일을 포함하는 COBOL 코드의 예입니다.

```
01 MQM-MESSAGE-DESCRIPTOR.
COPY CMQMDV.
```

각 구조 선언은 레벨-01 항목으로 시작하고 레벨-01 선언 코딩으로 구조의 여러 인스턴스를 선언할 수 있으며 나머지 구조 선언에서 복사를 위한 COPY 명령문이 뒤에 표시됩니다. 적절한 인스턴스를 참조하려면 IN 키워드를 사용하십시오.

다음은 CMQMDV의 두 인스턴스를 포함하는 COBOL 코드의 예입니다.

```
* Declare two instances of MQMD
01 MY-CMQMD.
COPY CMQMDV.
01 MY-OTHER-CMQMD.
COPY CMQMDV.
*
* Set MSGTYPE field in MY-OTHER-CMQMD
MOVE MQMT-REQUEST TO MQMD-MSGTYPE IN MY-OTHER-CMQMD.
```

구조를 4바이트 경계에 맞추십시오. COPY 명령문을 사용하여 레벨-01 항목이 아닌 항목 뒤에 구조를 포함시키는 경우 레벨-01 항목의 시작부터 이 구조가 다중 4바이트인지를 확인하십시오. 이를 수행하지 않으면 애플리케이션의 성능을 저하시킬 수 있습니다.

구조는 MQI에 사용된 데이터 유형에 설명되어 있습니다. 구조에 있는 필드의 설명은 접두부 없이 필드의 이름을 표시합니다. COBOL 선언에 표시된 대로 COBOL 프로그램에서 필드 이름에 구조의 이름을 접두부로 사용하고 그 다음에 하이픈을 사용하십시오. 구조 사본 파일의 필드는 다음 방식으로 접두부를 사용합니다.

구조 사본 파일의 선언에서 필드 이름은 대문자입니다. 대신 대소문자 혼용 또는 소문자를 사용할 수 있습니다. 예를 들어, MQGMO 구조의 필드 *StrucId*는 COBOL 선언에서 MQGMO-STRUCID로 표시되며 복사 파일에 있습니다.

V-접미부 구조는 모든 필드에 대해 초기값으로 선언되므로 필요한 값이 초기값과 다른 필드만 설정해야 합니다.

System/390 어셈블러 언어 매크로



IBM MQ for z/OS는 이름 지정된 상수를 포함하는 두 개의 어셈블러 언어 매크로와 각 구조를 생성하기 위한 하나의 매크로를 제공합니다.

이러한 매크로는 685 페이지의 『z/OS 어셈블러 COPY 파일』에 나열되고 **thlqual.SCSQMACS**에 설치됩니다.

이러한 매크로는 다음과 같은 코드를 사용하여 호출됩니다.

```
MY_MQMD CMQMDA EXPIRY=0,MSGTYPE=MQMT_DATAGRAM
```

PL/I 포함 파일

z/OS

IBM MQ for z/OS는 PL/I로 IBM MQ 애플리케이션을 작성할 때 필요한 모든 정의를 포함하는 포함 파일을 제공합니다.

이러한 파일은 682 페이지의 『PL/I 포함 파일』에 나열되고 **thlqual.SCSQPLIC** 디렉토리에 설치됩니다.

IBM MQ 스텝을 프로그램에 링크하려는 경우 이러한 파일을 프로그램에 포함시키십시오(957 페이지의 『프로그램을 실행할 준비 z/OS』 참조). IBM MQ 호출을 동적으로 링크하려는 경우에는 CMQP만 포함시키십시오(963 페이지의 『동적으로 IBM MQ 스텝 호출』 참조). 동적 링크를 배치 및 IMS 프로그램에만 수행할 수 있습니다.

IBM MQ COPY, 헤더, 포함 및 모듈 파일

다양한 프로그래밍 언어용 통합 컴파일러(MQI)에서 COPY, 헤더, 포함 및 모듈 파일을 사용하는 데 도움이 되는 정보입니다.

- 680 페이지의 『C 헤더 파일』
- 681 페이지의 『COBOL COPY 파일』
- **z/OS** 682 페이지의 『PL/I 포함 파일』
- **IBM i** 683 페이지의 『RPG 복사 파일』
- **Windows** 685 페이지의 『Visual Basic 모듈 파일』
- **z/OS** 685 페이지의 『z/OS 어셈블러 COPY 파일』

C 헤더 파일

이 헤더 파일들은 C++ 표준 라이브러리(MQI)를 사용하는 C 응용 프로그램을 작성하는 데 도움이 됩니다. C 헤더 파일은 다음 테이블에 요약되어 있습니다.

표 108. C 헤더 파일 - 호출 프로토타입, 데이터 유형, 리턴 코드, 상수 및 구조					
파일 이름	설명	IBM i	AIX and Linux 시스템	Windows	z/OS
호출 프로토타입, 데이터 유형, 리턴 코드, 상수 및 구조					
CMQC	MQI 정의	C	C	C	C
CMQBC	MQAI 정의	C	C	C	
CMQEC	인터페이스 시작점 정의(CMQC, CMQXC 및 CMQZC 포함)		C	C	
CMQCFC	PCF 정의	C	C	C	C
CMQPSC	발행/구독 정의	C	C	C	C
CMQXC	채널 및 엑시트 정의	C	C	C	C
CMQZC	설치 가능 서비스 정의	C	C	C	
키: C= 파일이 제공됨					

COBOL COPY 파일

이 COPY 파일들은 MQI 를 사용하는 COBOL 애플리케이션 프로그램을 작성하는 데 도움이 됩니다. COBOL 의 COPY 파일들은 다음 표에 요약되어 있습니다:

표 109. COBOL 복사 파일 - 리턴 코드, 상수 및 구조					
파일 이름	설명	IBM i	AIX and Linux	Windows	z/OS
리턴 코드 및 상수					
CMQx	MQI 정의	V	V	V	V
CMQCFx	PCF 정의	V	V	V	V
CMQPSx	발행/구독 정의	V	V	V	V
CMQXx	채널 및 엑시트 정의	V	V	V	V
구조					
CMQAIRx	MQAIR - 인증 정보 레코드		VL	VL	
CMQBOx	MQBO - 시작 옵션	VL	VL	VL	
CMQCDx	MQCD - 채널 정의	VL	VL	VL	VL
CMQCFBFx	MQCFBF - PCF 바이트 문자열 필터 매개변수	VL	VL	VL	VL
CMQCFBSx	MQCFBS - PCF 바이트 문자열 매개변수	VL	VL	VL	VL
CMQCFGRx	MQCFGR - PCF 그룹 매개변수	VL	VL	VL	VL
CMQCFHx	MQCFH - PCF 헤더	VL	VL	VL	VL
CMQCFIFx	MQCFIF - PCF 정수 필터 매개변수	VL	VL	VL	VL
CMQCFILx	MQCFIL - PCF 정수 목록 매개변수	VL	VL	VL	VL
CMQCFINx	MQCFIN - PCF 정수 매개변수	VL	VL	VL	VL
CMQCFSFx	MQCFSF - PCF 문자열 필터 매개변수	VL	VL	VL	VL
CMQCFSLx	MQCFSL - PCF 문자열 목록 매개변수	VL	VL	VL	VL
CMQCFSTx	MQCFST - PCF 문자열 매개변수	VL	VL	VL	VL
CMQCFXLx	MQCFIL64 - PCF 64비트 정수 목록 매개변수	VL	VL	VL	VL
CMQCFXNx	MQCFIN64 - PCF 64비트 정수 매개변수	VL	VL	VL	VL
CMQCHRVx	MQCHARV - 가변 길이 문자열	VL	VL	VL	VL
CMQCIHx	MQCIH - CICS bridge 헤더	VL	VL	VL	VL
CMQCNOx	MQCNO - 연결 옵션	VL	VL	VL	VL
CMQCSPx	MQCSP - 보안 매개변수	VL	VL	VL	VL
CMQCXPx	MQCXP - 채널 엑시트 매개변수	VL			VL
CMQDHx	MQDH - 분배 헤더	VL	VL	VL	VL
CMQDLHx	MQDLH - 데드-레터 헤더	VL	VL	VL	VL

표 109. COBOL 복사 파일 - 리턴 코드, 상수 및 구조 (계속)					
파일 이름	설명	IBM i	AIX and Linux	Windows	z/OS
CMQDXPx	MQDXP - 데이터 변환 엑시트 매개변수	V L		V L	
CMQEPHx	MQEPH - 임베드된 PCF 헤더	V L	V L	V L	V L
CMQGMox	MQGMO - 메시지 가져오기 옵션	V L	V L	V L	V L
CMQIIHx	MQIIH - IMS 정보 헤더	V L	V L	V L	V L
CMQMdx	MQMD - 메시지 디스크립터	V L	V L	V L	V L
CMQM1x	MQMD1 - 메시지 디스크립터 버전 1	V L	V L	V L	V L
CMQM2x	MQMD2 - 메시지 디스크립터 버전 2	V L	V L	V L	V L
CMQMDEx	MQMDE - 메시지 디스크립터 확장	V L	V L	V L	V L
CMQODx	MQOD - 오브젝트 디스크립터	V L	V L	V L	V L
CMQORx	MQOR - 오브젝트 레코드	V L	V L	V L	V L
CMQPMox	MQPMO - 메시지 넣기 옵션	V L	V L	V L	V L
CMQRFHx	MQRFH - 규칙 및 형식화 헤더	V L	V L	V L	V L
CMQRFH2x	MQRFH2 - 규칙 및 형식화 헤더 2	V L	V L	V L	V L
CMQRMHx	MQRMH - 참조 메시지 헤더	V L	V L	V L	V L
CMQRRx	MQRR - 응답 레코드	V L	V L	V L	
CMQSCox	MQSCO - TLS 구성 옵션		V L	V L	
CMQTMx	MQTM - 트리거 메시지	V L		V L	V L
CMQTMcx	MQTMC - 트리거 메시지 문자	V L	V L		
CMQTM2x	MQTMC2 - 트리거 메시지 2 문자	V L	V L	V L	V L
CMQWIHx	MQWIH - 작업 정보 헤더	V L	V L	V L	V L
CMQXQHx	MQXQH - 전송 큐 헤더	V L	V L	V L	V L
키: - 초기값이 제공된 파일, x=V - 초기값이 제공되지 않은 파일, x=L					

PL/I 포함 파일



이 포함 파일들은 `MQI`를 사용하는 `PL/I` 애플리케이션 프로그램을 작성하는 데 도움이 됩니다. 이러한 파일은 z/OS에서만 사용 가능합니다. 다음 표에 요약되어 있습니다:

표 110. PL/I 포함 파일 - 데이터 유형, 리턴 코드, 상수 및 구조					
파일 이름	설명	IBM i	AIX and Linux	Windows	z/OS
데이터 유형, 리턴 코드, 상수 및 구조					
CMQP	MQI 정의				P

표 110. PL/I 포함 파일 - 데이터 유형, 리턴 코드, 상수 및 구조 (계속)

파일 이름	설명	IBM i	AIX and Linux	Windows	z/OS
CMQCFP	PCF 정의				P
CMQEPP	시작점 정의				P
CMQPSP	발행/구독 정의				P
CMQXP	채널 및 엑시트 정의				P

키: P= 파일이 제공됨

RPG 복사 파일**IBM i**

이 예제 파일들은 `MQI`를 사용하는 RPG 애플리케이션 프로그램을 작성하는 데 도움이 됩니다. 이러한 파일은 IBM i에서만 사용 가능합니다. 다음 표에 요약되어 있습니다:

표 111. RPG 복사 파일 - 리턴 코드, 상수 및 구조

파일 이름	설명	IBM i	AIX and Linux	Windows	z/OS
리턴 코드 및 상수					
CMQx	MQI 정의	G R			
CMQCFx	PCF 정의	G			
CMQPSx	발행/구독 정의	G			
CMQXx	채널 및 엑시트 정의	G R			
구조					
CMQBOx	MQBO - 시작 옵션	G H			
CMQCDx	MQCD - 채널 정의	G H R			
CMQCFBFx	MQCFBF - PCF 바이트 문자열 필터 매개변수	G H			
CMQCFBSx	MQCFBS - PCF 바이트 문자열 매개변수	G H			
CMQCFGRx	MQCFGR - PCF 그룹 매개변수	G H			
CMQCFHx	MQCFH - PCF 헤더	G H			
CMQCFIFx	MQCFIF - PCF 정수 필터 매개변수	G H			
CMQCFILx	MQCFIL - PCF 정수 목록 매개변수	G H			
CMQCFINx	MQCFIN - PCF 정수 매개변수	G H			
CMQCFSFx	MQCFSF - PCF 문자열 필터 매개변수	G H			
CMQCFSLx	MQCFSL - PCF 문자열 목록 매개변수	G H			
CMQCFSTx	MQCFST - PCF 문자열 매개변수	G H			
CMQCFXLx	MQCFIL64 - PCF 64비트 정수 목록 매개변수	G H			

표 111. RPG 복사 파일 - 리턴 코드, 상수 및 구조 (계속)

파일 이름	설명	IBM i	AIX and Linux	Windows	z/OS
CMQCFXNx	MQCFIN64 - PCF 64비트 정수 매개변수	G H			
CMQCHARVx	MQCHARV - 가변 길이 문자열	G H			
CMQCIHx	MQCIH - CICS bridge 헤더	G H			
CMQCNOx	MQCNO - 연결 옵션	G H			
CMQCSPx	MQCSP - 보안 매개변수	G H			
CMQCXPx	MQCXP - 채널 엑시트 매개변수	G H R			
CMQDHx	MQDH - 분배 헤더	G H R			
CMQDLHx	MQDLH - 데드-레터 헤더	G H R			
CMQDXPx	MQDXP - 데이터 변환 엑시트 매개변수	G H R			
CMQEPHx	MQEPH - 임베드된 PCF 헤더	G H			
CMQGMOx	MQGMO - 메시지 가져오기 옵션	G H R			
CMQIIHx	MQIIH - IMS 정보 헤더	G H R			
CMQMDx	MQMD - 메시지 디스크립터	G H R			
CMQMD1x	MQMD1 - 메시지 디스크립터 버전 1	G H R			
CMQMD2x	MQMD2 - 메시지 디스크립터 버전 2	G H			
CMQMDEx	MQMDE - 메시지 디스크립터 확장	G H R			
CMQODx	MQOD - 오브젝트 디스크립터	G H R			
CMQORx	MQOR - 오브젝트 레코드	G H R			
CMQPMOx	MQPMO - 메시지 넣기 옵션	G H R			
CMQPXPx	MQPXP - 발행/구독 라우팅 엑시트 매개변수	G H			
CMQRFHx	MQRFH - 규칙 및 형식화 헤더	G H			
CMQRFH2x	MQRFH2 - 규칙 및 형식화 헤더 2	G H			
CMQRMHx	MQRMH - 참조 메시지 헤더	G H R			
CMQRRx	MQRR - 응답 레코드	G H R			
CMQTMx	MQTM - 트리거 메시지	G H R			
CMQTMCx	MQTMC - 트리거 메시지 문자	G H R			
CMQTM2x	MQTMC2 - 트리거 메시지 2 문자	G H R			
CMQWIHx	MQWIH - 작업 정보 헤더	G H			
CMQXQHx	MQXQH - 전송 큐 헤더	G H R			

표 111. RPG 복사 파일 - 리턴 코드, 상수 및 구조 (계속)					
파일 이름	설명	IBM i	AIX and Linux	Windows	z/OS
키: • 정적 링크용의 초기화되고 제공된 파일, x=G • 정적 링크용의 초기화되지 않고 제공된 파일, x=H • 동적 링크용의 초기화되고 제공된 파일, x=R					

Visual Basic 모듈 파일

Windows

이 헤더 파일이나 양식 파일은 .NET 프레임워크 (MQI)를 사용하는 Visual Basic 응용 프로그램을 작성하는 데 도움이 됩니다. 이러한 파일은 Windows에서만 사용 가능합니다. 다음 표에 요약되어 있습니다:

표 112. Visual Basic 모듈 파일 - 호출 선언, 데이터 유형, 리턴 코드, 상수 및 구조					
파일 이름	설명	IBM i	AIX and Linux 시스템	Windows	z/OS
호출 선언, 데이터 유형, 리턴 코드, 상수 및 구조					
CMQCB	MQI 정의			B	
CMQBB	MQAI 정의			B	
CMQCFB	PCF 정의			B	
CMQXCB	채널 및 엑시트 정의			B	
키: B= 파일이 제공됨					

z/OS 어셈블러 COPY 파일

z/OS

이 COPY 파일들은 MQI 를 사용하는 어셈블러 응용 프로그램을 작성하는 z/OS 데 도움이 됩니다. 이 파일들은 다음 표에 요약되어 있습니다:

표 113. z/OS 어셈블러 복사 파일 - 데이터 유형, 리턴 코드, 상수 및 구조					
파일 이름	설명	IBM i	AIX and Linux	Windows	z/OS
데이터 유형, 리턴 코드 및 상수					
CMQA	MQI 정의				A
CMQCFA	PCF 정의				A
CMQPSA	발행/구독 정의				A
CMQVERA	구조 버전 제어				A
CMQXA	채널 및 엑시트 정의				A
구조					
CMQCDA	MQCD - 채널 정의				
CMQCFBFA	MQCFBF - PCF 바이트 문자열 필터 매개변수				

표 113. z/OS 어셈블러 복사 파일 - 데이터 유형, 리턴 코드, 상수 및 구조 (계속)					
파일 이름	설명	IBM i	AIX and Linux	Windows	z/OS
CMQCFBSA	MQCFBS - PCF 바이트 문자열 매개변수				A
CMQCFGRA	MQCFGR - PCF 그룹 매개변수				A
CMQCFHA	MQCFH - PCF 헤더				A
CMQCFIFA	MQCFIF - PCF 정수 필터 매개변수				A
CMQCFILA	MQCFIL - PCF 정수 목록 매개변수				A
CMQCFINA	MQCFIN - PCF 정수 매개변수				A
CMQCFSA	MQCFSF - PCF 문자열 필터 매개변수				A
CMQCFSLA	MQCFSL - PCF 문자열 목록 매개변수				A
CMQCFSTA	MQCFST - PCF 문자열 매개변수				A
CMQCFXLA	MQCFIL64 - PCF 64비트 정수 목록 매개변수				A
CMQCFXNA	MQCFIN64 - PCF 64비트 정수 매개변수				A
CMQCHARVA	MQCHARV - 가변 길이 문자열				A
CMQCIHA	MQCIH - CICS bridge 헤더				A
CMQCNOA	MQCNO - 연결 옵션				A
CMQCSPA	MQCSP - 보안 매개변수				A
CMQCXPA	MQCXP - 채널 엑시트 매개변수				A
CMQDHA	MQDH - 분배 헤더				A
CMQDLHA	MQDLH - 데드-레터 헤더				A
CMQDXPA	MQDXP - 데이터 변환 엑시트 매개변수				A
CMQEPHA	MQEPH - 임베드된 PCF 헤더				A
CMQGMOA	MQGMO - 메시지 가져오기 옵션				A
CMQIIHA	MQIIH - IMS 정보 헤더				A
CMQMDA	MQMD - 메시지 디스크립터				A
CMQMD1A	MQMD1 - 메시지 디스크립터 버전 1				A
CMQMD2A	MQMD2 - 메시지 디스크립터 버전 2				A
CMQMDEA	MQMDE - 메시지 디스크립터 확장				A
CMQODA	MQOD - 오브젝트 디스크립터				A
CMQORA	MQOR - 오브젝트 레코드				A
CMQPMOA	MQPMO - 메시지 넣기 옵션				A
CMQRFHA	MQRFH - 규칙 및 형식화 헤더				A
CMQRFH2A	MQRFH2 - 규칙 및 형식화 헤더 2				A

표 113. z/OS 어셈블러 복사 파일 - 데이터 유형, 리턴 코드, 상수 및 구조 (계속)					
파일 이름	설명	IBM i	AIX and Linux	Windows	z/OS
CMQRMHA	MQRMH - 참조 메시지 헤더				A
CMQTMMA	MQTM - 트리거 메시지				A
CMQTMCA2A	MQTMCA2 - 트리거 메시지 2 문자				A
CMQWCRA	MQWCR - 클러스터 워크로드 클러스터 레코드				A
CMQWDRA	MQWDR - 클러스터 워크로드 목적지 레코드				A
CMQWDR1A	MQWDR1 - 클러스터 워크로드 목적지 레코드 버전 1				A
CMQWDR2A	MQWDR2 - 클러스터 워크로드 목적지 레코드 버전 2				A
CMQWIHA	MQWIH - 작업 정보 헤더				A
CMQWQRA	MQWQR - 클러스터 워크로드 큐 레코드				A
CMQWQR1A	MQWQR1 - 클러스터 워크로드 큐 레코드 버전 1				A
CMQWQR2A	MQWQR2 - 클러스터 워크로드 큐 레코드 버전 2				A
CMQWXPMA	MQWXP - 클러스터 워크로드 엑시트 매개변수				A
CMQWXP1A	MQWXP1 - 클러스터 워크로드 엑시트 매개변수 버전 1				A
CMQWXP2A	MQWXP2 - 클러스터 워크로드 엑시트 매개변수 버전 2				A
CMQWXP3A	MQWXP3 - 클러스터 워크로드 엑시트 매개변수 버전 3				A
CMQXPA	MQXP - CICS API 교차 엑시트 매개변수				A
CMQXQHA	MQXQH - 전송 큐 헤더				A
CMQXWDA	MQXWD - 엑시트 대기 디스크립터				A
키: A= 파일이 제공됨					

사용자 엑시트, API 엑시트 및 IBM MQ 설치 가능 서비스

이 주제에는 이러한 프로그램을 사용하고 개발하는 방법에 대한 정보 링크가 포함되어 있습니다.

사용자 엑시트, API 엑시트 및 설치 가능한 서비스를 사용하여 큐 관리자 기능을 확장하는 방법에 대한 소개는 [큐 관리자 기능 확장을 참조하십시오](#).

엑시트 및 설치 가능 서비스 작성 및 컴파일에 대한 정보는 하위 주제를 참조하십시오.

관련 개념

[MQI 채널에 대한 채널 엑시트 프로그램](#)

관련 참조

[API 엑시트 참조](#)

[설치 가능 서비스 인터페이스 참조 정보](#)

IBM i [IBM i에 설치 가능 서비스 인터페이스 참조 정보](#)

ALW **AIX, Linux, and Windows에 엑시트 및 설치 가능 서비스 작성**

라이브러리를 IBM MQ 링크하지 않고도 출력(exit)을 AIX, Linux, and Windows 작성하고 컴파일할 수 있습니다.

이 태스크 정보

이 토픽은 AIX, Linux, and Windows 시스템에만 적용됩니다. 다른 플랫폼의 엑시트 및 설치 가능 서비스를 작성하는 방법에 대한 세부사항은 관련 플랫폼 특정 주제를 참조하십시오.

IBM MQ가 기본이 아닌 위치에 설치된 경우에는 IBM MQ 라이브러리에 링크하지 않고 엑시트를 작성하고 컴파일해야 합니다.

다음 IBM MQ 라이브러리를 전혀 링크하지 않고도 시스템에서 AIX, Linux, and Windows 종료점을 작성하고 컴파일할 수 있습니다:

- mqmzf
- mqm
- mqmvx
- mqmvxd
- mqic
- mqutl

AIX and Linux 시스템에서 IBM MQ가 기본 위치에 설치되어 있으면 이 라이브러리에 링크된 기존 엑시트는 계속 작동합니다.

프로시저

1. cmqec.h 헤더 파일을 포함하십시오.

이 헤더 파일을 포함하면 cmqc.h, cmqxc.h 및 cmqzc.h 헤더 파일이 자동으로 포함됩니다.

2. MQIEP 구조를 통해 MQI 및 DCI가 호출되도록 엑시트를 작성하십시오. MQIEP 구조에 대한 자세한 정보는 [MQIEP 구조](#)를 참조하십시오.

- 설치 가능 서비스
 - **Hconfig** 매개변수를 사용하여 MQZEP 호출을 가리키십시오.
 - **Hconfig** 매개변수를 사용하기 전에 **Hconfig**의 처음 4바이트가 MQIEP 구조의 **StrucId**와 일치하는지 확인해야 합니다.
 - 설치 가능 서비스 컴포넌트 작성에 대한 자세한 정보는 [MQIEP](#)를 참조하십시오.
- API 엑시트
 - **Hconfig** 매개변수를 사용하여 MQXEP 호출을 가리키십시오.
 - **Hconfig** 매개변수를 사용하기 전에 **Hconfig**의 처음 4바이트가 MQIEP 구조의 **StrucId**와 일치하는지 확인해야 합니다.
 - API 엑시트 작성에 자세한 정보는 703 페이지의 [『API 엑시트 작성』](#)의 내용을 참조하십시오.
- 채널 엑시트
 - MQCXP 구조의 **pEntryPoints** 매개변수를 사용하여 MQI 및 DCI 호출을 가리키십시오.
 - **pEntryPoints**를 사용하기 전에 MQCXP 버전 번호가 버전 8 이상인지 확인해야 합니다.
 - 채널 엑시트 작성에 대한 자세한 정보는 712 페이지의 [『채널 엑시트 프로그램 작성』](#)의 내용을 참조하십시오.

- 데이터 변환 엑시트
 - MQDXP 구조의 **pEntryPoints** 매개변수를 사용하여 MQI 및 DCI 호출을 가리키십시오.
 - **pEntryPoints**를 사용하기 전에 MQDXP 버전 번호가 버전 2 이상인지 확인해야 합니다.
 - **crtmqcvx** 명령 및 amqsvfc0.c 소스 파일을 사용하면 **pEntryPoints** 매개변수를 사용하는 데이터 변환 코드를 작성할 수 있습니다. 735 페이지의 『IBM MQ for Windows의 데이터 변환 엑시트 작성』 및 733 페이지의 『IBM MQ for AIX or Linux 시스템에 대한 데이터 변환 엑시트 작성』의 내용을 참조하십시오.
 - **crtmqcvx** 명령을 사용하여 생성된 기존 데이터 변환 엑시트가 있는 경우 업데이트된 명령을 사용하여 엑시트를 다시 생성해야 합니다.
 - 데이터 변환 엑시트 작성에 대한 자세한 정보는 729 페이지의 『데이터 변환 엑시트 작성』의 내용을 참조하십시오.
- 사전 연결 엑시트
 - MQNXP 구조의 **pEntryPoints** 매개변수를 사용하여 MQI 및 DCI 호출을 가리키십시오.
 - **pEntryPoints**를 사용하기 전에 MQNXP 버전 번호가 버전 2 이상인지 확인해야 합니다.
 - 사전 연결 엑시트 작성에 대한 자세한 정보는 737 페이지의 『저장소의 사전 연결 엑시트를 사용하여 연결 정의 참조』의 내용을 참조하십시오.
- 발행 엑시트
 - MQPSXP 구조의 **pEntryPoints** 매개변수를 사용하여 MQI 및 DCI 호출을 가리키십시오.
 - **pEntryPoints**를 사용하기 전에 MQPSXP 버전 번호가 버전 2 이상인지 확인해야 합니다.
 - 발행 엑시트 작성에 대한 자세한 정보는 739 페이지의 『발행 엑시트 작성 및 컴파일』의 내용을 참조하십시오.
- 클러스터 워크로드 엑시트
 - MQWXP 구조의 **pEntryPoints** 매개변수를 사용하여 MQXCLWLN 호출을 가리키십시오.
 - **pEntryPoints**를 사용하기 전에 MQWXP 버전 번호가 버전 4 이상인지 확인해야 합니다.
 - 클러스터 워크로드 엑시트 작성에 대한 자세한 정보는 740 페이지의 『클러스터 워크로드 엑시트 작성 및 컴파일』의 내용을 참조하십시오.

예를 들어, 채널 엑시트에서 MQPUT 호출 방법은 다음과 같습니다.

```
pChannelExitParms -> pEntryPoints -> MQPUT_Call(pChannelExitParms -> Hconn,
                                                Hobj,
                                                &md,
                                                &pmo,
                                                messlen,
                                                buffer,
                                                &CompCode,
                                                &Reason);
```

추가적인 예시는 에서 확인할 수 971 페이지의 『멀티플랫폼용 샘플 절차적 프로그램 사용』 있습니다.

3. 다음과 같이 엑시트를 컴파일하십시오.

- IBM MQ 라이브러리에 링크하지 마십시오.
- 엑시트에 IBM MQ 라이브러리의 임베드된 RPath를 포함하지 마십시오.
- 엑시트 컴파일에 대한 자세한 정보는 다음 주제 중 하나를 참조하십시오.
 - API 엑시트: 705 페이지의 『API 엑시트 컴파일』
 - 채널 엑시트, 발행 엑시트, 클러스터 워크로드 엑시트: 728 페이지의 『AIX, Linux, and Windows 시스템에서 채널 엑시트 프로그램 컴파일』
 - 데이터 변환 엑시트: 729 페이지의 『데이터 변환 엑시트 작성』

4. 다음 위치 중 하나에 엑시트를 넣으십시오.

- 엑시트를 구성할 때 사용자가 선택한 완전한 경로

- 특정 설치 디렉토리에 있는 기본 엑시트 경로. 예를 들어, `MQ_DATA_PATH/exits/installation2`입니다.
- 기본 엑시트 경로

기본 엑시트 경로는 32비트 엑시트의 경우 `MQ_DATA_PATH/exits`이고 64비트 엑시트의 경우 `MQ_DATA_PATH/exits64`입니다. `qm.ini` 또는 `mqclient.ini` 파일에서 이 경로를 변경할 수 있습니다. 자세한 정보는 [엑시트 경로를 참조하십시오](#). Windows 및 Linux에서 IBM MQ Explorer를 사용하여 경로를 변경할 수 있습니다.

- 큐 관리자 이름을 마우스 오른쪽 단추로 클릭
- 특성...을 클릭하십시오.
- 엑시트를 클릭하십시오.
- 엑시트의 기본 경로 필드에서 엑시트 프로그램을 보유하는 디렉토리의 경로 이름을 지정하십시오.

엑시트가 특정 설치 디렉토리 및 기본 경로 디렉토리에 배치되면, 특정 설치 디렉토리 엑시트는 경로에 이름 지정된 IBM MQ를 설치하는 데 사용됩니다. 예를 들어, 엑시트는 `/exits/installation2` 및 `/exits`에 있지만 `/exits/installation1`에는 없습니다. IBM MQ 설치 `installation2`는 `/exits/installation2`의 엑시트를 사용합니다. IBM MQ 설치 `installation1`은 `/exits` 디렉토리의 엑시트를 사용합니다.

5. 필요한 경우 엑시트를 구성하십시오.

- 설치 가능 서비스: [697 페이지의 『서비스 및 컴포넌트 구성』](#)
- API 엑시트: [707 페이지의 『API 엑시트 구성』](#)
- 채널 엑시트: [729 페이지의 『채널 엑시트 구성』](#)
- 발행 엑시트: [740 페이지의 『발행 엑시트 구성』](#)
- 연결 전 엑시트: [클라이언트 구성 파일의 PreConnect 스탠자](#)

▶ ALW API 엑시트가 MQI 라이브러리와 링크되지 않음

특정 상황에서는 MQIEP 함수 포인터를 사용하도록 재코딩할 수 없는 기존 API 엑시트를 IBM MQ API 라이브러리와 링크해야 합니다.

이는 시스템의 런타임 링커를 사용하여 기존 API 엑시트를 함수 포인터가 아직 로드되지 않은 프로그램에 로드하는 데 필요합니다.

참고: 이 정보는 MQI 호출을 직접 작성하는 기존 API 엑시트로만 제한됩니다. 즉, MQIEP를 사용하지 않는 엑시트로만 제한됩니다. 가능한 경우 대신 MQIEP 시작점을 사용하도록 엑시트를 재코딩해야 합니다.

runmqsc MQI 라이브러리와 직접 링크되지 않는 프로그램의 예시입니다.

따라서 해당 필수 IBM MQ API 라이브러리와 링크되어 있지 않거나 MQIEP를 사용하도록 재코딩되지 않은 API 엑시트는 **runmqsc**로 로드되는 데 실패합니다.

큐 관리자 오류 로그에서 `undefined symbol: MQCONN`과 같은 규정 텍스트와 함께 `AMQ6175`: 시스템에서 공유 라이브러리를 동적으로 로드할 수 없음과 같은 오류를 볼 수 있습니다.

또한 `AMQ7214`: API 엑시트 'myexitname'에 대한 모듈을 로드할 수 없음과 같은 오류도 볼 수 있습니다.

관련 태스크

[688 페이지의 『AIX, Linux, and Windows에 엑시트 및 설치 가능 서비스 작성』](#)

라이브러리를 IBM MQ 링크하지 않고도 출력(exit)을 AIX, Linux, and Windows 작성하고 컴파일할 수 있습니다.

▶ ALW AIX, Linux, and Windows용 설치 가능 서비스 및 컴포넌트

이 절에서는 설치 가능 서비스 및 이와 연관된 기능과 컴포넌트를 소개합니다. 사용자 또는 소프트웨어 벤더가 컴포넌트를 제공할 수 있도록 이러한 기능에 대한 인터페이스를 문서화했습니다.

IBM MQ 설치 가능 서비스를 제공하기 위한 주된 이유는 다음과 같습니다.

- IBM MQ 제품에서 제공하는 컴포넌트의 사용 여부를 유연하게 선택할 수 있도록 하거나 해당 컴포넌트를 다른 컴포넌트로 바꾸거나 기능 보강합니다.
- IBM MQ 제품을 내부적으로 변경하지 않고 새 기술을 사용할 수 있는 컴포넌트를 제공하여 벤더를 참여시킵니다.
- IBM MQ가 새 기술을 빠르고 저렴하게 이용할 수 있도록 하여 제품을 좀 더 일찍 더 낮은 가격으로 제공합니다.

설치 가능 서비스 및 서비스 컴포넌트는 IBM MQ 제품 구조의 일부입니다. 이 구조의 가운데에는 MQI(Message Queue Interface)와 연관된 기능 및 규칙을 구현하는 큐 관리자의 부분이 있습니다. 이 중심 부분에서 작업을 수행하려면 설치 가능 서비스라는 여러 서비스 기능이 필요합니다. 설치 가능 서비스는 다음과 같습니다.

- 권한 서비스
- 이름 서비스

각 설치 가능 서비스는 하나 이상의 서비스 컴포넌트를 사용하여 구현된 관련 기능 세트입니다. 각 컴포넌트는 적절히 구조화되고 공용으로 사용 가능한 인터페이스를 사용하여 호출됩니다. 이 서비스를 사용하면, 독립된 소프트웨어 벤더 및 기타 써드파티가 IBM MQ 제품에서 제공하는 컴포넌트를 기능 보강하거나 바꾸도록 설치 가능 컴포넌트를 제공할 수 있습니다. 691 페이지의 표 114에서는 사용할 수 있는 서비스 및 컴포넌트를 요약합니다.

표 114. 설치 가능 서비스 컴포넌트 요약			
설치 가능 서비스	제공된 컴포넌트	Function	요구사항
권한 서비스	오브젝트 권한 관리자 (OAM, Object Authority Manager)	명령 및 MQI 호출에 대한 권한 검사를 제공합니다. 사용자는 OAM을 기능 보강하거나 바꾸기 위한 자체 컴포넌트를 작성할 수 있습니다. 예를 들어, 사용자 ID가 큐를 열 수 있는 권한이 있는지 검사합니다.	(적절한 플랫폼 권한 부여 기능이 가정됩니다.)
이름 서비스	없음	지정된 큐를 소유하는 큐 관리자의 이름을 검색하기 위한 지원을 큐 관리자에게 제공합니다. • 사용자 정의	• 써드파티 또는 사용자 작성 이름 관리자임

설치 가능 서비스 인터페이스는 [설치 가능 서비스 인터페이스 참조 정보](#)에 설명되어 있습니다.

관련 태스크

설치 가능 서비스 구성

서비스 컴포넌트 작성

이 절에서는 서비스, 컴포넌트, 시작점 및 리턴 코드의 관계에 대해 설명합니다.

기능 및 컴포넌트

각 서비스는 관련 기능 세트로 구성됩니다. 예를 들어, 이름 서비스에는 다음에 대한 기능이 포함됩니다.

- 큐 이름 검색 및 큐가 정의된 큐 관리자의 이름 리턴
- 큐 이름을 서비스의 디렉토리에 삽입
- 서비스의 디렉토리에서 큐 이름 삭제

또한 초기화 및 종료 기능도 포함됩니다.

설치 가능 서비스는 하나 이상의 서비스 컴포넌트에 의해 제공됩니다. 각 컴포넌트는 해당 서비스에 대해 정의된 일부 또는 모든 기능을 수행할 수 있습니다. 예를 들어, IBM MQ for AIX에서 제공된 권한 서비스 컴포넌트, OAM은 사용 가능한 모든 기능을 수행합니다. 자세한 정보는 694 페이지의 『권한 서비스 인터페이스』의 내용을 참조하십시오. 또한 컴포넌트는 서비스를 구현하는 데 필요한 모든 기본 자원 또는 소프트웨어(예: LDAP 디렉토

리) 관리를 담당합니다. 구성 파일은 컴포넌트를 로드하고 컴포넌트가 제공하는 기능적 루틴의 주소를 판별하는 표준 방식을 제공합니다.

692 페이지의 그림 72에서는 서비스와 컴포넌트가 관련되는 방식을 보여줍니다.

- 구성 파일의 스탠자로 큐 관리자에 대한 서비스가 정의됩니다.
- 각 서비스는 큐 관리자에서 제공된 코드로 지원됩니다. 사용자는 이 코드를 변경할 수 없으므로 고유 서비스를 작성할 수 없습니다.
- 각 서비스는 하나 이상의 컴포넌트에 의해 구현되며, 이러한 컴포넌트는 제품과 함께 제공되거나 사용자가 작성할 수 있습니다. 서비스의 여러 컴포넌트는 각각 서비스의 다른 기능을 지원하도록 호출될 수 있습니다.
- 시작점은 서비스 컴포넌트를 큐 관리자의 지원 코드에 연결합니다.

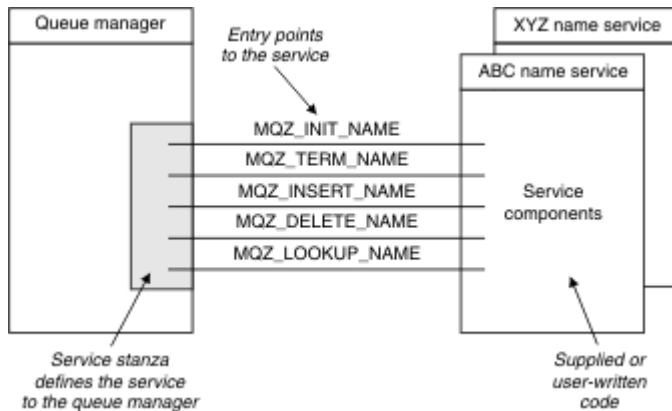


그림 72. 서비스, 컴포넌트 및 시작점 이해

시작점

각 서비스 컴포넌트는 특정 설치 가능 서비스를 지원하는 루틴의 시작점 주소 목록으로 표현됩니다. 설치 가능 서비스는 각 루틴에 의해 수행되는 기능을 정의합니다.

서비스 컴포넌트가 구성될 때의 정렬 기준은 서비스에 대한 요청을 만족시키려는 시도로 시작점이 호출되는 순서를 정의합니다.

제공된 헤더 파일 cmqzc.h에서 각 서비스에 대해 제공된 시작점에는 MQZID_ 접두부가 있습니다.

서비스가 존재하는 경우, 서비스는 사전정의된 순서로 로드됩니다. 다음 목록은 서비스와 서비스가 초기화되는 순서를 표시합니다.

1. NameService
2. AuthorizationService
3. UserIdentifierService

AuthorizationService는 기본적으로 구성되는 유일한 서비스입니다. NameService 및 UserIdentifierService를 사용하려면 수동으로 구성하십시오.

서비스 및 서비스 컴포넌트에는 일대일 또는 일대다 매핑이 있습니다. 각 서비스에 대해 여러 서비스 컴포넌트를 정의할 수 있습니다. AIX and Linux 시스템에서 ServiceComponent 스탠자의 Service 값은 qm.ini 파일에서 Service 스탠자의 Name 값과 일치해야 합니다. Windows에서 ServiceComponent의 Service 레지스트리 키 값은 Name 레지스트리 키 값과 일치해야 하고 다음과 같이 정의됩니다.

HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere

MQ\Installation\MQ_INSTALLATION_NAME\Configuration\QueueManager\qmname\. 여기서 qmname은 큐 관리자의 이름입니다.

AIX and Linux 시스템의 경우 서비스 컴포넌트는 qm.ini 파일에 정의된 순서대로 시작됩니다. Windows에서는 Windows 레지스트리가 사용되기 때문에, IBM MQ는 알파벳순으로 값을 리턴하는 RegEnumKey 호출을 발행합니다. 따라서 Windows에서 서비스는 레지스트리에 정의된 대로 알파벳순으로 호출됩니다.

`ServiceComponent` 정의의 정렬 기준은 중요합니다. 이 정렬 기준은 제공된 서비스에 대해 실행되는 컴포넌트의 순서를 좌우합니다. 예를 들어, Windows의 `AuthorizationService`는 `MQSeries.WindowsNT.auth.service`라는 기본 OAM 컴포넌트로 구성됩니다. 기본 OAM을 대체하기 위해 이 서비스에 대해 추가 컴포넌트를 정의할 수 있습니다. `MQCACF_SERVICE_COMPONENT`가 지정되지 않는 한 알파벳 순서로 첫 번째인 컴포넌트가 요청을 처리하는 데 사용되며 해당 컴포넌트의 이름이 사용됩니다.

리턴 코드

서비스 컴포넌트는 큐 관리자에 리턴 코드를 제공하여 다양한 조건에 대해 보고합니다. 서비스 컴포넌트는 조작의 성공 또는 실패를 보고하고 큐 관리자가 다음 서비스 컴포넌트로 진행하는지 여부를 표시합니다. 별도의 연속 매개변수가 이 표시를 전달합니다.

컴포넌트 데이터

단일 서비스 컴포넌트는 다양한 기능 간에 데이터를 공유해야 할 수도 있습니다. 설치 가능 서비스는 서비스 컴포넌트의 각 호출에 전달될 선택적 데이터 영역을 제공합니다. 이 데이터 영역은 서비스 컴포넌트만 사용할 수 있습니다. 각기 다른 주소 공간 또는 프로세스에서 작성되었더라도 특정 함수의 모든 호출에서 공유됩니다. 호출될 때마다 서비스 컴포넌트에서 주소 지정 가능하도록 보장됩니다. 이 영역의 크기를 `ServiceComponent` 스탠자에 선언해야 합니다.

컴포넌트의 초기화 및 종료

컴포넌트 초기화 및 종료 옵션의 용도입니다.

컴포넌트 초기화 루틴이 호출되면 컴포넌트에서 지원하는 각 시작점에 대해 큐 관리자 **MQZEP** 함수를 호출해야 합니다. **MQZEP**는 서비스의 시작점을 정의합니다. 정의되지 않은 모든 시작점은 `NULL`로 가정됩니다.

컴포넌트는 다른 방법으로 호출되기 전에 항상 1차 초기화 옵션으로 한 번 호출됩니다.

컴포넌트는 특정 플랫폼에서 2차 초기화 옵션으로 호출될 수 있습니다. 예를 들어, 서비스에 액세스한 각 운영 체제 프로세스, 스레드 또는 태스크에 대해 한 번 호출될 수 있습니다.

2차 초기화를 사용하는 경우:

- 컴포넌트는 2차 초기화를 위해 두 번 이상 호출될 수 있습니다. 이러한 각 호출에 대해 서비스가 더 이상 필요하지 않을 때는 2차 종료에 일치하는 호출이 발행됩니다.

이름 지정 서비스의 경우, 이것은 `MQZ_TERM_NAME` 호출입니다.

권한 서비스의 경우, 이것은 `MQZ_TERM_AUTHORITY` 호출입니다.

- 컴포넌트가 1차 및 2차 초기화를 위해 호출될 때마다 (`MQZEP`를 호출하여) 시작점을 재지정해야 합니다.
- 컴포넌트 데이터의 사본 하나만 컴포넌트에 사용되며 2차 초기화마다 다른 사본은 없습니다.
- 2차 초기화가 수행되기 전에는 (해당 경우에 따라 운영 체제 프로세스, 스레드 또는 태스크에서) 서비스의 다른 호출에 대해 컴포넌트가 호출되지 않습니다.
- 컴포넌트는 1차 및 2차 초기화에 대한 **Version** 매개변수를 동일한 값으로 설정해야 합니다.

컴포넌트는 더 이상 필요하지 않을 때에도 항상 1차 종료 옵션으로 한 번 호출됩니다. 이 컴포넌트에 대한 추가 호출은 수행되지 않습니다.

컴포넌트는 2차 초기화를 위해 호출된 경우 2차 종료 옵션으로 호출됩니다.

OAM(Object Authority Manager)

IBM MQ 제품과 함께 제공된 권한 서비스 컴포넌트는 오브젝트 권한 관리자(OAM)라고 합니다.

기본적으로 OAM은 활성 상태이며 제어 명령 **dspmqaout**(표시 권한), **dmpmqaut**(덤프 권한) 및 **setmqaut**(설정 또는 재설정 권한)로 작동합니다.

이러한 명령어의 구문과 사용 방법은 [제어 명령어를 사용한 IBM MQ for Multiplatforms 관리](#)에서 설명되어 있습니다.

OAM은 프린시펄 또는 그룹의 엔티티로 작동합니다:

- **Linux** **AIX** AIX and Linux 시스템에서 프린시펄은 사용자 ID 또는 자동으로 실행되는 애플리케이션 프로그램과 연관된 ID이며, 그룹은 시스템 정의된 프린시펄 컬렉션입니다.
- **Windows** Windows 시스템에서 프린시펄은 Windows 사용자 ID 또는 자동으로 실행되는 애플리케이션 프로그램과 연관된 ID이며, 그룹은 Windows 그룹입니다.

권한은 프린시펄 또는 그룹 레벨에서 부여되거나 취소될 수 있습니다.

MQI 요청을 작성하거나 명령을 실행할 때, OAM은 조작과 연관된 엔티티에 요청된 조작을 수행하고 지정된 큐 관리자 자원에 액세스할 수 있는 권한이 있는지 확인합니다.

권한 서비스를 사용하면 고유 권한 서비스 컴포넌트를 작성하여 큐 관리자에 제공된 권한 검사를 기능 보강하거나 바꿀 수 있습니다.

이름 서비스

이름 서비스는 지정된 큐를 소유하는 큐 관리자의 이름 검색에 필요한 지원을 큐 관리자에 제공하는 설치 가능 서비스입니다. 다른 큐 속성은 이름 서비스에서 검색할 수 없습니다.

이름 서비스를 사용하면 애플리케이션에서 로컬 큐인 것처럼 출력에 대한 리모트 큐를 열 수 있습니다. 이름 서비스는 큐 이외의 오브젝트에 대해서는 호출되지 않습니다.

참고: 리모트 큐에는 해당 **Scope** 속성이 CELL로 설정되어 있어야 합니다.

애플리케이션은 큐를 열 때 큐 관리자의 디렉토리에서 먼저 큐의 이름을 찾습니다. 해당 이름을 찾지 못한 경우, 큐 이름을 인식하는 이름 서비스를 찾을 때까지 구성된 수만큼 이름 서비스를 찾아봅니다. 이름을 인식하는 애플리케이션이 없으면 열기는 실패합니다.

이름 서비스는 해당 큐의 소유 큐 관리자를 리턴합니다. 그런 다음, 큐 관리자는 명령이 원래 요청에서 큐 및 큐 관리자 이름을 지정한 것처럼 MQOPEN 요청을 계속합니다.

이름 서비스 인터페이스(NSI)는 IBM MQ 프레임워크의 일부입니다.

이름 서비스 작동 방법

큐 정의가 **Scope** 속성을 큐 관리자(즉, MQSC의 SCOPE(QMGR))로 지정하는 경우, 큐 정의는 모든 큐 속성과 함께 큐 관리자의 디렉토리에만 저장됩니다. 이것을 설치 가능 서비스로 바꿀 수 없습니다.

큐 정의가 **Scope** 속성을 셀(즉, MQSC의 SCOPE(CELL))로 지정하는 경우, 큐 정의는 다시 모든 큐 속성과 함께 큐 관리자의 디렉토리에 저장됩니다. 그러나 큐와 큐 관리자 이름은 이름 서비스에도 저장됩니다. 이 정보를 저장할 수 있는 서비스가 사용 불가능한 경우, **Scope** 셀이 있는 큐를 정의할 수 없습니다.

정보가 저장되는 디렉토리는 서비스에서 관리할 수 있습니다. 또는 이 목적을 위해 서비스는 기본 서비스(예: LDAP 디렉토리)를 사용할 수 있습니다. 어느 경우에도 디렉토리에 저장된 정의는 컴포넌트 및 큐 관리자가 종료된 후에도 명시적으로 삭제될 때까지 지속되어야 합니다.

참고:

1. 이름 지정 디렉토리 셀 내의 다른 큐 관리자에서 원격 호스트의 로컬 큐 정의(CELL 범위에 해당)에 메시지를 보내려면 채널을 정의해야 합니다.
2. CELL 범위에 해당되는 경우에도 리모트 큐에서 직접 메시지를 가져올 수 없습니다.
3. CELL 범위의 큐로 송신할 때에는 리모트 큐 정의가 필요하지 않습니다.
4. 이름 지정 서비스는 목적지 큐를 중심으로 정의하지만, 목적지 큐 관리자에 대한 전송 큐와 한 쌍의 채널 정의가 여전히 필요합니다. 또한 로컬 시스템의 전송 큐는 대상 큐를 소유하는 큐 관리자와 이름이 같고 원격 시스템에서 셀 범위여야 합니다.

예를 들어, 리모트 큐 관리자의 이름이 QM01인 경우 로컬 시스템의 전송 큐의 이름도 QM01이어야 합니다.

권한 서비스 인터페이스

권한 서비스는 큐 관리자가 사용할 시작점을 제공합니다.

시작점은 다음과 같습니다.

MQZ_AUTHENTICATE_USER

사용자 ID 및 비밀번호를 인증하고 ID 컨텍스트 필드를 설정할 수 있습니다.

MQZ_CHECK_AUTHORITY

지정된 오브젝트에 대한 하나 이상의 조작을 수행할 수 있는 권한이 엔티티에 있는지 여부를 검사합니다.

MQZ_CHECK_PRIVILEGED

지정된 사용자가 권한 부여된 사용자인지 여부를 검사합니다.

MQZ_COPY_ALL_AUTHORITY

참조된 오브젝트에 대해 존재하는 모든 현재 권한을 다른 오브젝트에 복사합니다.

MQZ_DELETE_AUTHORITY

지정된 오브젝트와 연관된 모든 권한을 삭제합니다.

MQZ_ENUMERATE_AUTHORITY_DATA

지정된 선택 기준과 일치하는 모든 권한 데이터를 검색합니다.

MQZ_FREE_USER

연관된 할당 자원을 비웁니다.

MQZ_GET_AUTHORITY

엔티티가 지정된 오브젝트에 액세스하기 위해 갖는 권한을 가져옵니다.

MQZ_GET_EXPLICIT_AUTHORITY

이름 지정된 그룹이 지정된 오브젝트에 액세스하기 위해 갖는 권한(**nobody** 그룹의 추가 권한을 포함하지 않음) 또는 이름 지정된 프린시펄의 1차 그룹이 지정된 오브젝트에 액세스하기 위해 갖는 권한을 가져옵니다.

MQZ_INIT_AUTHORITY

권한 서비스 컴포넌트를 초기화합니다.

MQZ_INQUIRE

지원되는 권한 서비스 기능을 조회합니다.

MQZ_REFRESH_CACHE

모든 권한을 새로 고칩니다.

MQZ_SET_AUTHORITY

엔티티가 지정된 오브젝트에 대해 갖는 권한을 설정합니다.

MQZ_TERM_AUTHORITY

권한 서비스 컴포넌트를 종료합니다.

또한 IBM MQ for Windows의 권한 서비스는 큐 관리자에서 사용되는 다음 시작점을 제공합니다.

- **MQZ_CHECK_AUTHORITY_2**
- **MQZ_GET_AUTHORITY_2**
- **MQZ_GET_EXPLICIT_AUTHORITY_2**
- **MQZ_SET_AUTHORITY_2**

이러한 시작점은 Windows 보안 ID(NT SID)의 사용을 지원합니다.

이러한 이름은 컴포넌트 기능의 프로토타입을 작성하는 데 사용할 수 있는 헤더 파일 `cmqzc.h`에서 **typedef**로 정의됩니다.

초기화 함수(**MQZ_INIT_AUTHORITY**)는 컴포넌트의 기본 시작점이어야 합니다. 기타 함수는 초기화 함수가 컴포넌트 시작점 벡터에 추가한 시작점 주소를 통해 호출됩니다.

이름 서비스 인터페이스

이름 서비스는 큐 관리자가 사용할 시작점을 제공합니다.

다음 시작점이 제공됩니다.

MQZ_INIT_NAME

이름 서비스 컴포넌트를 초기화합니다.

MQZ_TERM_NAME

이름 서비스 컴포넌트를 종료합니다.

MQZ_LOOKUP_NAME

지정된 큐에 대한 큐 관리자 이름을 검색합니다.

MQZ_INSERT_NAME

지정한 큐에 대한 소유하고 있는 큐 관리자 이름이 포함된 입력 항목을 서비스에서 사용하는 디렉토리에 삽입합니다.

MQZ_DELETE_NAME

서비스에서 사용하는 디렉토리에서 지정된 큐의 입력 항목을 삭제합니다.

둘 이상의 이름 서비스가 구성된 경우:

- 검색의 경우, 큐 이름을 해석할 때까지 목록에 있는 각 서비스에 대해 MQZ_LOOKUP_NAME 함수가 호출됩니다(검색을 중지하도록 컴포넌트에서 표시하는 경우는 제외).
- 삽입의 경우, 이 기능을 지원하는 목록의 처음 서비스에 대해 MQZ_INSERT_NAME 함수가 호출됩니다.
- 삭제의 경우, 이 기능을 지원하는 목록의 첫 번째 서비스에 대해 MQZ_DELETE_NAME 함수가 호출됩니다.

삽입 및 삭제 기능을 지원하는 컴포넌트는 둘 이상이 되도록 하지 마십시오. 하지만 검색만을 지원하는 컴포넌트는 실행이 가능하며, 예를 들어 이를 목록의 마지막 컴포넌트로 사용하면 이름을 정의할 수 있는 큐 관리자에서 다른 이름 서비스 컴포넌트에 알려지지 않은 이름을 확인할 수 있습니다.

C 프로그래밍 언어에서 이름은 typedef 명령문을 사용하는 함수 데이터 유형으로 정의됩니다. 이러한 이름을 서비스 기능의 프로토타입을 작성하는 데 사용하여 매개변수가 올바른지 확인할 수 있습니다.

설치 가능 서비스에 특정한 모든 자료가 들어 있는 헤더 파일은 C 언어의 경우 cmqzc.h입니다.

컴포넌트의 기본 시작점이어야 하는 초기화 함수(MQZ_INIT_NAME)를 제외하고, MQZEP 호출을 사용하여 초기화 함수가 추가한 시작점 주소를 통해 함수가 호출됩니다.

다중 서비스 컴포넌트 사용

서비스의 컴포넌트를 둘 이상 설치할 수 있습니다. 이렇게 하면 컴포넌트가 서비스의 일부 구현만 제공하고 나머지는 다른 컴포넌트에서 지원됩니다.

다중 컴포넌트 사용의 예

ABC_name_serv 및 XYZ_name_serv라는 두 가지 이름 서비스 컴포넌트를 작성한다고 가정하십시오.

ABC_name_serv

이 컴포넌트는 서비스 디렉토리에서 이름 삽입이나 이름 삭제를 지원하지만, 큐 이름 검색은 지원하지 않습니다.

XYZ_name_serv

이 컴포넌트는 큐 이름 검색을 지원하지만, 서비스 디렉토리에서의 이름 삽입이나 이름 삭제는 지원하지 않습니다.

ABC_name_serv 컴포넌트는 큐 이름의 데이터베이스를 보유하고 두 가지 단순 알고리즘을 사용하여 서비스 디렉토리에서 이름을 삽입하거나 삭제합니다.

XYZ_name_serv 컴포넌트는 호출되는 큐 이름에 대해 고정된 큐 관리자 이름을 리턴하는 단순 알고리즘을 사용합니다. 큐 이름의 데이터베이스를 보유하지 않으므로 삽입 및 삭제 기능을 지원하지 않습니다.

컴포넌트는 동일한 큐 관리자에 설치됩니다. *ServiceComponent* 스탠자는 ABC_name_serv 컴포넌트가 먼저 호출되도록 정렬됩니다. 컴포넌트 디렉토리에서 큐를 삽입하거나 삭제하는 모든 호출은 ABC_name_serv 컴포넌트에 의해 핸들링됩니다. 이 컴포넌트는 이러한 기능을 구현하는 유일한 컴포넌트입니다. 그러나 ABC_name_serv 컴포넌트가 해결할 수 없는 검색 호출은 검색 전용 컴포넌트인 XYZ_name_serv로 전달됩니다. 이 컴포넌트는 해당 단순 알고리즘의 큐 관리자 이름을 제공합니다.

다중 컴포넌트 사용 시 시작점 생략

여러 컴포넌트를 사용하여 서비스를 제공하기로 결정하면, 특정 기능을 구현하지 않는 서비스 컴포넌트를 설계할 수 있습니다. 설치 가능 서비스 프레임워크는 생략할 수 있는 제한사항을 적용하지 않습니다. 그러나 특정 설치 가능 서비스의 경우, 하나 이상의 기능을 생략하면 서비스의 용도와 논리적으로 일치하지 않을 수 있습니다.

여러 컴포넌트에서 사용되는 시작점의 예

697 페이지의 표 115에서는 두 개의 컴포넌트가 설치된 설치 가능 이름 서비스의 예를 보여줍니다. 각 컴포넌트는 이러한 특정 설치 가능 서비스와 연관된 서로 다른 기능 세트를 지원합니다. 삽입 기능의 경우, ABC 컴포넌트 시작점이 처음으로 호출됩니다. (MQZEP를 사용하여) 서비스에 정의되지 않은 시작점은 NULL이라고 가정됩니다. 테이블에 초기화 시작점이 제공되어 있지만, 컴포넌트의 기본 시작점에서 초기화가 수행되므로 이 시작점은 필요하지 않습니다.

큐 관리자는 설치 가능 서비스를 사용해야 할 때 해당 서비스에 대해 정의된 시작점(697 페이지의 표 115의 열)을 사용합니다. 큐 관리자는 각 컴포넌트를 순서대로 사용하여 필수 기능을 구현하는 루틴의 주소를 판별합니다. 그런 다음 루틴이 존재하면 호출합니다. 조작이 성공하면, 큐 관리자는 모든 결과 및 상태 정보를 사용합니다.

표 115. 설치 가능 서비스 시작점의 예		
기능 번호	ABC 이름 서비스 컴포넌트	XYZ 이름 서비스 컴포넌트
MQZID_INIT_NAME(초기화)	ABC_initialize()	XYZ_initialize()
MQZID_TERM_NAME(종료)	ABC_terminate()	XYZ_terminate()
MQZID_INSERT_NAME(삽입)	ABC_Insert()	NULL
MQZID_DELETE_NAME(삭제)	ABC_Delete()	NULL
MQZID_LOOKUP_NAME(검색)	NULL	XYZ_Lookup()

루틴이 존재하지 않는 경우, 큐 관리자는 목록의 다음 컴포넌트에 대해 이 프로세스를 반복합니다. 또한 루틴이 존재하지만 조작을 수행할 수 없음을 표시하는 코드를 리턴하는 경우, 사용 가능한 다음 컴포넌트에 대해 프로세스를 계속합니다. 서비스 컴포넌트의 루틴은 조작을 수행하려는 추가 시도를 할 수 없음을 표시하는 코드를 리턴할 수 있습니다.

서비스 및 컴포넌트 구성

각 큐 관리자가 레지스트리에서 자체 스탠자를 가지고 있는 Windows 시스템을 제외하고, 큐 관리자 구성 파일을 사용하여 서비스 컴포넌트를 구성합니다.

프로시저

1. 큐 매니저 구성 파일(`/etc/sysvnt/queue-manager.qm.iniconf`)에 다음 구문을 추가하여 큐 매니저에서 서비스를 정의하고 모듈 위치를 지정하십시오:

- 사용된 각 서비스에는 큐 관리자에 대한 서비스를 정의하는 Service 스탠자가 있어야 합니다. 자세한 내용은 `qm.ini` 파일의 서비스 스탠자(stanza)를 참조하십시오.
- 서비스 내 각 컴포넌트의 경우 ServiceComponent 스탠자가 있어야 합니다. 이 구절은 해당 구성 요소의 코드를 포함하는 모듈의 이름과 경로를 식별합니다. 자세한 내용은 `qm.ini` 파일의 ServiceComponent 구절을 참조하십시오.

오브젝트 권한 관리자(OAM)라고 하는 권한 서비스 컴포넌트가 제품과 함께 제공됩니다. 큐 관리자 작성 시, 큐 관리자 구성 파일(또는 Windows 시스템의 레지스트리)은 권한 서비스 및 기본 컴포넌트(OAM)에 대해 적절한 스탠자를 포함하도록 자동으로 업데이트됩니다. 기타 컴포넌트의 경우에는 큐 관리자 구성 파일을 수동으로 구성해야 합니다.

각 서비스 컴포넌트의 코드는 큐 관리자를 시작할 때 동적 바인딩이 지원되는 플랫폼에서 동적 바인딩을 사용하여 큐 관리자 로드를 로드합니다.

2. 큐 관리자를 중지하고 재시작하여 컴포넌트를 활성화하십시오.

관련 참조

[qm.ini 파일의 Service 스탠자](#)

[ServiceComponent qm.ini 파일의 한 연](#)

사용자 권한 변경 후 OAM 새로 고치기

IBM MQ에서는 사용자의 권한 그룹 멤버십을 변경한 후 바로 OAM 권한 그룹 정보를 새로 고칠 수 있으므로, 큐 관리자를 중지 및 재시작하지 않아도 운영 체제 레벨에서 작성된 변경사항이 반영됩니다. 이 작업을 수행하려면 **REFRESH SECURITY** 명령을 실행하십시오.

참고: setmqaut 명령으로 권한 변경 시, OAM은 해당 변경사항을 즉시 구현합니다.

큐 관리자는 SYSTEM.AUTH.DATA.QUEUE라는 로컬 큐에 대한 권한 데이터를 저장합니다. 이 데이터는 **amqzfuma.exe**에 의해 관리됩니다.

관련 참조

[REFRESH SECURITY](#)

IBM i IBM i의 설치 가능 서비스 및 컴포넌트

이 정보를 사용하여 설치 가능 서비스 및 이와 연관된 기능과 컴포넌트에 대해 알아봅니다. 사용자 또는 소프트웨어 벤더가 컴포넌트를 제공할 수 있도록 이러한 기능에 대한 인터페이스를 문서화했습니다.

IBM MQ 설치 가능 서비스를 제공하기 위한 주된 이유는 다음과 같습니다.

- IBM MQ for IBM i에서 제공하는 컴포넌트의 사용 여부를 유연하게 선택할 수 있도록 하거나 해당 컴포넌트를 다른 컴포넌트로 바꾸거나 기능 보강합니다.
- IBM MQ for IBM i를 내부적으로 변경하지 않고 새 기술을 사용할 수 있는 컴포넌트를 제공하여 벤더를 참여시킵니다.
- IBM MQ가 새 기술을 빠르고 저렴하게 이용할 수 있도록 하여 제품을 좀 더 일찍 더 낮은 가격으로 제공합니다.

설치 가능 서비스 및 서비스 컴포넌트는 IBM MQ 제품 구조의 일부입니다. 이 구조의 가운데에는 MQI(Message Queue Interface)와 연관된 기능 및 규칙을 구현하는 큐 관리자의 부분이 있습니다. 이 중심 부분에서 작업을 수행하려면 설치 가능 서비스라는 여러 서비스 기능이 필요합니다. IBM MQ for IBM i에서 사용할 수 있는 설치 가능 서비스는 권한 서비스입니다.

각 설치 가능 서비스는 하나 이상의 서비스 컴포넌트를 사용하여 구현된 관련 기능 세트입니다. 각 컴포넌트는 적절히 구조화되고 공용으로 사용 가능한 인터페이스를 사용하여 호출됩니다. 이 서비스를 사용하면, 솔루션파트너(ISV) 및 기타 써드파티가 IBM MQ for IBM i에서 제공하는 컴포넌트를 기능 보강하거나 바꾸도록 설치 가능 컴포넌트를 제공할 수 있습니다. 698 페이지의 표 116은 권한 서비스의 지원을 요약합니다.

표 116. 권한 서비스 컴포넌트 요약		
제공된 컴포넌트	Function	요구사항
OAM(Object Authority Manager)	명령 및 MQI 호출에 대한 권한 검사를 제공합니다. 사용자는 OAM을 기능 보강하거나 바꾸기 위한 자체 컴포넌트를 작성할 수 있습니다.	(적절한 플랫폼 권한 부여 기능이 고정됩니다.)
DCE 이름 서비스 컴포넌트 참고: DCE는 V6.0 이전의 IBM MQ 버전에서만 지원됩니다.	• 큐 관리자가 큐를 공유할 수 있거나 • 사용자 정의 참고: 공유 큐는 Scope 속성이 CELL로 설정되어 있어야 합니다.	• 제공된 컴포넌트에 DCE가 필요하거나 • 써드파티 또는 사용자 작성 이름 관리자임

IBM i IBM i에서의 기능 및 컴포넌트

이 정보를 사용하면 IBM MQ for IBM i에서 사용할 수 있는 기능과 컴포넌트, 시작점, 리턴 코드 및 컴포넌트 데이터를 이해할 수 있습니다.

각 서비스는 관련 기능 세트로 구성됩니다. 예를 들어, 이름 서비스에는 다음에 대한 기능이 포함됩니다.

- 큐 이름 검색 및 큐가 정의된 큐 관리자의 이름 리턴
- 큐 이름을 서비스의 디렉토리에 삽입
- 서비스의 디렉토리에서 큐 이름 삭제

또한 초기화 및 종료 기능도 포함됩니다.

설치 가능 서비스는 하나 이상의 서비스 컴포넌트에 의해 제공됩니다. 각 컴포넌트는 해당 서비스에 대해 정의된 일부 또는 모든 기능을 수행할 수 있습니다. 또한 컴포넌트는 서비스를 구현하는 데 필요한 모든 기본 자원 또는 소프트웨어 관리를 담당합니다. 구성 파일은 컴포넌트를 로드하고 컴포넌트가 제공하는 기능적 루틴의 주소를 판별하는 표준 방식을 제공합니다.

서비스와 컴포넌트는 다음과 같이 관련됩니다.

- 구성 파일의 스탠자로 큐 관리자에 대한 서비스가 정의됩니다.
- 각 서비스는 큐 관리자에서 제공된 코드로 지원됩니다. 사용자는 이 코드를 변경할 수 없으므로 고유 서비스를 작성할 수 없습니다.
- 각 서비스는 하나 이상의 컴포넌트에 의해 구현되며, 이러한 컴포넌트는 제품과 함께 제공되거나 사용자가 작성할 수 있습니다. 서비스의 여러 컴포넌트는 각각 서비스의 다른 기능을 지원하도록 호출될 수 있습니다.
- 시작점은 서비스 컴포넌트를 큐 관리자의 지원 코드에 연결합니다.

시작점

각 서비스 컴포넌트는 특정 설치 가능 서비스를 지원하는 루틴의 시작점 주소 목록으로 표현됩니다. 설치 가능 서비스는 각 루틴에 의해 수행되는 기능을 정의합니다. 서비스 컴포넌트가 구성될 때의 정렬 기준은 서비스에 대한 요청을 만족시키려는 시도로 시작점이 호출되는 순서를 정의합니다. 제공된 헤더 파일 `cmqzc.h`에서 각 서비스에 대해 제공된 시작점에는 `MQZID_` 접두부가 있습니다.

리턴 코드

서비스 컴포넌트는 큐 관리자에 리턴 코드를 제공하여 다양한 조건에 대해 보고합니다. 서비스 컴포넌트는 조작의 성공 또는 실패를 보고하고 큐 관리자가 다음 서비스 컴포넌트로 진행하는지 여부를 표시합니다. 별도의 연속 매개변수가 이 표시를 전달합니다.

컴포넌트 데이터

단일 서비스 컴포넌트는 다양한 기능 간에 데이터를 공유해야 할 수도 있습니다. 설치 가능 서비스는 특정 서비스 컴포넌트의 각 호출에 전달될 선택적 데이터 영역을 제공합니다. 이 데이터 영역은 서비스 컴포넌트만 사용할 수 있습니다. 각기 다른 주소 공간 또는 프로세스에서 작성되었더라도 지정된 함수의 모든 호출에서 공유됩니다. 호출될 때마다 서비스 컴포넌트에서 주소 지정 가능하도록 보장됩니다. 이 영역의 크기를 `ServiceComponent` 스탠자에 선언해야 합니다.

IBM i IBM i의 초기화

컴포넌트 초기화 루틴이 호출되면 컴포넌트에서 지원하는 각 시작점에 대해 큐 관리자 `MQZEP` 함수를 호출해야 합니다. `MQZEP`는 서비스의 시작점을 정의합니다. 정의되지 않은 모든 시작점은 `NULL`로 가정됩니다.

1차 초기화

컴포넌트는 다른 방법으로 호출되기 전에 항상 이 옵션으로 한 번 호출됩니다.

2차 초기화

컴포넌트는 특정 플랫폼에서 이 옵션으로 호출될 수 있습니다. 예를 들어, 서비스에 액세스한 각 운영 체제 프로세스, 스레드 또는 태스크에 대해 한 번 호출될 수 있습니다.

2차 초기화를 사용하는 경우:

- 컴포넌트는 2차 초기화를 위해 두 번 이상 호출될 수 있습니다. 이러한 각 호출에 대해 서비스가 더 이상 필요하지 않을 때는 2차 종료에 일치하는 호출이 발행됩니다.
- 권한 서비스의 경우, 이것은 `MQZ_TERM_AUTHORITY` 호출입니다.
- 컴포넌트가 1차 및 2차 초기화를 위해 호출될 때마다 (`MQZEP`를 호출하여) 시작점을 재지정해야 합니다.
- 컴포넌트 데이터의 사본 하나만 컴포넌트에 사용되며 2차 초기화마다 다른 사본은 없습니다.
- 2차 초기화가 수행되기 전에는 (해당 경우에 따라 운영 체제 프로세스, 스레드 또는 태스크에서) 서비스의 다른 호출에 대해 컴포넌트가 호출되지 않습니다.
- 컴포넌트는 1차 및 2차 초기화에 대한 **Version** 매개변수를 동일한 값으로 설정해야 합니다.

1차 종료

컴포넌트는 더 이상 필요하지 않을 때에도 항상 이 옵션으로 한 번 시작됩니다. 이 컴포넌트에 대한 추가 호출은 수행되지 않습니다.

2차 종료

컴포넌트는 2차 초기화를 위해 시작된 경우 이 옵션으로 시작됩니다.

IBM i IBM i에서의 서비스 및 컴포넌트 구성

서비스 구성 요소는 큐 관리자 구성 파일을 사용하여 설정합니다.

프로시저

1. 큐 매니저 구성 파일(`/etc/sysvnt/queue-manager.qm.iniconf`)에 다음 구문을 추가하여 큐 매니저에서 서비스를 정의하고 모듈 위치를 지정하십시오:

- 사용된 각 서비스에는 큐 관리자에 대한 서비스를 정의하는 **Service** 스탠자가 있어야 합니다. 자세한 내용은 `qm.ini` 파일의 서비스 스탠자(*stanza*)를 참조하십시오.
- 서비스 내 각 컴포넌트의 경우 **ServiceComponent** 스탠자가 있어야 합니다. 이 구절은 해당 구성 요소의 코드를 포함하는 모듈의 이름과 경로를 식별합니다. 자세한 내용은 `qm.ini` 파일의 **ServiceComponent** 구절을 참조하십시오.

오브젝트 권한 관리자(OAM)라고 하는 권한 서비스 컴포넌트가 제품과 함께 제공됩니다. 큐 관리자 작성 시, 큐 관리자 구성 파일은 권한 서비스 및 기본 컴포넌트(OAM)에 대해 적절한 스탠자를 포함하도록 자동으로 업데이트됩니다. 기타 컴포넌트의 경우에는 큐 관리자 구성 파일을 수동으로 구성해야 합니다.

각 서비스 컴포넌트의 코드는 큐 관리자를 시작할 때 동적 바인딩이 지원되는 플랫폼에서 동적 바인딩을 사용하여 큐 관리자로 로드합니다.

2.

IBM i IBM i에서 자체 서비스 컴포넌트 작성

이 정보를 사용하여 IBM MQ for IBM i에서 서비스 컴포넌트의 작성 방법에 대해 알아봅니다.

사용자 자체 서비스 컴포넌트를 작성하려면 다음을 수행하십시오.

- 헤더 파일 `cmqzc.h`가 프로그램에 포함되어 있는지 확인하십시오.
- 프로그램을 컴파일하고 공유 라이브러리 `libmqm*` 및 `libmqmzf*`와 링크하여 공유 라이브러리를 작성하십시오.

참고: 에이전트가 스레드 환경에서 실행될 수 있으므로, 스레드 환경에서 실행할 OAM을 빌드해야 합니다. 이 작업에는 `libmqm` 및 `libmqmzf`의 스레드 버전을 사용하는 것도 포함됩니다.

- 스탠자를 큐 관리자 구성 파일에 추가하여 큐 관리자에 서비스를 정의하고 모듈의 위치를 지정하십시오.
- 큐 관리자를 중지하고 재시작하여 컴포넌트를 활성화하십시오.

IBM i IBM i의 권한 서비스

권한 서비스는 큐 관리자가 권한 기능(예: 사용자 ID에 큐를 열 수 있는 권한이 있는지 검사하는 기능)을 호출할 수 있도록 하는 설치 가능 서비스입니다.

이 서비스는 IBM MQ 보안 사용 인터페이스(SEI)의 컴포넌트이며, IBM MQ 프레임워크의 일부입니다. 다음 주제에 대해 설명합니다.

- 700 페이지의 『OAM(Object Authority Manager)』
- 701 페이지의 『운영 체제에 대한 서비스 정의』
- 701 페이지의 『권한 서비스 스탠자 구성』
- 702 페이지의 『IBM i의 권한 서비스 인터페이스』

OAM(Object Authority Manager)

IBM MQ 제품과 함께 제공된 권한 서비스 컴포넌트는 오브젝트 권한 관리자(OAM)라고 합니다. 기본적으로 OAM은 활성이며 다음 제어 명령으로 작동합니다.

- **WRKMQMAUT** 권한에 대한 작업
- **WRKMQMAUTD** 권한 데이터에 대한 작업
- **DSPMQMAUT** 오브젝트 권한 표시
- **GRTMQMAUT** 오브젝트 권한 부여
- **RVKMQMAUT** 오브젝트 권한 취소
- **RFRMQMAUT** 보안 새로 고치기

이러한 명령의 구문과 해당 명령의 사용 방법은 CL 명령 도움말에 설명되어 있습니다. OAM은 프린시פל 또는 그룹의 엔티티로 작동합니다.

MQI 요청을 작성하거나 명령을 실행할 때, OAM은 조작과 연관된 엔티티의 권한을 검사하여 다음 조치를 수행할 수 있는지 여부를 확인합니다.

- 요청된 조작 수행
- 지정된 큐 관리자 자원 액세스

권한 서비스를 사용하면 고유 권한 서비스 컴포넌트를 작성하여 큐 관리자에 제공된 권한 검사를 기능 보강하거나 바꿀 수 있습니다.

운영 체제에 대한 서비스 정의

큐 관리자 구성 파일 `qm.ini`의 권한 서비스 스탠자는 큐 관리자에 대한 권한 서비스를 정의합니다. 스탠자 유형에 대한 정보는 700 페이지의 『IBM i에서의 서비스 및 컴포넌트 구성』의 내용을 참조하십시오.

권한 서비스 스탠자 구성

IBM MQ for IBM i의 경우:

프린시פל

IBM i 시스템 사용자 프로파일입니다.

그룹

IBM i 시스템 그룹 프로파일입니다.

권한은 그룹 레벨에서만 부여되거나 취소될 수 있습니다. 사용자 권한 부여 또는 취소 요청은 해당 사용자의 1차 그룹을 업데이트합니다.

각 큐 관리자에는 고유 큐 관리자 구성 파일이 있습니다. 예를 들어, 큐 관리자 `QMNAME`에 대한 큐 관리자 구성 파일의 기본 경로 및 파일 이름은 `/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini`입니다.

기본 권한 부여 컴포넌트의 *Service* 스탠자 및 *ServiceComponent* 스탠자는 자동으로 `qm.ini`에 추가되지만, `WRKENVVAR`로 대체될 수 있습니다. 기타 *ServiceComponent* 스탠자는 수동으로 추가해야 합니다.

예를 들어, 큐 관리자 구성 파일의 다음 스탠자는 두 가지 권한 서비스 컴포넌트를 정의합니다.

```
Service:
  Name=AuthorizationService
  EntryPoints=7

ServiceComponent:
  Service=AuthorizationService
  Name=MQ.UNIX.authorization.service
  Module=QMOM/AMQZFU
  ComponentDataSize=0

ServiceComponent:
  Service=AuthorizationService
  Name=user.defined.authorization.service
  Module=LIBRARY/SERVICE PROGRAM NAME
  ComponentDataSize=96
```

그림 73. IBM i의 `qm.ini`에 있는 권한 서비스 스탠자

첫 번째 서비스 컴포넌트 스탠자 `MQ.UNIX.authorization.service`는 기본 권한 서비스 컴포넌트, OAM을 정의합니다. 이 스탠자를 제거하고 큐 관리자를 재시작하는 경우, OAM이 사용 불가능하고 권한 검사가 수행되지 않습니다.

IBM i IBM i의 권한 서비스 인터페이스
권한 서비스는 큐 관리자가 사용할 여러 시작점을 제공합니다.

MQZ_AUTHENTICATE_USER

사용자 ID 및 비밀번호를 인증하고 ID 컨텍스트 필드를 설정할 수 있습니다.

MQZ_CHECK_AUTHORITY

지정된 오브젝트에 대한 하나 이상의 조작을 수행할 수 있는 권한이 엔티티에 있는지 여부를 검사합니다.

MQZ_COPY_ALL_AUTHORITY

참조된 오브젝트에 대해 존재하는 모든 현재 권한을 다른 오브젝트에 복사합니다.

MQZ_DELETE_AUTHORITY

지정된 오브젝트와 연관된 모든 권한을 삭제합니다.

MQZ_ENUMERATE_AUTHORITY_DATA

지정된 선택 기준과 일치하는 모든 권한 데이터를 검색합니다.

MQZ_FREE_USER

연관된 할당 자원을 비웁니다.

MQZ_GET_AUTHORITY

엔티티가 지정된 오브젝트에 액세스하기 위해 갖는 권한을 가져옵니다.

MQZ_GET_EXPLICIT_AUTHORITY

이름 지정된 그룹이 지정된 오브젝트에 액세스하기 위해 갖는 권한(**nobody** 그룹의 추가 권한을 포함하지 않음) 또는 이름 지정된 프린시펄의 1차 그룹이 지정된 오브젝트에 액세스하기 위해 갖는 권한을 가져옵니다.

MQZ_INIT_AUTHORITY

권한 서비스 컴포넌트를 초기화합니다.

MQZ_INQUIRE

지원되는 권한 서비스 기능을 조회합니다.

MQZ_REFRESH_CACHE

모든 권한을 새로 고칩니다.

MQZ_SET_AUTHORITY

엔티티가 지정된 오브젝트에 대해 갖는 권한을 설정합니다.

MQZ_TERM_AUTHORITY

권한 서비스 컴포넌트를 종료합니다.

이러한 시작점은 Windows 보안 ID(NT SID)의 사용을 지원합니다.

이러한 이름은 컴포넌트 기능의 프로토타입을 작성하는 데 사용할 수 있는 헤더 파일 `cmqzc.h`에서 **typedef**로 정의됩니다.

초기화 함수(**MQZ_INIT_AUTHORITY**)는 컴포넌트의 기본 시작점이어야 합니다. 기타 함수는 초기화 함수가 컴포넌트 시작점 벡터에 추가한 시작점 주소를 통해 호출됩니다.

자세한 정보는 700 페이지의 『IBM i에서 자체 서비스 컴포넌트 작성』의 내용을 참조하십시오.

Multi 멀티플랫폼에 API 엑시트 작성 및 컴파일

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.

참고: **z/OS** IBM MQ for z/OS에서 지원되지 않습니다.

API 엑시트 사용 이유

애플리케이션 각각은 수행해야 하는 특정 작업이 있으며 이러한 작업의 코드는 해당 태스크를 최대한 효율적으로 수행해야 합니다. 상위 레벨에서 큐 관리자를 사용하는 모든 애플리케이션에 대해 표준 또는 비즈니스 프로세스를 특정 큐 관리자에 적용하려고 할 수 있습니다. 개별 애플리케이션의 레벨 위에서 이 작업을 수행하는 것은 영향을 받는 각 애플리케이션의 코드를 변경할 필요가 없기 때문에 좀 더 효율적입니다.

다음은 API 엑시트가 도움이 될 수 있는 몇 가지 제안 영역입니다.

보안

보안의 경우, 애플리케이션에 큐 및 큐 관리자에 액세스할 수 있는 권한이 부여되는지 확인하는 인증을 제공할 수 있습니다. 또한 개별 API 호출 또는 이러한 호출이 사용하는 매개변수를 인증하여 애플리케이션의 API 사용을 감시할 수도 있습니다.

유연성

유연성의 경우, 해당 환경의 데이터에 의존하는 애플리케이션을 변경하지 않고 비즈니스 환경의 빠른 변화에 대응할 수 있습니다. 예를 들어, 이자율, 통화 환율 또는 제조 환경에서의 컴포넌트 가격 변경에 대응하는 API 엑시트를 가질 수 있습니다.

큐 또는 큐 관리자의 사용 모니터링

큐 또는 큐 관리자의 사용을 모니터링하기 위해 애플리케이션 및 메시지의 플로우를 추적하고 API 호출의 오류를 기록하거나 회계 용도의 감사 추적을 설정하고 계획 용도를 위해 사용량 통계를 수집할 수 있습니다.

API 엑시트 실행 결과

엑시트 프로그램을 작성하고 IBM MQ에서 이를 식별하면, 큐 관리자는 등록된 지점에서 해당 엑시트 코드를 자동으로 호출합니다.

실행할 API 엑시트 루틴은 멀티플랫폼의 스탠자에서 식별됩니다. 이 주제에서는 구성 파일 `mqs.ini` 및 `qm.ini`의 스탠자를 다룹니다.

루틴에 대한 정의는 다음 세 곳에서 발생할 수 있습니다.

1. `mqs.ini` 파일의 `ApiExitCommon`은 IBM MQ 전체에서 큐 관리자가 시작될 때 적용되는 루틴을 식별합니다. 이러한 루틴은 개별 큐 관리자에 대해 정의된 루틴으로 대체될 수 있습니다(이 목록의 항목 [703 페이지의 『3』](#) 참조).
2. `ApiExitTemplate`, `mqs.ini` 파일에서, IBM MQ 전체에 대한 루틴을 식별하며, 새 큐 관리자가 생성될 때 `ApiExitLocal` 세트(이 목록의 [703 페이지의 『3』](#) 항목 참조)로 복사됩니다.
3. `qm.ini` 파일의 `ApiExitLocal`은 특정 큐 관리자에 적용되는 루틴을 식별합니다.

새 큐 관리자가 작성될 때, `mqs.ini`의 `ApiExitTemplate` 정의는 해당 새 큐 관리자에 해당되는 `qm.ini`의 `ApiExitLocal` 정의로 복사됩니다. 큐 관리자가 시작되면 `ApiExitCommon` 및 `ApiExitLocal` 정의가 모두 사용됩니다. 두 정의 모두 동일한 이름의 루틴을 식별하는 경우 `ApiExitLocal` 정의가 `ApiExitCommon` 정의를 대체합니다. 속성은 Sequence 스탠자에 정의된 루틴들이 [707 페이지의 『API 엑시트 구성』](#) 실행되는 순서를 결정합니다.

여러 IBM MQ 설치에서 API 엑시트 사용

이전 버전의 API 출구(IBM MQ exits)는 모든 버전에서 작동하도록 작성되어야 합니다. 왜냐하면 버전에서 출구에 가해진 변경 사항이 이전 IBM WebSphere MQ 7.1 버전에서는 작동하지 않을 수 있기 때문입니다. 엑시트 변경에 대한 자세한 정보는 [688 페이지의 『AIX, Linux, and Windows에 엑시트 및 설치 가능 서비스 작성』](#)의 내용을 참조하십시오.

API 출구 `amqsaem` 및 `amqsaxe`에 제공된 샘플은 출구 작성 시 필요한 변경 사항을 반영합니다. 클라이언트 애플리케이션은 애플리케이션 시작 전에 애플리케이션과 연관되는 큐 관리자의 설치에 해당하는 올바른 IBM MQ 라이브러리가 링크되어 있는지 확인해야 합니다.

Multi API 엑시트 작성

C 프로그래밍 언어를 사용하여 모든 API 호출에 대해 엑시트를 작성할 수 있습니다.

사용 가능한 엑시트

엑시트는 다음과 같이 모든 API 호출에서 사용할 수 있습니다.

- MQCB. 지정된 오브젝트 핸들의 콜백을 재등록하고 콜백에 대한 변경 및 활성화를 제어합니다.
- MQCTL. 연결을 위해 열려 있는 오브젝트 핸들에 대한 제어 조치를 수행합니다.
- MQCONN/MQCONN. 후속 API 호출에 사용하는 큐 관리자 연결 핸들을 제공합니다.
- MQDISC. 큐 관리자와의 연결을 끊습니다.

- MQBEGIN. 글로벌 작업 단위(UOW)를 시작합니다.
- MQBACK. UOW를 백아웃합니다.
- MQCMIT. UOW를 커밋합니다.
- MQOPEN. 후속 액세스를 위한 IBM MQ 자원을 엽니다.
- MQCLOSE. 이전에 액세스를 위해 열린 IBM MQ 자원을 닫습니다.
- MQGET. 이전에 액세스를 위해 열린 큐에서 메시지를 검색합니다.
- MQPUT1. 큐에 메시지를 배치합니다.
- MQPUT. 이전에 액세스를 위해 열린 큐에 메시지를 배치합니다.
- MQINQ. 이전에 액세스를 위해 열린 IBM MQ 자원의 속성을 조회합니다.
- MQSET. 이전에 액세스를 위해 열린 큐의 속성을 설정합니다.
- MQSTAT. 상태 정보를 검색합니다.
- MQSUB, 애플리케이션의 특정 토픽 구독을 등록합니다.
- MQSUBRQ, 구독 요청을 작성합니다.

MQ_CALLBACK_EXIT는 콜백 처리 전 및 후에 수행할 엑시트 함수를 제공합니다. 자세한 정보는 [콜백 - MQ_CALLBACK_EXIT](#)를 참조하십시오.

API 엑시트 작성

API 엑시트 내에서 호출은 다음의 일반 양식을 사용합니다.

```
MQ_call_EXIT (parameters, context, ApiCallParameters)
```

여기에서 *call*은 MQ 접두부가 없는 MQI 호출 이름입니다(예: PUT, GET). *parameters*는 주로 엑시트와 외부 제어 블록인 MQAXP(API 엑시트 매개변수 구조) 및 MQAXC(API 엑시트 컨텍스트 구조) 간의 통신을 제공하는 엑시트의 함수를 제어합니다. *context*는 API 엑시트가 호출된 컨텍스트를 설명하고, *ApiCallParameters*는 MQI 호출에 대한 매개변수를 나타냅니다.

API 엑시트 작성을 돕기 위해 샘플 엑시트인 amqsaxe0.c가 제공되며, 이 엑시트는 사용자가 지정하는 파일에 추적 항목을 생성합니다. 엑시트를 작성할 때 이 샘플을 시작점으로 사용할 수 있습니다. 샘플 엑시트 사용에 대한 자세한 정보는 986 페이지의 『API 엑시트 샘플 프로그램』의 내용을 참조하십시오.

API 엑시트 호출, 외부 제어 블록 및 연관된 토픽에 대한 자세한 정보는 [API 엑시트 참조](#)를 확인하십시오.

엑시트를 작성, 컴파일 및 구성하는 방법에 대한 일반 정보는 688 페이지의 『AIX, Linux, and Windows에 엑시트 및 설치 가능 서비스 작성』의 내용을 참조하십시오.

필수 라이브러리

사용 가능한 32비트 및 64비트 라이브러리 목록은 [외부 라이브러리](#) 항목을 참조하십시오.

API 엑시트에서 메시지 핸들 사용

API 엑시트가 액세스할 수 있는 메시지 특성을 제어할 수 있습니다. 특성은 ExitMsgHandle과 연관됩니다. Put 엑시트에 설정된 특성은 넣을 메시지에 설정되지만, Get 엑시트에서 검색된 특성은 애플리케이션으로 리턴되지 않습니다.

Function이 MQXF_INIT로 설정되고 **ExitReason**이 MQXR_CONNECTION으로 설정된 MQXEP MQI 호출을 사용하여 MQ_INIT_EXIT 엑시트 함수를 등록할 때, MQXEPO 구조를 **ExitOpts** 매개변수로 전달합니다. MQXEPO 구조에는 엑시트에 사용 가능하도록 특성 세트를 지정하는 ExitProperties 필드가 포함됩니다. 이 필드는 MQRFH2 폴더 이름에 해당하는 특성의 접두부를 나타내는 문자열로 지정됩니다.

각 API 엑시트는 ExitMsgHandle 필드를 포함하는 MQAXP 구조를 수신합니다. 이 필드는 IBM MQ에 의해 생성된 값으로 설정되며 연결에 고유합니다. 따라서 동일하게 연결된 같거나 다른 유형의 API 엑시트 사이에서는 핸들이 변경되지 않습니다.

API 엑시트가 메시지를 넣기 전에 수행한 MQXR_BEFORE의 **ExitReason**을 포함하는 MQ_PUT_EXIT 또는 MQ_PUT1_EXIT에서는 엑시트 완료 시 ExitMsgHandle과 연관된 모든 특성(메시지 디스크립터 특성은 제외)이 넣을 메시지에 설정됩니다. 이러한 문제를 방지하려면 ExitMsgHandle을 MQHM_NONE으로 설정하십시오. 또한 다른 메시지 핸들을 제공할 수도 있습니다.

MQ_GET_EXIT 및 MQ_CALLBACK_EXIT 시, 메시지 설명자(ExitMsgHandle)는 속성이 지워지고, MQ_INIT_EXIT이 등록되었을 때 메시지 설명자 속성(message descriptor properties)을 제외한 메시지 설명자 속성(ExitProperties) 필드에 지정된 속성들로 채워집니다. 이러한 특성은 가져오기 애플리케이션에서 사용할 수 없습니다. 가져오기 애플리케이션이 MQGMO(메시지 가져오기 옵션) 필드에 메시지 핸들을 지정한 경우, 메시지 디스크립터 특성을 포함하여 해당 핸들과 연관된 모든 특성을 API 엑시트에서 사용할 수 있습니다. ExitMsgHandle을 특성으로 채워지지 않게 하려면 MQHM_NONE으로 설정하십시오.

샘플 프로그램인 amqsaxem0.c는 API 엑시트에서 메시지 핸들의 사용을 설명하기 위해 제공됩니다.

관련 참조

[사용자 엑시트, API 엑시트 및 설치 가능 서비스 참조](#)

Multi API 엑시트 컴파일

엑시트를 작성한 후에 다음과 같이 컴파일하고 링크합니다.

다음 예는 986 페이지의 『API 엑시트 샘플 프로그램』에 설명된 샘플 프로그램에 사용된 명령을 보여줍니다. Windows 시스템 이외 플랫폼의 경우 MQ_INSTALLATION_PATH/samp의 샘플 API 엑시트 코드 및 MQ_INSTALLATION_PATH/samp/bin의 컴파일되고 링크된 공유 라이브러리를 찾을 수 있습니다.

Windows Windows 시스템의 경우, MQ_INSTALLATION_PATH\Tools\c\Samples에서 샘플 API 엑시트 코드를 찾을 수 있습니다. MQ_INSTALLATION_PATH는 IBM MQ가 설치된 상위 레벨 디렉토리를 나타냅니다.

참고: 64비트 애플리케이션 프로그래밍에 대한 자세한 내용은 [64비트 플랫폼의 코딩 표준](#)에 나열되어 있습니다.

멀티캐스트 클라이언트의 경우, API 엑시트 및 데이터 변환 엑시트는 일부 메시지가 큐 관리자를 통해 전달되지 않을 수 있기 때문에 클라이언트 측에서 실행될 수 있어야 합니다. 다음 라이브러리는 서버 패키지와 마찬가지로 클라이언트 패키지의 일부입니다.

표 117. 클라이언트 및 서버 패키지에 있는 라이브러리	
운영 체제	라이브러리
AIX AIX	32비트 및 64비트: libmqm.a 및 libmqm_r.a
IBM i IBM i	LIBMQM 및 LIBMQM_R
Linux Linux	32비트 및 64비트: libmqm.so 및 libmqm_r.so
Windows Windows	32비트 및 64비트: mqm.dll 및 mqm.pdb

Linux AIX AIX and Linux 시스템의 API 엑시트 컴파일

AIX and Linux 시스템에서 API 엑시트를 컴파일하는 방법의 예입니다.

모든 플랫폼에서 모듈의 시작점은 MQStart입니다.

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

AIX의 경우

AIX

다음 명령 중 하나를 실행하여 API 엑시트 소스 코드를 컴파일하십시오.

32비트 애플리케이션 스레드되지 않음

```
cc -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

스레드됨

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

64비트 애플리케이션 스레드되지 않음

```
cc -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

스레드됨

```
xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Linux의 경우



다음 명령 중 하나를 실행하여 API 엑시트 소스 코드를 컴파일하십시오.

32비트 애플리케이션 스레드되지 않음

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
-I MQ_INSTALLATION_PATH/inc
```

스레드됨

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
-I MQ_INSTALLATION_PATH/inc
```

64비트 애플리케이션 스레드되지 않음

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
-I MQ_INSTALLATION_PATH/inc
```

스레드됨

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
-I MQ_INSTALLATION_PATH/inc
```



Windows Windows 시스템의 API 엑시트 컴파일

Windows에서 샘플 API 엑시트 프로그램, amqsaxe0.c 컴파일 및 링크

Manifest 파일은 컴파일된 애플리케이션 또는 DLL에 임베드될 수 있는 버전 또는 기타 정보를 포함하는 선택적 XML 문서입니다.

이러한 문서가 없는 경우, **mt** 명령에서 **-manifest manifest.file** 매개변수를 생략하십시오.

707 페이지의 그림 74 또는 707 페이지의 그림 75의 예에 명령을 적용하여 Windows에서 amqsaxe0.c을 (를) 컴파일하고 링크하십시오. 명령은 Microsoft Visual Studio 2008, 2010 또는 2012에서 작동합니다. 예에서는 C:\Program Files\IBM\MQ\tools\c\samples 디렉토리가 현재 디렉토리라고 가정합니다.

32비트

```
cl /c /nologo /MD /Foamsaxe0.obj amqsaxe0.c
link /nologo /dll /def:amsaxe.def

amsaxe0.obj \
/manifest /out:amsaxe.dll

mt -nologo -manifest amsaxe.dll.manifest \
-outputresource:amsaxe.dll;2
```

그림 74. 32비트 Windows에서 *amsaxe0.c* 컴파일 및 링크

64비트

```
cl /c /nologo /MD /Foamsaxe0.obj amqsaxe0.c
link /nologo /dll /def:amsaxe.def \
/libpath:..\..\lib64 \

amsaxe0.obj /manifest /out:amsaxe.dll

mt -nologo -manifest amsaxe.dll.manifest \
-outputresource:amsaxe.dll;2
```

그림 75. 64비트 Windows에서 *amsaxe0.c* 컴파일 및 링크

관련 개념

986 페이지의 『API 엑시트 샘플 프로그램』

이 샘플 API 종료 시, 환경 변수에 **MQAPI_TRACE_LOGFILE** 정의된 접두사를 사용하여 사용자가 지정한 파일에 'MQI' 추적 정보를 생성합니다.

IBM i 'IBM i'에서 API 종료 컴파일하기
IBM i에서 API 엑시트를 컴파일합니다.

엑시트는 다음과 같이 작성됩니다(C 언어 예제의 경우).

1. CRTCMOD를 사용하여 모듈을 작성하십시오. TERASPACE(*YES *TSIFC) 매개변수를 포함하여 테라스페이스를 사용하도록 해당 모듈을 컴파일하십시오.
2. CRTSRVPGM을 사용하여 모듈에서 서비스 프로그램을 작성하십시오. 멀티스레드 API 엑시트에 대한 서비스 프로그램 QMQM/LIBMQMZF_R에 해당 서비스 프로그램을 바인딩해야 합니다.

API 엑시트 구성

구성 정보를 변경하여 API 엑시트를 사용하도록 IBM MQ를 구성합니다.

구성 정보를 변경하려면 엑시트 루틴 및 엑시트 루틴이 실행되는 순서를 정의하는 스탠자를 변경해야 합니다. 이 정보는 다음 방법으로 변경할 수 있습니다.

- **Windows** **Linux** 'Windows' 및 'Linux'(x86 및 x86-64 플랫폼)에서 'IBM MQ Explorer'를 사용합니다.
- **Windows** 'Windows'에서 'amqmdain' 명령 사용.
- **Multi** 멀티플랫폼에서 mqs.ini 및 qm.ini 파일 직접 사용.

mqm.ini 파일에는 특정 노드의 모든 큐 관리자에 관련된 정보가 포함되어 있습니다. 다음 위치에서 이 파일을 찾을 수 있습니다.

-   AIX and Linux의 /var/mqm 디렉토리.
-  'Windows' 시스템의 'HKLM\SOFTWARE\IBM\WebSphere MQ' 키에 지정된 WorkPath.
-  IBM i의 /QIBM/UserData/mqm 디렉토리.

qm.ini 파일에는 특정 큐 관리자에 관련된 정보가 포함되어 있습니다. 각 큐 관리자에 대해서 하나의 큐 관리자 구성 파일이 있으며 큐 관리자가 사용 중인 디렉토리 트리의 루트에 보유됩니다. 예를 들어, QMNAME이라는 큐 관리자의 구성 파일에 대한 경로 및 이름은 다음과 같습니다.

 IBM i 시스템:

```
/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini
```

  AIX and Linux 시스템:

```
/var/mqm/qmgrs/QMNAME/qm.ini
```

 Windows 시스템:

```
C:\ProgramData\ IBM \ MQ\qmgrs\QMNAME\qm.ini
```

구성 파일을 편집하기 전에, 필요에 따라 되돌릴 수 있도록 백업 사본을 만들어 두십시오.

다음 방법 중 하나로 구성 파일을 편집할 수 있습니다.

- 노드에서 큐 관리자 구성을 변경하는 명령을 사용하여 자동으로 편집
- 표준 텍스트 편집기를 사용하여 수동으로

구성 파일 속성에 올바르지 않은 값을 설정한 경우, 이 값을 무시하고 문제점을 표시하는 운영자 메시지가 발행됩니다. 결과는 속성 전체를 누락한 것과 동일합니다.

구성할 스탠자

변경해야 할 스탠자는 다음과 같습니다.

ApiExitCommon

mqm.ini 및 IBM MQ Explorer의 IBM MQ 특성 페이지에서, 엑시트 아래에 정의됩니다.

큐 관리자가 시작되면 이 스탠자의 속성은 판독된 후 qm.ini에 정의된 API 엑시트로 재정의됩니다.

ApiExitTemplate

mqm.ini 및 IBM MQ Explorer의 IBM MQ 특성 페이지에서, 엑시트 아래에 정의됩니다.

큐 관리자가 작성되면 이 스탠자의 속성이 ApiExitLocal 스탠자 아래에 새로 작성된 qm.ini 파일에 복사됩니다.

ApiExitLocal

qm.ini 및 IBM MQ Explorer의 큐 관리자 특성 페이지에서 엑시트 아래에 정의됩니다.

큐 관리자가 시작되면 여기에 정의된 API 엑시트는 qm.ini에 정의된 기본값을 재정의합니다.

스탠자에 대한 속성

- 다음 속성을 사용하여 API 엑시트의 이름을 지정하십시오.

Name=ApiExit_name

MQAXP 구조의 ExitInfoName 필드에서 API 엑시트에 전달되는 해당 엑시트의 설명 이름입니다.

이 이름은 고유해야 하며 48자를 초과해서는 안 되고 IBM MQ 오브젝트의 이름(예: 큐 이름)에 대해 올바른 문자만을 포함해야 합니다.

- 다음 속성을 사용하여 실행할 API 엑시트 코드의 모듈 및 시작점을 식별하십시오.

Function=function_name

API 엑시트 코드를 포함하는 모듈에 대한 함수 시작점 이름입니다. 이 시작점은 MQ_INIT_EXIT 함수입니다.

이 필드의 길이는 MQ_EXIT_NAME_LENGTH로 제한됩니다.

Module=module_name

API 엑시트 코드가 포함된 모듈입니다.

이 필드에 모듈의 전체 경로 이름이 들어 있으면 그대로 사용됩니다.

이 필드에 모듈 이름만 포함된 경우, qm.ini의 ExitPath에 있는 ExitsDefaultPath 속성을 사용하여 모듈을 찾습니다.

별도의 스레드 라이브러리를 지원하는 플랫폼에서 API 엑시트 모듈의 스레드 버전과 비스레드 버전을 둘 다 제공해야 합니다. 스레드 버전에는 _r 접미부가 있어야 합니다. IBM MQ 애플리케이션 스텝의 스레드 버전은 로드되기 전에 제공된 모듈 이름에 _r을 암시적으로 추가합니다.

이 필드의 길이는 플랫폼이 지원하는 최대 경로 길이로 제한됩니다.

- 다음 속성을 사용하여 엑시트와 함께 데이터를 선택적으로 전달하십시오.

Data=data_name

MQAXP 구조의 ExitData 필드에서 API 엑시트로 전달할 데이터입니다.

이 속성을 포함하는 경우, 앞과 뒤의 공백이 제거되고 나머지 문자열은 32자로 잘리며 결과는 엑시트로 전달됩니다. 이 속성을 생략하는 경우, 기본값인 32개의 공백이 엑시트로 전달됩니다.

이 필드의 최대 길이는 32자입니다.

- 다음 속성을 사용하여 다른 엑시트에 관하여 이 엑시트의 순서를 식별하십시오.

Sequence=sequence_number

다른 API 엑시트와 비교하여 이 API 엑시트가 호출되는 순서입니다. 낮은 순서 번호를 가진 엑시트가 더 높은 순서 번호를 가진 엑시트보다 먼저 호출됩니다. 엑시트의 순서 번호 매기기가 연속적일 필요는 없습니다. 1, 2, 3의 순서는 7, 42, 1096의 순서와 동일한 결과를 가집니다. 두 엑시트의 순서 번호가 동일한 경우, 큐 관리자는 먼저 호출할 엑시트를 결정합니다. MQAXP의 ExitChainAreaPtr로 표시된 ExitChainArea에 시간 또는 마커를 넣거나 사용자 자체 로그 파일을 작성하여 이벤트 후에 호출된 엑시트를 알 수 있습니다.

이 속성은 사인되지 않은 숫자 값입니다.

샘플 스탠자

샘플 mqs.ini 파일에는 다음 스탠자가 포함되어 있습니다.

ApiExitTemplate

이 스탠자는 설명 이름 OurPayrollQueueAuditor, 모듈 이름 auditor 및 순서 번호 2로 엑시트를 정의합니다. 데이터 값 123이 엑시트에 전달됩니다.

ApiExitCommon

이 스탠자는 설명 이름 MQPoliceman, 모듈 이름 tmqp 및 순서 번호 1로 엑시트를 정의합니다. 전달된 데이터는 명령어(CheckEverything)입니다.

```
mqs.ini

ApiExitTemplate:
  Name=OurPayrollQueueAuditor
  Sequence=2
  Function=EntryPoint
  Module=/usr/ABC/auditor
  Data=123
ApiExitCommon:
  Name=MQPoliceman
  Sequence=1
  Function=EntryPoint
```

```
Module=/usr/MQPolice/tmqp
Data=CheckEverything
```

다음 샘플 qm.ini 파일에는 설명 이름이 ClientApplicationAPIchecker이고 모듈 이름이 ClientAppChecker이며 순서 번호가 3인 엑시트의 ApiExitLocal 정의가 포함되어 있습니다.

```
qm.ini

ApiExitLocal:
  Name=ClientApplicationAPIchecker
  Sequence=3
  Function=EntryPoint
  Module=/usr/Dev/ClientAppChecker
  Data=9.20.176.20
```

메시지 채널에 대한 채널 엑시트 프로그램

이 주제 컬렉션에는 메시징 채널의 IBM MQ 채널 엑시트 프로그램에 대한 정보가 포함되어 있습니다.

메시지 채널 에이전트(MCA)는 데이터 변환 엑시트를 호출할 수도 있습니다. 데이터 변환 엑시트 작성에 대한 자세한 정보는 729 페이지의 『데이터 변환 엑시트 작성』의 내용을 참조하십시오.

이 정보 중 일부는 IBM MQ MQI clients를 큐 관리자에 연결하는 MQI 채널의 엑시트에도 적용됩니다. 자세한 정보는 MQI 채널에 대한 채널 엑시트 프로그램을 참조하십시오.

채널 엑시트 프로그램은 MCA 프로그램이 수행하는 처리 중에 정의된 위치에서 호출됩니다.

이러한 사용자 엑시트 프로그램 중 일부는 보완적인 쌍으로 작동합니다. 예를 들어, 송신 MCA가 전송할 메시지를 암호화하려고 사용자 엑시트 프로그램을 호출한 경우 수신 끝에서 보완 프로세스가 프로세스를 취소하는 기능을 수행해야 합니다.

710 페이지의 표 118에서는 각 채널 유형에 대해 사용 가능한 채널 엑시트의 유형을 보여줍니다.

표 118. 각 채널 유형에 사용 가능한 채널 엑시트						
채널 유형	메시지 엑시트	메시지 재시도 엑시트	수신 엑시트	보안 엑시트	송신 엑시트	자동 정의 엑시트
송신자 채널	예		예	예	예	
서버 채널	예		예	예	예	
클러스터-송신자 채널	예		예	예	예	예
수신자 채널	예	예	예	예	예	예
요청자 채널	예	예	예	예	예	
클러스터-수신자 채널	예	예	예	예	예	예
클라이언트-연결 채널			예	예	예	
서버-연결 채널			예	예	예	예

참고:  z/OS

1. z/OS에서 자동 정의 엑시트는 클러스터-송신자 및 클러스터-수신자 채널에만 적용됩니다.

클라이언트에서 채널 엑시트를 실행할 경우 MQSERVER 환경 변수를 사용할 수 없습니다. 대신, 클라이언트 채널 정의 테이블에서 설명한 대로 클라이언트 채널 정의 테이블(CCDT)을 작성하고 참조하십시오.

처리 개요

MCA가 채널 엑시트 프로그램을 사용하는 방법의 개요입니다.

시동 시 MCA는 시동 대화 상자를 교환하여 처리를 동기화합니다. 그런 다음, 보안 엑시트를 포함하는 데이터 교환으로 전환됩니다. 시동 단계가 완료되고 메시지가 전송되도록 하려면 이러한 엑시트가 정상적으로 끝나야 합니다.

보안 검사 단계는 711 페이지의 그림 76에 표시된 바와 같은 루프입니다.

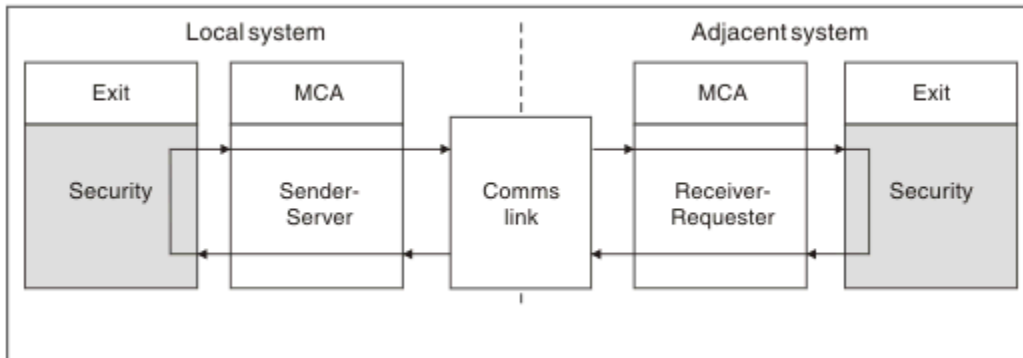


그림 76. 보안 엑시트 루프

메시지 전송 단계 중에 송신 MCA는 711 페이지의 그림 77와 같이 전송 큐에서 메시지를 가져오고 메시지 엑시트를 호출한 후 메시지를 수신 MCA로 송신합니다.

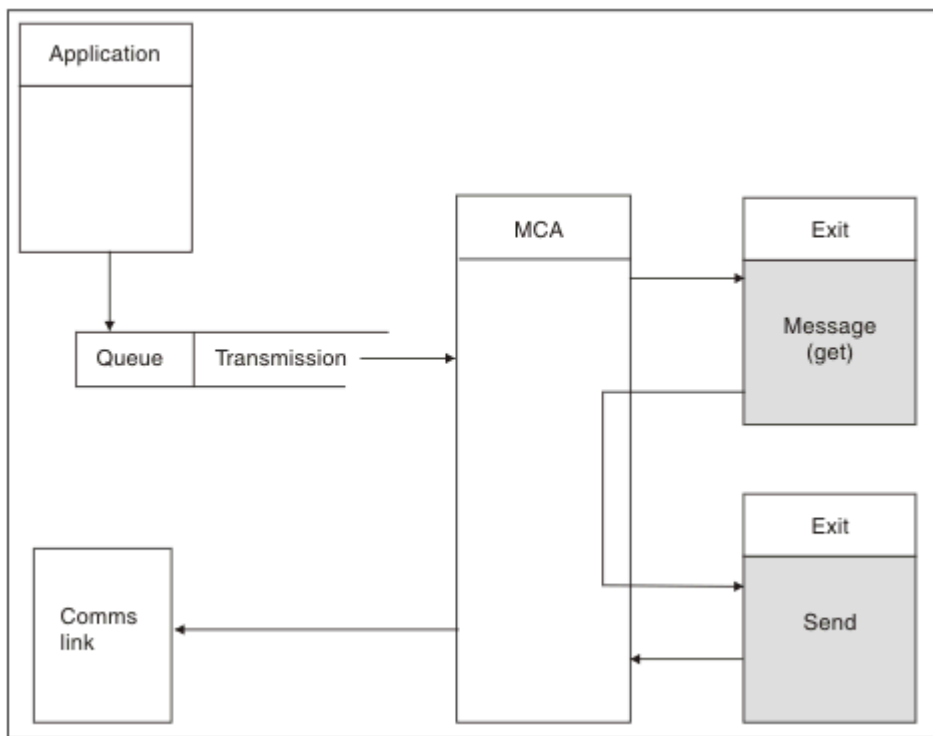


그림 77. 메시지 채널의 송신자 끝에서의 송신 엑시트의 예

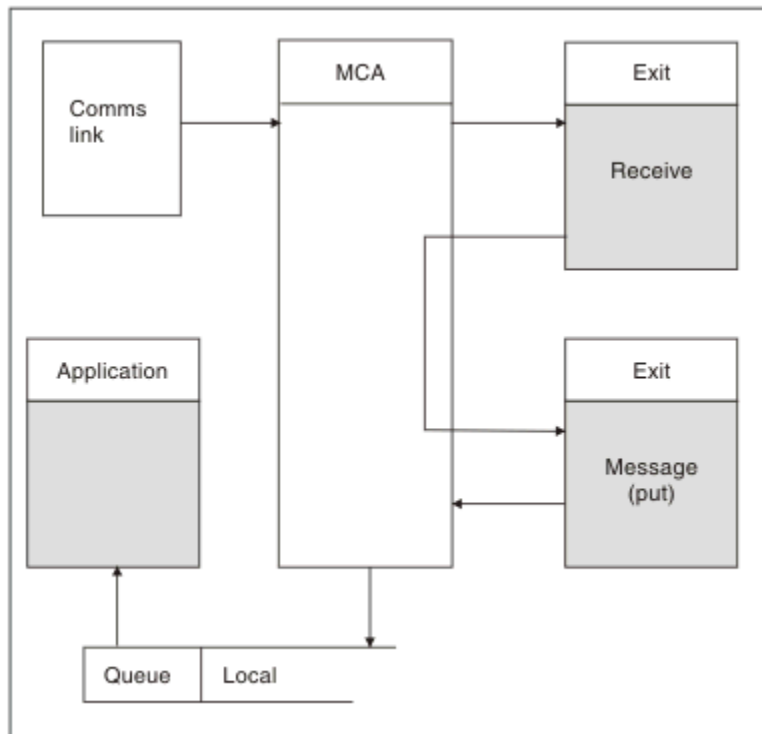


그림 78. 메시지 채널의 수신자 끝에서의 수신 엑시트의 예

712 페이지의 그림 78와 같이 수신 MCA는 통신 링크에서 메시지를 수신하고 수신 엑시트를 호출하고 메시지 엑시트를 호출한 후 로컬 큐에 메시지를 넣습니다. (메시지 엑시트를 호출하기 전에 수신 엑시트를 두 번 이상 호출할 수 있습니다.)

채널 엑시트 프로그램 작성

다음 정보를 사용하면 채널 엑시트 프로그램 작성에 도움이 됩니다.

사용자 엑시트 및 채널 엑시트 프로그램은 다음 절에서 언급된 것을 제외한 모든 MQI 호출을 사용할 수 있습니다. MQ V7 이상의 경우, MQCXP 구조 버전 7 이상에는 MQCONN을 발행하는 대신 사용할 수 있는 연결 핸들 hConn이 있습니다. 이전 버전의 경우, 채널 자체가 큐 관리자에 연결되기 때문에 MQRC_ALREADY_CONNECTED 경고가 리턴되더라도 연결 핸들을 확보하기 위해 MQCONN을 발행해야 합니다.

채널 엑시트는 스레드에 안전하게 설계되어 있어야 합니다.

클라이언트 연결 채널의 엑시트의 경우, 엑시트가 연결하려고 하는 큐 관리자는 엑시트가 링크된 방법에 따라 달라집니다. 엑시트가 MQM.LIB(또는 IBM i의 QMQM/LIBMQM)에 링크된 경우 MQCONN 호출에 큐 관리자 이름을 지정하지 않으면, 엑시트는 시스템의 기본 큐 관리자에 연결하려고 합니다. 엑시트가 MQM.LIB(또는 IBM i의 QMQM/LIBMQM)에 링크된 경우 MQCD의 QMgrName 필드를 통해 엑시트에 전달된 큐 관리자의 이름을 지정하면, 엑시트는 해당 큐 관리자에 연결하려고 합니다. 엑시트가 MQIC.LIB 또는 다른 라이브러리와 링크된 경우, 큐 관리자 이름 지정 여부에 관계없이 MQCONN 호출은 실패합니다.

채널 엑시트에서 전달된 hConn과 연관된 트랜잭션의 상태는 대체하지 않아야 합니다. 채널 hConn과 함께 MQCMIT, MQBACK 또는 MQDISC verb를 사용해서는 안 되며 채널 hConn을 지정하는 MQBEGIN verb를 사용할 수 없습니다.

MQCNO_HANDLE_SHARE_BLOCK 또는 MQCNO_HANDLE_SHARE_NO_BLOCK을 지정하는 MQCONNx를 사용하여 새 IBM MQ 연결을 작성하는 경우, 연결이 올바르게 관리되고 큐 관리자에서 올바르게 연결이 해제되는지 확인해야 합니다. 예를 들어, 연결을 끊지 않고 모든 호출에서 큐 관리자에 대한 새 연결을 작성하는 채널 엑시트가 있으면 연결 핸들이 빌드되고 에이전트 스레드 수가 증가합니다.

엑시트는 MCA 자체와 동일한 스레드에서 실행되며 동일한 연결 핸들을 사용합니다. 그러므로 MCA와 동일한 UOW 안에서 실행되며, 동기점에서 수행된 모든 호출은 배치의 마지막에 채널에서 커밋되거나 백아웃됩니다.

그러므로 원래 메시지가 포함된 배치가 커미트될 때, 채널 메시지 엑시트는 해당 큐에만 커미트되는 알림 메시지를 보낼 수 있습니다. 따라서 채널 메시지 엑시트로부터 동기점 MQI 호출을 발행하는 것이 가능합니다.

채널 엑시트는 MQCD의 필드를 변경할 수 있습니다. 그러나 이러한 변경사항은 나열된 환경을 제외하고는 적용되지 않습니다. 채널 엑시트 프로그램에서 MQCD 데이터 구조의 필드를 변경한 경우 IBM MQ 채널 프로세스는 새 값을 무시합니다. 그러나 새 값은 MQCD에 남아 있으며, 엑시트 체인에 남아 있는 모든 엑시트와 채널 인스턴스를 공유하는 모든 대화에 전달됩니다. 자세한 정보는 채널 엑시트에서 MQCD 필드 변경을 참조하십시오.

또한 C로 기록된 프로그램의 경우, 채널 엑시트 프로그램에서 재입력되지 않는 C 라이브러리 함수를 사용하지 않아야 합니다.

Linux **AIX** 동시에 여러 채널 엑시트 라이브러리를 사용하는 경우, 두 가지 다른 엑시트의 코드가 동일한 이름의 함수를 포함하고 있으면 일부 UNIX and Linux 플랫폼에서 문제점이 발생할 수 있습니다. 채널 엑시트가 로드되면 동적 로더는 엑시트 라이브러리의 함수 이름을 라이브러리가 로드된 주소로 해석합니다. 두 개의 엑시트 라이브러리가 동일한 이름을 가진 별도의 함수를 정의하는 경우, 이 해석 프로세스는 한 라이브러리의 함수 이름을 잘못 해석하여 다른 라이브러리의 함수를 사용할 수 있습니다. 이 문제점이 발생하더라도 MQStart 함수는 영향을 받지 않으므로 필수 엑시트 및 이러한 함수만 내보내도록 링커에 지정하십시오. 기타 함수에는 해당 고유 엑시트 라이브러리 외부의 함수에서 사용되지 않도록 로컬 가시성을 제공해야 합니다. 자세한 정보는 링커 문서를 참조하십시오.

모든 엑시트는 채널 엑시트 매개변수 구조(MQXCP), 채널 정의 구조(MQCD), 준비된 데이터 버퍼, 데이터 길이 매개변수 및 버퍼 길이 매개변수와 함께 호출됩니다. 버퍼 길이를 초과하지 않아야 합니다.

- 메시지 엑시트의 경우, MQXQH 구조 길이 외에 채널을 통해 송신해야 할 최대 메시지를 감안해야 합니다.
- 송신 및 수신 엑시트에 대해 허용해야 할 최대 버퍼는 다음과 같습니다.

LU 6.2

32KB

TCP:

IBM i IBM i 16KB

IBM i 기타 32KB

참고: 사용 가능한 최대 길이는 이 길이보다 2바이트 작을 수 있습니다. 자세한 내용은 **MaxSegmentLength**에서 반환된 값을 확인하십시오. 에 대한 자세한 **MaxSegmentLength**내용은 을 참조하십시오 [MaxSegmentLength](#).

NetBIOS:

64KB

SPX:

64KB

참고: 송신자 채널의 수신 엑시트와 수신자 채널의 송신자 엑시트는 TCP용으로 2KB 버퍼를 사용합니다.

- 보안 엑시트의 경우, 분산 큐잉 기능이 4000바이트의 버퍼를 할당합니다.

엑시트가 관련 매개변수와 함께 대체 버퍼를 리턴하는 것이 허용됩니다. 호출 세부사항은 [710 페이지의 『메시지 채널에 대한 채널 엑시트 프로그램』](#)의 내용을 참조하십시오.

z/OS z/OS에서 채널 엑시트 프로그램 작성

다음 정보를 사용하면 z/OS용 채널 엑시트 프로그램을 작성하고 컴파일하는 데 도움이 됩니다.

엑시트는 다음 상태에서 z/OS LINK에 의한 것처럼 시작됩니다.

- 비승인 문제점의 프로그램 상태
- 1차 주소 공간 제어 모드
- 비교차 메모리 모드
- 비액세스 등록 모드
- 31비트 주소 지정 모드

링크 편집된 모듈은 채널 시작기 주소 공간 프로시저의 CSQXLIB DD문에 의해 지정된 데이터 세트에 배치되어야 합니다. 로드 모듈 이름은 채널 정의에서 엑시트 이름으로 지정됩니다.

z/OS에 대한 채널 엑시트를 작성할 때 다음 규칙이 적용됩니다.

- 엑시트는 어셈블러 또는 C로 작성해야 합니다. C가 사용되는 경우 z/OS C/C++ 프로그래밍 안내서에 설명된 시스템 엑시트에 대한 C 시스템 프로그래밍 환경을 준수해야 합니다.
- 엑시트는 CSQXLIB DD문으로 정의된 비승인 라이브러리에서 로드됩니다. CSQXLIB에 DISP=SHR이 있는 경우, 채널 시작기가 실행 중인 동안 엑시트를 업데이트할 수 있습니다. 채널이 재시작될 때 새 버전이 사용됩니다.
- 엑시트는 재입력 가능해야 하고 가상 스토리지의 임의의 위치에서 실행할 수 있어야 합니다.
- 엑시트는 리턴 시 환경을 시작 시 환경으로 재설정해야 합니다.
- 엑시트는 확보된 스토리지를 비우거나, 후속 엑시트 호출로 스토리지가 비워졌는지 확인해야 합니다.

호출 사이에 스토리지를 지속해야 할 경우, z/OS STORAGE 서비스를 사용하거나 System Programming C용 4kmalc 라이브러리 함수를 사용하십시오.

이 함수에 대한 자세한 정보는 4kmalc() -- 페이지 정렬 스토리지 할당을 참조하십시오.

- MQCMIT 또는 CSQBCMT 및 MQBACK 또는 CSQBBAK를 제외한 모든 IBM MQ MQI 호출을 사용할 수 있습니다. MQCONN 다음에 포함되어야 합니다(큐 관리자 이름은 공백임). 이러한 호출이 사용되면, 스텝 CSQXSTUB으로 엑시트를 링크 편집해야 합니다.

이 규칙의 예외는 보안 채널 엑시트가 MQI 호출을 발행, 커밋 및 백아웃할 수 있다는 것입니다. 이러한 호출을 발행하려면 MQCMIT 또는 CSQBCMT 및 MQBACK 또는 CSQBBAK 대신에 verb CSQXCMT 및 CSQXBAK를 코딩하십시오.

- IBM WebSphere MQ 7.0 이상에서 스텝 CSQXSTUB을 사용하는 모든 엑시트는 PDS-E 형식으로 CSQXLIB 로드 라이브러리에서 링크 편집해야 합니다.
- 시스템 서비스를 사용하면 일부 채널 또는 다른 모든 채널의 핸들링에 심각한 영향이 미치므로, 엑시트는 대기를 야기하는 시스템 서비스를 사용하지 않아야 합니다. 많은 채널은 일반적으로 단일 TCB 하에서 실행됩니다. 대기를 야기하는 엑시트에서 특정 작업을 수행하고 MQXWAIT를 사용하지 않으면 이러한 모든 채널이 대기 상태가 됩니다. 채널 대기를 야기해도 기능 문제점은 발생하지 않지만 성능에 악영향을 미칠 수 있습니다. 대부분의 SVC에는 대기가 포함되므로 다음 SVC를 제외하고 대기를 방지해야 합니다.

- GETMAIN/FREEMAIN/STORAGE
- LOAD/DELETE

따라서 일반적으로 SVC, PC 및 I/O를 피하십시오. 대신에, MQXWAIT 호출을 사용하십시오.

- 엑시트는 첨부되는 하위 태스크 외에는 ESTAE 또는 SPIE를 발행하지 않습니다. 해당 오류 핸들링이 IBM MQ에서 수행되는 오류 핸들링을 방해할 수 있기 때문입니다. 이는 IBM MQ가 오류에서 복구할 수 없거나 엑시트 프로그램이 모든 오류 정보를 수신하지는 못한다는 것을 의미합니다.
 - MQXWAIT 호출(MQXWAIT 참조)은 I/O 및 기타 이벤트를 대기하는 대기 서비스를 제공합니다. 이 서비스가 사용되면 엑시트는 연계 스택을 사용하지 않아야 합니다.
- 비차단 기능이나 ECB를 제공하지 않는 I/O 및 기타 기능이 대기하도록 하려면, 별도의 하위 태스크에 연결해서 MQXWAIT로 해당 태스크가 완료될 때까지 대기해야 합니다. 이 기술이 발생시키는 처리 때문에 보안 엑시트에서만 이 기능을 사용해야 합니다.
- MQDISC MQI 호출은 엑시트 프로그램 내에서 암시적 커밋을 발생시키지 않습니다. 채널 프로세스의 커밋은 채널 프로토콜이 지시할 때에만 수행됩니다.

다음 엑시트 예제는 IBM MQ for z/OS와 함께 제공됩니다.

CSQ4BAX0

이 샘플은 어셈블러로 기록되었으며 MQXWAIT 사용을 보여줍니다.

CSQ4BCX1 및 CSQ4BCX2

이러한 샘플은 C로 기록되었으며 매개변수 액세스 방법을 보여줍니다.

CSQ4BCX3 및 CSQ4BAX3

이러한 샘플은 각각 C 및 어셈블러로 기록되었습니다.

CSQ4BCX3 샘플(SCSQAUTH LOADLIB로 사전 컴파일됨)은 엑시트 자체를 기반으로 변경할 필요 없이 작동해야 합니다. LOADLIB(예를 들면, MY.TEST.LOADLIB라고 함)를 작성하고 여기에 SCSQAUTH(CSQ4BCX3) 멤버를 복사할 수 있습니다.

클라이언트 연결의 보안 엑시트를 설정하려면 다음 프로시저를 수행하십시오.

1. 채널 시작기가 사용하는 사용자 ID에 유효한 OMVS 세그먼트를 설정하십시오.

이를 통해 채널 생성자는 IBM MQ for z/OS 종료 처리를 원활하게 진행하기 위해 (z/OS UNIX) 소켓 인터페이스와 z/OS UNIX System Services 함께 `TCP/IP`를 사용할 수 있습니다. 연결 클라이언트의 사용자 ID에 대해 OMVS 세그먼트를 정의할 필요가 없다는 점에 유의하십시오.

2. 엑시트 코드 자체가 프로그램 제어 환경에서만 실행되는지 확인하십시오.

이는 CHINIT 주소 공간에 로드된 모든 정보를 프로그램 제어 라이브러리 (STEPLIB의 모든 라이브러리를 의미함) 및 CSQXLIB에 이름 지정된 모든 라이브러리에서 로드해야 한다는 의미입니다.

```
++hlq++.SCSQANLx
++hlq++.SCSQMVR1
++hlq++.SCSQAUTH
```

로드 라이브러리를 제어된 프로그램으로 설정하려면 이 예제와 비슷한 명령을 사용하십시오.

```
RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
```

그리고 다음 명령을 실행하여 프로그램 제어 환경을 활성화하거나 새로 고칠 수 있습니다.

```
SETROPTS WHEN(PROGRAM) REFRESH
```

3. 다음 명령을 실행하여 CHINIT 시작 프로시저의 CSQXLIB DD에 엑시트 LOADLIB를 추가하십시오.

```
ALTER CHANNEL(xxxx) CHLTYPE(SVRCONN)SCYEXIT(CSQ4BCX3)
```

이는 이름 지정된 채널의 엑시트를 활성화합니다.

4. 외부 보안 관리자(ESM)는 프로그램 제어되는 다른 라이브러리를 나열합니다. 하지만 ESM 또는 C 라이브러리가 프로그램 제어를 받아야 한다는 점에 유의하십시오.

샘플 CSQ4BCX3을 사용한 보안 엑시트 설정에 대한 추가 정보는 [IBM MQ for z/OS 서버 연결 채널](#)을 참조하십시오.

CSQ4BCX4

이 샘플은 C로 기록되었으며 MQCXP에서 **RemoteProduct** 및 **RemoteVersion** 필드를 사용하는 방법에 대해 설명합니다.

관련 개념

[715 페이지의 『IBM i에서 채널 엑시트 프로그램 작성』](#)

다음 정보를 사용하면 IBM i용 채널 엑시트 프로그램을 작성하고 컴파일하는 데 도움이 됩니다.

[716 페이지의 『AIX, Linux, and Windows에서의 채널 엑시트 프로그램 작성』](#)

다음 정보를 사용하면 AIX, Linux, and Windows 시스템의 채널 엑시트 프로그램을 작성하는 데 도움이 될 수 있습니다.

관련 참조

[IBM MQ for z/OS 서버 연결 채널](#)

IBM i IBM i에서 채널 엑시트 프로그램 작성

다음 정보를 사용하면 IBM i용 채널 엑시트 프로그램을 작성하고 컴파일하는 데 도움이 됩니다.

엑시트는 ILE C, ILE RPG 또는 ILE COBOL 언어로 기록된 프로그램 오브젝트입니다. 엑시트 프로그램 이름과 해당 라이브러리는 채널 정의에서 이름 지정됩니다.

엑시트 프로그램을 작성하여 컴파일할 때 다음 조건을 준수하십시오.

- 프로그램은 스레드에 안전해야 하며 ILE C, ILE RPG 또는 ILE COBOL 컴파일러로 작성되어야 합니다. ILE RPG의 경우 THREAD(*SERIALIZE) 제어 스펙을 지정해야 하고, ILE COBOL의 경우 PROCESS문의 THREAD 옵션에 대해 SERIALIZE를 지정해야 합니다. 또한 프로그램을 스레드된 IBM MQ 라이브러리에 바인딩해야 합니다. ILE C 및 ILE RPG의 경우에는 QMQM/LIBMQM_R에 바인딩하고 ILE COBOL의 경우에는 AMQ0STUB_R에 바인딩해야 합니다. RPG 또는 COBOL 애플리케이션을 스레드 안전 상태로 설정하는 방법에 대해서는 해당 언어용 프로그래머 안내서를 참조하십시오.

- IBM MQ for IBM i에서는 테라스페이스 지원을 위해 엑시트 프로그램을 사용으로 설정해야 합니다. (테라스페이스는 OS/400 V4R4에 도입된 공유 메모리의 양식입니다.) ILE RPG 및 COBOL 컴파일러의 경우, OS/400 V4R4 이상에서 컴파일된 모든 프로그램이 사용 가능합니다. C의 경우, CRTCMOD 또는 CRTBNDC 명령에 TERASPACE(*YES *TSIFC) 옵션을 지정하여 프로그램을 컴파일해야 합니다.
- 자체 버퍼 공간에 대한 포인터를 리턴하는 엑시트는 가리킨 오브젝트가 채널 엑시트 프로그램의 시간 범위를 벗어나 있는지 확인해야 합니다. 포인터는 프로그램 스택의 변수 주소일 수 없고 프로그램 힙(heap)의 변수일 수도 없습니다. 대신 시스템에서 포인터를 확보해야 합니다. 한 예로 사용자 엑시트에서 작성된 사용자 공간이 있습니다. 채널 엑시트 프로그램이 할당한 데이터 영역이 MCA에서 여전히 사용 가능한지 확인하려면, 호출자의 활성화 그룹 또는 이름 지정된 활성화 그룹에서 채널 엑시트를 실행해야 합니다. CRTPGM의 ACTGRP 매개 변수를 사용자 정의 값이나 *CALLER로 설정하여 이를 실행하십시오. 이런 방식으로 프로그램을 작성할 경우 채널 엑시트 프로그램에서 동적 메모리를 할당하고 이 메모리의 포인터를 MCA에 다시 전달할 수 있습니다.

관련 개념

[716 페이지의 『AIX, Linux, and Windows에서의 채널 엑시트 프로그램 작성』](#)

다음 정보를 사용하면 AIX, Linux, and Windows 시스템의 채널 엑시트 프로그램을 작성하는 데 도움이 될 수 있습니다.

[713 페이지의 『z/OS에서 채널 엑시트 프로그램 작성』](#)

다음 정보를 사용하면 z/OS용 채널 엑시트 프로그램을 작성하고 컴파일하는 데 도움이 됩니다.

ALW AIX, Linux, and Windows에서의 채널 엑시트 프로그램 작성

다음 정보를 사용하면 AIX, Linux, and Windows 시스템의 채널 엑시트 프로그램을 작성하는 데 도움이 될 수 있습니다.

688 페이지의 『AIX, Linux, and Windows에 엑시트 및 설치 가능 서비스 작성』에 설명된 지시사항을 따르십시오. 적절한 경우, 다음 채널 엑시트 특정 정보를 사용하십시오.

엑시트는 C로 기록되어야 하며, Windows의 DLL입니다.

엑시트에서 더미 MQStart() 루틴을 정의하고 라이브러리의 시작점으로 MQStart를 지정하십시오. [716 페이지의 그림 79](#)에서는 프로그램에 입력 항목을 설정하는 방법을 보여줍니다.

```
#include <cmqec.h>

void MQStart() {} /* dummy entry point - for consistency only */
void MQENTRY ChannelExit ( PMQCXP  pChannelExitParms,
                           PMQCD   pChannelDefinition,
                           PMQLONG pDataLength,
                           PMQLONG pAgentBufferLength,
                           PMQVOID pAgentBuffer,
                           PMQLONG pExitBufferLength,
                           PMQPTR  pExitBufferAddr)
{
    ... Insert code here
}
```

그림 79. 채널 엑시트에 대한 샘플 소스 코드

Visual C++를 사용하여 Windows에 대한 채널 엑시트를 작성할 때에는 사용자 고유의 DEF 파일을 작성해야 합니다. 방법의 예는 [716 페이지의 그림 80](#)에 표시되어 있습니다. 채널 엑시트 프로그램 작성에 대한 추가 정보는 [712 페이지의 『채널 엑시트 프로그램 작성』](#)의 내용을 참조하십시오.

```
EXPORTS
ChannelExit
```

그림 80. Windows용 샘플 DEF 파일

관련 개념

[715 페이지의 『IBM i에서 채널 엑시트 프로그램 작성』](#)

다음 정보를 사용하면 IBM i용 채널 엑시트 프로그램을 작성하고 컴파일하는 데 도움이 됩니다.

[713 페이지의 『z/OS에서 채널 엑시트 프로그램 작성』](#)

다음 정보를 사용하면 z/OS용 채널 엑시트 프로그램을 작성하고 컴파일하는 데 도움이 됩니다.

채널 보안 엑시트 프로그램

보안 엑시트 프로그램을 사용하여 채널의 다른 끝에 있는 파트너가 진짜인지 확인할 수 있습니다. 이를 인증이라고 합니다.

채널이 보안 출구를 사용해야 함을 지정하려면 채널 정의의 필드에 **SCYEXIT** 출구 이름을 지정하십시오.

참고: 인증은 채널 인증 레코드로도 얻을 수 있습니다. 채널 인증 레코드는 특정 사용자 및 채널에서 큐 관리자에 대한 액세스를 방지하고 원격 사용자를 IBM MQ 사용자 ID로 매핑할 때에 뛰어난 유연성을 제공합니다. 또한 IBM MQ가 사용자를 인증하고 데이터에 대한 암호화 및 데이터 무결성 검사를 제공하도록 TLS 지원도 제공됩니다. TLS에 대한 자세한 정보는 IBM MQ의 TLS 보안 프로토콜을 참조하십시오. 단, 보다 복잡한 양식(또는 다른 양식)의 보안 처리와, 다른 유형의 검사 및 보안 컨텍스트 설정이 여전히 필요한 경우에는 보안 엑시트 작성을 고려하십시오.

제목 및 발행인 DN 속성은 다음과 같은 채널 상태 속성에 표시됩니다.

- SSLPEER(PCF 선택자 MQCACH_SSL_SHORT_PEER_NAME)
- SSLCERTI(PCF 선택자 MQCACH_SSL_CERT_ISSUER_NAME)

이러한 값은 다음과 같이 나열된 채널 보안 엑시트에 전달된 데이터뿐만 아니라 채널 상태 명령에 의해서도 리턴됩니다.

- MQCD SSLPeerNamePtr
- MQCXP SSLRemCertIssNamePtr

보안 엑시트는 C 또는 Java로 기록될 수 있습니다.

채널 보안 엑시트 프로그램은 MCA 처리 순환의 다음 위치에서 호출됩니다.

- MCA 시작 및 종료 시
- 채널 시동 시 초기 데이터 조정이 완료된 직후, 채널의 수신자 또는 서버 끝은 리모트 끝에 있는 보안 엑시트로 전달될 메시지를 제공하여 리모트 끝과 보안 메시지 교환을 시작할 수 있습니다. 또한 이를 거부할 수도 있습니다. 엑시트 프로그램이 다시 시작되어 리모트 끝에서 수신된 보안 메시지를 처리합니다.
- 채널 시동 시 초기 데이터 조정이 완료된 직후, 채널의 송신자 또는 요청자 끝은 리모트 끝에서 수신된 보안 메시지를 처리하거나, 리모트 끝이 수신할 수 없을 때 보안 교환을 시작합니다. 엑시트 프로그램이 다시 시작되어 수신될 수 있는 모든 후속 보안 메시지를 처리합니다.

요청자 채널은 MQXR_INIT_SEC로 호출되지 않습니다. 채널은 보안 엑시트 프로그램이 있다는 것을 서버에 알리고, 서버는 보안 엑시트를 시작하는 기회를 갖습니다. 기회가 없는 경우, 요청자에게 알리고 길이가 0인 플로우가 엑시트 프로그램으로 리턴됩니다.

참고: 길이가 0인 보안 메시지는 보내지 마십시오.

보안 엑시트 프로그램에서 교환된 데이터의 예는 그림 718 페이지의 그림 81 - 720 페이지의 그림 84에 예시되어 있습니다. 이 예에서는 수신자의 보안 엑시트 및 송신자의 보안 엑시트와 관련되어 발생하는 이벤트의 순서를 보여줍니다. 그림에서 연속 행은 시간의 경과를 나타냅니다. 일부 경우에는 수신자 및 송신자의 이벤트가 상관 관계가 없으므로 동시에 또는 서로 다른 시간에 발생할 수 있습니다. 다른 경우, 한 엑시트 프로그램의 이벤트로 인해 나중에 다른 엑시트 프로그램에서 보완적인 이벤트가 발생합니다. 예를 들어, 718 페이지의 그림 81에서는 다음과 같습니다.

1. 수신자 및 송신자는 각각 MQXR_INIT로 호출되지만, 이러한 호출은 상관 관계가 없으므로 동시에 또는 서로 다른 시간에 발생할 수 있습니다.
2. 수신자는 다음에 MQXR_INIT_SEC로 호출되지만, 송신자 엑시트에서 보완적인 이벤트를 요구하지 않는 MQXCC_OK를 리턴합니다.
3. 송신자는 다음에 MQXR_INIT_SEC로 호출됩니다. 이는 MQXR_INIT_SEC를 사용한 수신자의 호출과 상관 관계가 없습니다. 송신자는 수신자 엑시트에서 보완적인 이벤트를 발생시키는 MQXCC_SEND_SEC_MSG를 리턴합니다.
4. 그런 다음 수신자는 MQXR_SEC_MSG로 호출되며, 송신자 엑시트에서 보완적인 이벤트를 발생시키는 MQXCC_SEND_SEC_MSG를 리턴합니다.
5. 그런 다음 송신자는 MQXR_SEC_MSG로 호출되며, 수신자 엑시트에서 보완적인 이벤트를 요구하지 않는 MQXCC_OK를 리턴합니다.

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG	
<i>Message transfer begins</i>	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK

그림 81. 동의에 따른 송신자 시작 교환

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION
<i>Channel closes</i>	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

그림 82. 동의에 따르지 않은 송신자 시작 교환

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
<i>Message transfer begins</i>	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

그림 83. 동의에 따른 수신자 시작 교환

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION	
<i>Channel closes</i>	

그림 84. 동의에 따르지 않은 수신자 시작 교환

보안 엑시트가 생성한 보안 데이터를 포함하는 에이전트 버퍼(전송 헤더 제외)에 채널 보안 엑시트 프로그램이 전달됩니다. 이 데이터는 적당한 데이터일 수 있으므로 채널의 양 끝이 보안 유효성 검증을 수행할 수 있습니다. 메시지 채널의 송신 및 수신 끝 모두에 있는 보안 엑시트 프로그램은 임의의 호출에 대해 다음 두 응답 코드 중 하나를 리턴할 수 있습니다.

- 오류 없이 보안 교환 종료
- 채널 억제 및 닫기

참고:

1. 채널 보안 엑시트는 일반적으로 쌍으로 작동합니다. 적절한 채널을 정의할 때, 채널의 양 끝에 대해 호환 가능한 엑시트 프로그램 이름이 지정되었는지 확인하십시오.
2.  IBM i에서 Use adopted authority(USEADPAUT=*YES)로 컴파일된 보안 엑시트 프로그램은 QMQM 또는 QMQMADM 권한을 채택할 수 있습니다. 엑시트가 이 기능을 사용하여 시스템에 보안 위험을 제기하지 않도록 주의하십시오.
3. 채널의 다른 끝이 인증서를 제공하는 TLS 채널에서 보안 엑시트는 SSLPeerNamePtr에 의해 액세스되는 MQCD 필드에 있는 이 인증서 제목의 식별 이름과, SSLRemCertIssNamePtr에 의해 액세스되는 MQCXP 필드에 있는 발행인의 식별 이름을 수신합니다. 이 이름을 넣을 수 있는 용도는 다음과 같습니다.
 - TLS 채널에 대한 액세스를 제한합니다.
 - 이름에 따라 MQCD.MCAUserIdentifier를 변경합니다.

관련 개념

[TLS\(Transport Layer Security\) 개념](#)

관련 참조

[채널 인증 레코드](#)

보안 엑시트 작성

보안 엑시트 스케레톤 코드를 사용하여 보안 엑시트를 작성할 수 있습니다.

[721 페이지의 그림 85](#)에서는 보안 엑시트를 작성하는 방법을 보여줍니다.

```
void MQENTRY MQStart() {}
void MQENTRY EntryPoint (PMQVOID pChannelExitParms,
                        PMQVOID pChannelDefinition,
                        PMQLONG pDataLength,
                        PMQLONG pAgentBufferLength,
                        PMQVOID pAgentBuffer,
                        PMQLONG pExitBufferLength,
                        PMQPTR pExitBufferAddr)
{
    PMQCXP pParms = (PMQCXP)pChannelExitParms;
    MQCD pChDef = (MQCD)pChannelDefinition;
    /* TODO: Add Security Exit Code Here */
}
```

그림 85. 보안 엑시트 스케레톤 코드

표준 IBM MQ Entry Point MQStart는 존재해야 하지만 기능을 수행하는 데에는 필요하지 않습니다. 함수의 이름(이 예제에서는 EntryPoint)은 변경할 수 있지만, 라이브러리를 컴파일 및 링크할 때에는 함수를 내보내야 합니다. 이전 예제와 같이 pChannelExitParms 포인터는 PMQCXP로, pChannelDefinition 포인터는 MQCD로 캐스팅해야 합니다. 채널 엑시트 호출 및 매개변수 사용에 대한 일반 정보는 [MQ_CHANNEL_EXIT](#)를 참조하십시오. 이러한 매개변수는 다음과 같이 보안 엑시트에서 사용됩니다.

PMQVOID pChannelExitParms

입출력(I/O)

MQCXP 구조에 대한 포인터 - 필드를 액세스하기 위해 PMQCXP로 캐스팅합니다. 이 구조는 엑시트와 MCA 사이에서 통신하는 데 사용됩니다. MQCXP의 다음 필드는 보안 엑시트에 특히 중요합니다.

ExitReason

보안 엑시트에 보안 교환의 현재 상태를 알리며, 수행할 조치를 결정할 때 사용됩니다.

ExitResponse

보안 교환의 다음 스테이지를 지시하는 MCA에 대한 응답입니다.

ExitResponse2

MCA에서 보안 엑시트 응답을 해석하는 방법을 관리하는 추가 제어 플래그입니다.

ExitUserArea

보안 엑시트에서 호출 간의 상태를 유지보수하기 위해 사용할 수 있는 16바이트(최대값)의 스토리지입니다.

ExitData

채널 정의의 SCYDATA 필드에 지정된 데이터(오른쪽에 공백을 채워 넣은 32바이트)를 포함합니다.

PMQVOID pChannelDefinition

입출력(I/O)

MQCD 구조에 대한 포인터 - 필드에 액세스하기 위해 PMQCD로 캐스팅합니다. 이 매개변수는 채널의 정의를 포함합니다. MQCD의 다음 필드는 보안 엑시트에 특히 중요합니다.

ChannelName

채널 이름(오른쪽에 공백을 채워 넣은 20바이트)입니다.

ChannelType

채널 유형을 정의하는 코드입니다.

MCA 사용자 ID

이 그룹의 세 필드는 채널 정의에 지정된 MCAUSER 필드 값으로 초기화됩니다. 보안 엑시트에서 이러한 필드에 지정한 사용자 ID는 액세스 제어에 사용됩니다(SDR, SVR, CLNTCONN 또는 CLUSSDR 채널에 적용할 수 없음).

MCAUserIdentifier

오른쪽에 공백을 채워 넣은 ID의 처음 12바이트입니다.

LongMCAUserIdPtr

전체 길이 ID를 포함하는 버퍼에 대한 포인터(널(Null) 종료를 보장하지 않음)은 MCAUserIdentifier보다 우선합니다.

LongMCAUserIdLength

LongMCAUserIdPtr에서 가리키는 문자열의 길이 - LongMCAUserIdPtr이 설정된 경우 설정해야 합니다.

원격 사용자 ID

LNTCONN/SVRCONN 채널 쌍에만 적용됩니다. CLNTCONN 보안 엑시트가 정의되지 않으면 이러한 세 필드는 클라이언트 MCA에 의해 초기화되므로, MCA 사용자 ID를 지정할 때와 인증할 때에 SVRCONN 보안 엑시트에서 사용할 수 있는 클라이언트 환경의 사용자 ID가 해당 필드에 포함될 수 있습니다. CLNTCONN 보안 엑시트가 정의되어 있으면 이러한 필드는 초기화되지 않고 CLNTCONN 보안 엑시트에서 설정할 수 있습니다. 또는 보안 메시지를 사용하여 클라이언트에서 서버로 사용자 ID를 전달할 수 있습니다.

RemoteUserIdentifier

오른쪽에 공백을 채워 넣은 ID의 처음 12바이트입니다.

LongRemoteUserIdPtr

전체 길이 ID를 포함하는 버퍼에 대한 포인터(널(Null) 종료를 보장하지 않음)은 RemoteUserIdentifier보다 우선합니다.

LongRemoteUserIdLength

LongRemoteUserIdPtr에서 가리키는 문자열의 길이 - LongRemoteUserIdPtr이 설정된 경우 설정해야 합니다.

PMQLONG pDataLength

입출력(I/O)

PMQLONG에 대한 포인터입니다. 보안 엑시트 호출 시 AgentBuffer에 포함된 보안 엑시트의 길이를 포함합니다. AgentBuffer 또는 ExitBufferMust에서 보안 엑시트가 송신하는 메시지의 길이로 설정되어야 합니다.

PMQLONG pAgentBufferLength

입력

MQLONG에 대한 포인터입니다. 보안 엑시트 호출 시 AgentBuffer에 포함된 데이터의 길이입니다.

PMQVOID pAgentBuffer

입출력(I/O)

보안 엑시트 호출 시, 파트너 엑시트에서 송신된 메시지를 지정합니다. MQCXP 구조의 ExitReason2에 MQXR2_USE_AGENT_BUFFER 플래그 세트(기본값)가 있는 경우(기본값) 보안 엑시트는 송신하는 메시지 데이터를 가리키도록 이 매개변수를 설정해야 합니다.

PMQLONG pExitBufferLength

입출력(I/O)

MQLONG에 대한 포인터입니다. 이 매개변수는 보안 엑시트의 처음 호출 시 0으로 초기화되며, 리턴된 값은 보안 교환 시 보안 엑시트 호출 사이에 유지보수됩니다.

PMQPTR pExitBufferAddr

입출력(I/O)

이 매개변수는 보안 엑시트의 처음 호출 시 널 포인터로 초기화되며, 리턴된 값은 보안 교환 시 보안 엑시트 호출 사이에 유지보수됩니다. MQXR2_USE_EXIT_BUFFER 플래그가 MQCXP 구조의 ExitReason2에 설정되면, 보안 엑시트는 송신하는 메시지 데이터를 가리키도록 이 매개변수를 설정해야 합니다.

CLNTCONN/SVRCONN 채널 쌍과 기타 채널 쌍에 정의된 보안 엑시트 간의 작동 차이점

보안 엑시트는 모든 유형의 채널에 정의할 수 있습니다. 그러나 CLNTCONN/SVRCONN 채널 쌍에 정의된 보안 엑시트의 작동은 다른 채널 쌍에 정의된 보안 엑시트와는 조금 차이가 있습니다.

정의된 SVRCONN 보안 엑시트가 없고 SVRCONN의 MCAUSER 필드가 설정되어 있지 않은 경우, CLNTCONN 채널의 보안 엑시트는 OAM 권한 부여나 파트너 SVRCONN 엑시트의 처리에 필요한 채널 정의에 원격 사용자 ID를 설정할 수 있습니다.

정의되어 있는 CLNTCONN 보안 종료는 없는 경우, 채널 정의의 원격 사용자 ID는 클라이언트 MCA에 의해 클라이언트 환경의 사용자 ID(공백일 수 있음)로 설정됩니다.

CLNTCONN 및 SVRCONN 채널 쌍에 정의된 보안 엑시트 사이의 보안 교환은 SVRCONN 보안 엑시트가 MQXCC_OK의 ExitResponse를 리턴할 때 완료됩니다. 기타 채널 쌍 사이의 보안 교환은 교환을 시작한 보안 엑시트가 MQXCC_OK의 ExitResponse를 리턴할 때 완료됩니다.

그러나 MQXCC_SEND_AND_REQUEST_SEC_MSG ExitResponse 코드는 보안 교환을 연속적으로 강제 실행할 때 사용할 수 있습니다. MQXCC_SEND_AND_REQUEST_SEC_MSG의 ExitResponse가 CLNTCONN 또는 SVRCONN 보안 엑시트에 의해 리턴되면 파트너 엑시트는 보안 메시지(MQXCC_OK 또는 널 응답이 아님) 송신으로 응답해야 합니다. 그렇지 않으면 채널이 종료됩니다. 다른 유형의 채널에 정의된 보안 엑시트의 경우, 파트너 보안 엑시트에서 MQXCC_SEND_AND_REQUEST_SEC_MSG에 응답하여 MQXCC_OK의 ExitResponse가 리턴되면 결과적으로 널 응답이 리턴된 것처럼 보안 교환이 연속적으로 발생하고 채널이 종료되지 않습니다.

SSPI 보안 엑시트

IBM MQ for Windows는 보안 서비스 프로그래밍 인터페이스(SSPI)를 사용하여 IBM MQ 채널에 대해 인증하는 보안 엑시트를 제공합니다. SSPI는 Windows의 통합 보안 기능을 제공합니다.

이 보안 엑시트는 IBM MQ 클라이언트와 IBM MQ 서버 모두에서 사용됩니다.

보안 패키지는 security.dll 또는 secur32.dll에서 로드됩니다. 이러한 DLL은 운영 체제와 함께 제공됩니다.

단방향 인증은 Windows에서 NTLM 인증 서비스를 사용하여 제공됩니다. 양방향 인증은 Windows 2000에서 Kerberos 인증 서비스를 사용하여 제공됩니다.

보안 엑시트 프로그램은 소스 및 오브젝트 형식으로 제공됩니다. 오브젝트 코드를 그대로 사용하거나, 소스 코드를 고유 사용자 엑시트 프로그램 작성의 시작 지점으로 사용할 수 있습니다. SSPI 보안 엑시트의 오브젝트 또는 소스 코드를 사용하는 방법에 대한 자세한 정보는 1041 페이지의 『Windows에서 SSPI 보안 엑시트 사용』의 내용을 참조하십시오.

채널 송신 및 수신 엑시트 프로그램

송신 및 수신 엑시트를 사용하여 데이터 압축 및 압축 해제와 같은 태스크를 수행할 수 있습니다. 연속으로 실행할 송신 및 수신 엑시트 프로그램의 목록을 지정할 수 있습니다.

채널 송신 및 수신 엑시트 프로그램은 MCA의 처리 순환의 다음 위치에서 호출됩니다.

- 송신 및 수신 엑시트 프로그램은 MCA 시작 시 초기화용으로 호출되고 MCA 종료 시 종료용으로 호출됩니다.
- 하나의 메시지 전송에서 전송물이 송신되는 끝에 따라, 링크를 통해 전송물을 보내기 직전에 채널의 한쪽 끝 또는 다른 끝에서 송신 엑시트 프로그램이 호출됩니다. 참고 4에서는 메시지 채널이 한 방향으로만 메시지를 보내더라도 엑시트가 양방향에서 사용 가능한 이유에 대해 설명합니다.
- 하나의 메시지 전송에서 전송물이 수신되는 끝에 따라, 링크로부터 전송물을 받은 직후에 채널의 한쪽 끝 또는 다른 끝에서 수신 엑시트 프로그램이 호출됩니다. 참고 4에서는 메시지 채널이 한 방향으로만 메시지를 보내더라도 엑시트가 양방향에서 사용 가능한 이유에 대해 설명합니다.

하나의 메시지 전송에 여러 전송물이 있을 수 있고, 메시지가 수신 끝의 메시지 엑시트에 도착하기 전에 송신 및 수신 엑시트 프로그램을 여러 번 반복할 수 있습니다.

채널 송신 및 수신 엑시트 프로그램은 통신 링크로부터 송신 또는 수신된 전송 데이터를 포함하여 에이전트 버퍼에 전달됩니다. 송신 엑시트 프로그램의 경우, 버퍼의 처음 8바이트는 MCA 전용으로 예약되어 있으므로 변경해서는 안 됩니다. 프로그램이 다른 버퍼를 리턴하면, 이러한 처음 8바이트는 새 버퍼에 있어야 합니다. 엑시트 프로그램에 제공되는 데이터의 형식은 정의되지 않습니다.

송신 및 수신 엑시트 프로그램은 적절한 응답 코드를 리턴해야 합니다. 다른 응답은 MCA 이상종료를 야기합니다.

참고: 송신 또는 수신 엑시트의 동기점 내에서는 MQGET, MQPUT 또는 MQPUT1 호출을 발행하지 마십시오.

참고:

1. 송신 및 수신 엑시트는 일반적으로 쌍으로 작동합니다. 예를 들어, 송신 엑시트가 데이터를 압축하고 수신 엑시트가 압축 해제하거나, 송신 엑시트가 데이터를 암호화하고 수신 엑시트가 복호화할 수 있습니다. 적절한 채널을 정의할 때, 채널의 양 끝에 대해 호환 가능한 엑시트 프로그램 이름이 지정되었는지 확인하십시오.
2. 채널에 대해 압축이 켜진 경우, 엑시트에 압축된 데이터가 전달됩니다.
3. 채널 송신 및 수신 엑시트는 애플리케이션 데이터(예: 상태 메시지) 이외의 메시지 세그먼트에 대해 호출할 수 있습니다. 시동 대화상자나 보안 검사 단계에서는 호출되지 않습니다.
4. 메시지 채널은 메시지를 한 방향으로만 보내지만, 채널 제어 데이터(예: 하트비트 및 배치 처리 종료)는 양방향으로 이동하므로 이러한 엑시트도 양방향에서 사용 가능합니다. 그러나 초기 채널 시동 데이터 플로우 중 일부는 엑시트의 처리에서 면제됩니다.
5. 송신 및 수신 엑시트가 순서를 벗어나 호출될 수도 있습니다. 예를 들어, 사용자가 일련의 엑시트 프로그램을 실행하거나 보안 엑시트를 실행하는 경우가 이에 해당합니다. 그러면 수신 엑시트가 먼저 데이터를 처리하도록 요청받을 때 해당 송신 엑시트를 통해 전달되지 않은 데이터를 수신할 수 있습니다. 수신 엑시트가 압축 해제와 같은 조작이 필요한지 먼저 검사하지 않고 해당 조작을 수행한 경우 예상치 못한 결과가 나타납니다.

수신 엑시트가 수신 중인 데이터를 해당 송신 엑시트에서 처리했는지 확인할 수 있도록 송신 및 수신 엑시트를 코딩해야 합니다. 이렇게 하기 위해 권장되는 방법은 엑시트 프로그램을 코딩하여 다음을 수행하는 것입니다.

- 조작을 수행하기 전에 송신 엑시트는 데이터의 9번째 바이트 값을 0으로 설정하고 모든 데이터를 1바이트 앞으로 이동합니다. (처음 8바이트는 MCA에서 사용하도록 예약됩니다.)
- 수신 엑시트는 바이트 9에 0이 있는 데이터를 수신할 경우 해당 데이터가 송신 엑시트에서 생성된 것이라고 인식합니다. 그래서 0을 제거하고 보완 조작을 수행하고 결과 데이터를 1바이트 뒤로 이동합니다.
- 수신 엑시트는 바이트 9에 0이 아닌 다른 값이 있는 데이터를 수신할 경우 송신 엑시트가 실행되지 않은 것으로 가정하고 데이터를 변경하지 않고 호출자에게 다시 보냅니다.

보안 엑시트를 사용할 때 채널이 보안 엑시트에 의해 종료되면, 해당 수신 엑시트 없이 송신 엑시트가 호출될 수 있습니다. 이런 문제점을 방지하는 한 가지 방법은 예를 들어, 엑시트가 채널을 끝내기로 결정할 때 MQCD.SecurityUserData 또는 MQCD.SendUserData에서 플래그를 설정하는 보안 엑시트를 코딩하는 것입니다. 그런 다음 송신 엑시트는 이 필드를 검사하고 플래그가 설정되지 않은 경우에만 데이터를 처리해야 합니다. 이 검사는 송신 엑시트가 불필요하게 데이터를 대체하지 못하게 하여 보안 엑시트가 대체된 데이터를 수신하는 경우에 발생할 수 있는 모든 변환 오류를 방지합니다.

채널 송신 엑시트 프로그램 - 공간 예약

전송 전에 송신 및 수신 엑시트를 사용하여 데이터를 변환할 수 있습니다. 채널 송신 엑시트 프로그램은 전송 버퍼에서 공간을 예약하여 변환에 대한 고유 데이터를 추가할 수 있습니다.

이 데이터는 수신 엑시트 프로그램으로 처리된 후 버퍼에서 제거됩니다. 예를 들어, 데이터를 암호화하고 복호화 용 보안 키를 추가할 수도 있습니다.

공간 예약 및 사용 방법

초기화를 위해 송신 엑시트 프로그램이 호출될 때, MQXCP의 *ExitSpace* 필드를 예약 바이트 수로 설정하십시오. 세부사항은 MQXCP를 참조하십시오. *ExitSpace*는 초기화 중에만 즉, *ExitReason*의 값이 MQXR_INIT 일 때만 설정할 수 있습니다. 송신 엑시트가 전송 직전에 호출되고 *ExitReason*이 MQXR_XMIT로 설정되면 전송 버퍼에서 *ExitSpace*바이트가 예약됩니다. *ExitSpace*는 z/OS에서 지원되지 않습니다.

송신 엑시트는 모든 예약된 공간을 사용할 필요가 없습니다. 이 엑시트는 *ExitSpace* 바이트 미만을 사용하거나, 전송 버퍼가 가득 차지 않은 경우 예약된 크기 이상을 사용할 수 있습니다. *ExitSpace* 값을 설정할 때 전송 버퍼에 최소 1KB의 메시지 데이터를 남겨 두어야 합니다. 대용량 데이터에 예약된 공간을 사용하면 채널 성능이 영향을 받을 수 있습니다.

전송 버퍼의 길이는 일반적으로 32KB입니다. 그러나 채널이 TLS를 사용하는 경우, 전송 버퍼 크기는 RFC 6101 및 관련 TLS 표준에 정의된 최대 레코드 길이에 맞게 15,352바이트로 감축됩니다. IBM MQ에 사용하도록 1024바이트가 추가로 예약되므로, 송신 엑시트에서 사용할 수 있는 최대 전송 버퍼 공간은 14,328바이트입니다.

채널의 수신 끝에서 발생하는 상황

채널의 수신 엑시트 프로그램은 해당 송신 엑시트와 호환 가능하도록 설정해야 합니다. 수신 엑시트는 예약된 공간의 바이트 수를 알고 있어야 하고 해당 공간에서 데이터를 제거해야 합니다.

다중 송신 엑시트

연속으로 실행할 송신 및 수신 엑시트 프로그램의 목록을 지정할 수 있습니다. IBM MQ는 모든 전송 엑시트에 의해 예약된 공간의 합계를 유지보수합니다. 이 총 공간은 전송 버퍼에 최소 1KB의 메시지 데이터를 남겨 두어야 합니다.

다음 예제에서는 연속적으로 호출된 세 개의 송신 엑시트에 대한 공간 할당 방법을 보여줍니다.

1. 초기화를 위해 호출 시:

- 송신 엑시트 A는 1KB를 예약합니다.
- 송신 엑시트 B는 2KB를 예약합니다.
- 송신 엑시트 C는 3KB를 예약합니다.

2. 최대 전송 크기는 32KB이고 사용자 데이터 길이는 5KB입니다.

3. 엑시트 A는 5KB의 데이터로 호출되며, 5KB가 엑시트 B 및 C에 예약되어 있기 때문에 최대 27KB가 사용 가능합니다. 엑시트 A가 예약한 크기인 1KB를 추가합니다.

4. 엑시트 B는 6KB의 데이터로 호출되며, 3KB가 엑시트 C에 예약되어 있기 때문에 최대 29KB가 사용 가능합니다. 엑시트 B가 예약된 2KB보다 작은 1KB를 추가합니다.

5. 엑시트 C는 7KB의 데이터로 호출되며, 최대 32KB를 사용할 수 있습니다. 엑시트 C는 예약된 3KB를 초과하는 10KB를 추가합니다. 총 데이터 크기인 17KB가 최대값 32KB 미만이기 때문에 이 크기는 유효합니다.

TLS를 사용하는 채널의 최대 전송 버퍼 크기는 32KB가 아니라 15,352바이트입니다. 이는 기본 보안 소켓 전송 세그먼트가 16KB로 제한되고 일부 공간이 TLS 레코드 오버헤드에 필요하기 때문입니다. IBM MQ에 사용하도록 1024바이트가 추가로 예약되므로, 송신 엑시트에서 사용할 수 있는 최대 전송 버퍼 공간은 14,328바이트입니다.

채널 메시지 엑시트 프로그램

채널 메시지 엑시트를 사용하여 태스크(예: 링크에 대한 암호화, 수신 사용자 ID의 유효성 검증 또는 대체, 메시지 데이터 변환, 저널링 및 참조 메시지 핸들링)를 수행할 수 있습니다. 연속으로 실행할 메시지 엑시트 프로그램의 목록을 지정할 수 있습니다.

채널 메시지 엑시트 프로그램은 MCA 처리 순환의 다음 위치에서 호출됩니다.

- MCA 시작 및 종료 시

- 송신 MCA가 MQGET 호출을 발행한 직후
- 수신 MCA가 MQPUT 호출을 발행하기 전

메시지 엑시트는 큐에서 검색될 때 전송 큐 헤더 MQXQH 및 애플리케이션 메시지 텍스트를 포함하는 에이전트 버퍼로 전달됩니다. MQXQH의 형식은 MQXQH - 전송 큐 헤더에서 지정됩니다.

Multi 참조 메시지(즉 송신되는 다른 오브젝트를 나타내는 헤더만 들어있는 메시지)를 사용하는 경우, 메시지 엑시트는 헤더 MQRMH를 인식합니다. 해당 객체를 식별하고, 적절한 방식으로 이를 검색한 후 헤더에 추가한 다음, 수신 MCA로 전송하기 위해 MCA로 전달합니다. 수신 MCA에서 다른 메시지 엑시트는 이 메시지가 참조 메시지라고 인식하고, 오브젝트를 추출하며, 헤더를 목적지 큐에 전달합니다. 참조 메시지와 이를 핸들링하는 몇 가지 샘플 메시지 엑시트에 대한 자세한 정보는 810 페이지의 『참조 메시지 및 대용량 객체 전송』 및 1015 페이지의 『참조 메시지 샘플 실행』의 내용을 참조하십시오.

메시지 엑시트는 다음 응답을 리턴할 수 있습니다.

- 메시지를 송신합니다(GET 엑시트). 메시지가 엑시트에 의해 변경되었을 수 있습니다. (이는 MQXCC_OK를 리턴합니다.)
- 메시지를 큐에 넣습니다(PUT 엑시트). 메시지가 엑시트에 의해 변경되었을 수 있습니다. (이는 MQXCC_OK를 리턴합니다.)
- 메시지를 처리하지 않습니다. 메시지가 MCA에 의해 데드-레터 큐(미전달 메시지 큐)에 배치됩니다.
- 채널을 닫습니다.
- MCA의 비정상 종료를 야기하는 잘못된 리턴 코드입니다.

참고:

1. 메시지가 부분으로 분할될 때에도 전송된 모든 완료 메시지에 대해 메시지 엑시트가 한 번만 호출됩니다.
2.   AIX 또는 Linux에서 메시지 엑시트를 제공하는 경우, (여기에서 설명한 대로) 사용자 ID를 소문자로 자동 변환하지 않습니다.
3. 엑시트는 MCA 자체와 동일한 스레드에서 실행됩니다. 또한 동일한 연결 핸들을 사용하기 때문에 MCA와 동일한 작업 단위(UOW) 안에서 실행됩니다. 그러므로 동기점에서 수행된 모든 호출은 배치 마지막에서 채널에 의해 커밋 또는 백아웃됩니다. 예를 들어, 한 채널 메시지 엑시트 프로그램은 알림 메시지를 다른 채널 메시지 엑시트 프로그램에 보낼 수 있고, 이러한 메시지는 원래 메시지를 포함하는 배치가 커밋된 경우에만 큐에 커밋됩니다.

그러므로 채널 메시지 엑시트 프로그램에서 동기점 MQI 호출을 발행할 수 있습니다.

메시지 엑시트 외부의 메시지 변환

메시지 엑시트를 호출하기 전에, 수신 MCA는 메시지에 대한 일부 변환을 수행합니다. 이 주제에서는 변환을 수행하는 데 사용되는 알고리즘에 대해 설명합니다.

처리되는 헤더

메시지 엑시트가 호출되기 전에 변환 루틴이 수신자의 MCA에서 실행됩니다. 변환 루틴은 메시지 처음에 있는 MQXQH 헤더부터 시작합니다. 그런 다음, 필요한 위치에서 변환을 수행하여 MQXQH 뒤에 체인 연결된 헤더를 통해 변환 루틴을 처리합니다. 체인 연결된 헤더는 수신자의 메시지 엑시트로 전달되는 MQCXP 데이터의 HeaderLength 매개변수에 포함된 오프셋 이상으로 확장할 수 있습니다. 다음 헤더가 적절하게 변환됩니다.

- MQXQH(형식 이름 "MQXMIT")
- MQMD(이 헤더는 MQXQH의 일부이고 형식 이름이 없음)
- MQMDE(형식 이름 "MQHMDE")
- MQDH(형식 이름 "MQHDIST")
- MQWIH(형식 이름 "MQHWIH")

다음 헤더는 변환되지 않고, MCA가 체인 연결된 헤더 처리를 계속하는 동안 스텝 오버됩니다.

- MQDLH(형식 이름 "MQDEAD")

- 별도의 언급이 없이 형식 이름이 3개의 문자 'MQH'로 시작하는 모든 헤더(예: "MQHRF")

헤더 처리 방법

MCA에서 각 IBM MQ 헤더의 형식 매개변수를 읽습니다. Format 매개변수는 헤더 내에서 8바이트이며 이름을 포함한 8개의 1바이트 문자입니다.

그런 다음 MCA는 각 헤더 뒤의 데이터를 이름 지정된 유형으로 해석합니다. Format이 IBM MQ 데이터 변환에 적합한 헤더 유형의 이름이면 변환됩니다. 비MQ 데이터(예: MQFMT_NONE 또는 MQFMT_STRING)를 표시하는 다른 이름이면, MCA는 헤더 처리를 중지합니다.

MQCXP HeaderLength의 개념

메시지 엑시트에 제공된 MQCXP 데이터의 HeaderLength 매개변수는 메시지 처음에 있는 MQXQH(MQMD 포함), MQMDE 및 MQDH 헤더의 총 길이입니다. 이러한 헤더는 'Format' 이름 및 길이를 사용하여 체인으로 연결됩니다.

MQWIH

체인 연결된 헤더는 HeaderLength를 넘어서 사용자 데이터 영역까지 확장할 수 있습니다. MQWIH 헤더는 존재하는 경우 HeaderLength 이상으로 표시되는 헤더 중 하나입니다.

체인 연결된 헤더에 MQWIH 헤더가 있는 경우, 수신자의 메시지 엑시트가 호출되기 전에 해당 헤더가 적절히 변환됩니다.

채널 메시지 재시도 엑시트 프로그램

채널 메시지 재시도 엑시트는 대상 큐를 열려는 시도가 실패할 때 호출됩니다. 사용자는 엑시트를 사용하여 재시도할 상황, 재시도하는 횟수 및 재시도의 빈도수를 판별할 수 있습니다.

또한 이 엑시트는 MCA 시작 및 종료 시에 채널의 수신 끝에서도 호출됩니다.

채널 메시지 재시도 엑시트에는 큐에서 검색될 때 전송 큐 헤더 MQXQH 및 애플리케이션 메시지 텍스트를 포함한 에이전트 버퍼가 전달됩니다. MQXQH의 형식은 [MQXQH의 개요](#)에 나와 있습니다.

이 엑시트는 모든 이유 코드에 대해 호출되며, MCA가 재시도할 이유 코드, 재시도 횟수 및 재시도 간격 등을 판별합니다. (채널이 정의되었을 때 설정된 메시지 재시도 수의 값은 MQCD의 엑시트에 전달되지만, 엑시트는 이 값을 무시할 수 있습니다.)

MQCXP의 MsgRetryCount 필드는 엑시트를 호출할 때마다 MCA에 의해 증분되며, 엑시트는 MQCXP의 MsgRetryInterval 필드에 대기 시간이 포함된 MQXCC_OK나, MQXCC_SUPPRESS_FUNCTION을 리턴합니다. 엑시트가 MQCXP의 ExitResponse 필드에서 MQXCC_SUPPRESS_FUNCTION을 리턴할 때까지 재시도가 끊임 없이 지속됩니다. MCA가 이러한 완료 코드에 대해 수행한 조치 관련 정보는 [MQCXP](#)의 내용을 참조하십시오.

모든 재시도가 실패하는 경우 메시지가 데드-레터 큐에 기록됩니다. 사용 가능한 데드-레터 큐가 없는 경우 채널이 중지됩니다.

채널의 메시지 재시도 엑시트를 정의하지 않았고 임시적인 실패(예: MQRC_Q_FULL)가 발생한 경우, MCA는 채널이 정의되었을 때 설정된 메시지 재시도 수 및 메시지 재시도 간격을 사용합니다. 보다 더 영구적인 네이처로 인해 실패하고 이를 핸들링하도록 엑시트 프로그램을 정의하지 않은 경우 메시지가 데드-레터 큐에 기록됩니다.

채널 자동 정의 엑시트 프로그램

채널 자동 정의 엑시트는 수신자 또는 서버 연결 채널을 시작하라는 요청이 수신되지만 해당 채널에 대한 정의가 존재하지 않을 때 사용할 수 있습니다(IBM MQ for z/OS용은 아님). 또한 클러스터-송신자 및 클러스터-수신자 채널의 모든 플랫폼에서 이 엑시트를 호출하여 채널의 인스턴스에 대한 정의를 수정할 수 있습니다.

수신자 또는 서버 연결 채널을 시작하라는 요청이 수신되지만 채널 정의가 존재하지 않을 때 z/OS를 제외한 모든 플랫폼에서 채널 자동 정의 엑시트를 호출할 수 있습니다. 이를 사용하면 자동으로 정의된 수신자 또는 서버 연결 채널, SYSTEM.AUTO.RECEIVER 또는 SYSTEM.AUTO.SVRCON에 대해 제공된 기본 정의를 수정할 수 있습니다. 채널 정의를 자동으로 작성할 수 있는 방법에 대한 설명은 [채널 준비](#)를 참조하십시오.

또한 채널 자동 정의 엑시트는 클러스터-송신자 채널을 시작하라는 요청이 수신될 때 호출할 수 있습니다. 이는 클러스터-송신자 및 클러스터-수신자 채널이 채널의 이 인스턴스에 대한 정의를 수정할 수 있도록 호출될 수 있습니다. 이 경우에는 엑시트가 IBM MQ for z/OS에도 적용됩니다. 엑시트 이름은 여러 플랫폼에서 형식이 서로

다르기 때문에, 메시지 엑시트(MSGEXIT, RCVEXIT, SCYEXIT 및 SENDEXIT)의 이름을 변경하기 위한 용도로 채널 자동 정의 엑시트가 가장 많이 사용됩니다. 채널 자동 정의 엑시트를 지정하지 않으면, z/OS에서 기본적으로 `[path]/libraryname(function)` 양식의 분산 엑시트 이름을 조사하고 최대 8문자의 함수(존재하는 경우) 또는 라이브러리 이름을 사용하도록 작동합니다. z/OS에서 채널 자동 정의 엑시트 프로그램은 MsgExit, MsgUserData, SendExit, SendUserData, ReceiveExit 및 ReceiveUserData 필드 자체가 아니라 MsgExitPtr, MsgUserDataPtr, SendExitPtr, SendUserDataPtr, ReceiveExitPtr 및 ReceiveUserDataPtr로 지정된 필드를 대체해야 합니다.

자세한 정보는 [자동 정의된 채널에 대한 작업](#)을 참조하십시오.

다른 채널 엑시트와 마찬가지로, 매개변수 목록은 다음과 같습니다.

```
MQ_CHANNEL_AUTO_DEF_EXIT (ChannelExitParms, ChannelDefinition)
```

ChannelExitParms는 [MQCXP](#)에서 설명합니다. ChannelDefinition은 [MQCD](#)에서 설명합니다.

MQCD는 기본 채널 정의에 사용된 값을 포함합니다(엑시트가 이 값을 대체하지 않은 경우). 엑시트는 필드의 서브셋만 수정할 수 있습니다([MQ_CHANNEL_AUTO_DEF_EXIT](#) 참조). 그러나 다른 필드를 변경하려 해도 오류가 야기되지는 않습니다.

채널 자동 정의 엑시트는 MQXCC_OK 또는 MQXCC_SUPPRESS_FUNCTION의 응답을 리턴합니다. 이들 응답 중 어느 것도 리턴되지 않으면 MCA는 MQXCC_SUPPRESS_FUNCTION이 리턴된 것처럼 처리를 계속합니다. 즉, 자동 정의가 취소되고 새로운 채널 정의가 작성되지 않으며 채널을 시작할 수 없습니다.

AIX, Linux, and Windows 시스템에서 채널 엑시트 프로그램 컴파일

다음 예제는 AIX, Linux, and Windows 시스템의 채널 엑시트 프로그램을 컴파일하는 데 도움이 됩니다.

필수 라이브러리

사용 가능한 32비트 및 64비트 라이브러리 목록은 [외부 라이브러리](#) 항목을 참조하십시오.

Windows

 Windows

Windows에서 채널 엑시트 프로그램의 컴파일러 및 링커 명령은 다음과 같습니다.

```
cl.exe /Ic:\mqm\tools\c\include /nologo /c myexit.c
link.exe /nologo /dll myexit.obj /def:myexit.def /out:myexit.dll
```

AIX and Linux 시스템

 Linux

 AIX

이 예제에서 exit는 라이브러리 이름이며 ChannelExit는 함수 이름입니다. AIX에서 내보내기 파일은 exit.exp입니다. 이 이름은 채널 정의가 [MQCD - 채널 정의](#)에 설명된 형식을 사용한 엑시트 프로그램을 참조하는 데 사용됩니다. [DEFINE CHANNEL](#) 명령의 MSGEXIT 매개변수도 참조하십시오.

 AIX

AIX에서 채널 엑시트에 대한 샘플 컴파일러 및 링커 명령:

```
$ xlc_r -q64 -e MQStart -bE:exit.exp -bM:SRE -o /var/mqm/exits64/exit
exit.c -I/usr/mqm/inc
```

 Linux

큐 관리자가 32비트인 Linux에서 채널 엑시트에 대한 샘플 컴파일러 및 링커 명령:

```
$ gcc -shared -fPIC -o /var/mqm/exits/exit exit.c -I/opt/mqm/inc
```

 Linux

큐 관리자가 64비트인 Linux에서의 채널 엑시트에 대한 샘플 컴파일러 및 링커 명령:

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
```

클라이언트에서 32비트 또는 64비트 엑시트를 사용할 수 있습니다. 이 엑시트는 mqic_r에 링크되어야 합니다.

AIX에서 IBM MQ에 의해 호출되는 모든 함수를 내보내야 합니다. 이 작성 파일에 대한 샘플 내보내기 파일:

```
#
!channelExit
MQStart
```

채널 엑시트 구성

채널 엑시트를 호출하려면 채널 정의에서 이름을 지정해야 합니다.

채널 엑시트는 채널 정의에서 이름을 지정해야 합니다. 채널을 처음 정의할 때 이 이름을 지정할 수 있으며, 또는 예를 들어 MQSC 명령을 사용하여 나중에 **ALTER CHANNEL** 정보를 추가할 수 있습니다. 또한 MQCD 채널 데이터 구조에서 채널 엑시트 이름을 제공할 수도 있습니다. 엑시트 이름의 형식은 IBM MQ 플랫폼에 따라 다릅니다. 정보는 MQCD 또는 MQSC 명령을 참조하십시오.

채널 정의에 사용자 엑시트 프로그램 이름이 없으면 사용자 엑시트가 호출되지 않습니다.

채널 자동 정의 엑시트는 개별 채널이 아닌 큐 관리자의 특성입니다. 이 엑시트를 호출하려면 큐 관리자 정의에서 이름을 지정해야 합니다. 큐 관리자 정의를 변경하려면 MQSC 명령을 사용하십시오 **ALTER QMGR**.

데이터 변환 엑시트 작성

이 주제 컬렉션에는 데이터 변환 엑시트 작성 방법에 대한 정보가 있습니다.

참고: VSE/ESA용 MQSeries에서는 지원되지 않습니다.

MQPUT 수행 시, 애플리케이션은 메시지의 메시지 디스크립터(MQMD)를 작성합니다. IBM MQ는 작성되는 플랫폼에 상관없이 MQMD의 콘텐츠를 이해할 필요가 있기 때문에 시스템에 의해 자동으로 변환됩니다.

그러나 애플리케이션 데이터는 자동으로 변환되지 않습니다. CodedCharSetId 및 Encoding 필드가 다른 플랫폼 사이(예: ASCII와 EBCDIC 사이)에서 문자 데이터를 교환할 경우, 애플리케이션은 메시지의 변환에 대해 준비해야 합니다. 애플리케이션 데이터 변환은 큐 관리자 자체 또는 데이터 변환 엑시트라고 하는 사용자 엑시트 프로그램에 의해 수행될 수 있습니다. 큐 관리자는 애플리케이션 데이터가 내장 형식(예: MQFMT_STRING) 중 하나인 경우 큐 관리자의 내장 변환 루틴 중 하나를 사용하여 데이터 변환 자체를 수행할 수 있습니다. 이 주제에는 애플리케이션 데이터가 내장 형식이 아닌 경우에 IBM MQ가 제공하는 데이터 변환 엑시트 기능에 대한 정보가 포함되어 있습니다.

MOGET 호출 중에 제어를 데이터 변환 엑시트로 전달할 수 있습니다. 이는 최종 목적지에 도달하기 전에 다른 플랫폼 사이의 변환을 방지합니다. 하지만 최종 목적지가 MOGET에 대한 데이터 변환을 지원하지 않는 플랫폼인 경우, 데이터를 최종 목적지로 송신하는 송신자 채널에 CONVERT(YES)를 지정해야 합니다. 이렇게 하면 IBM MQ가 전송 중에 데이터를 변환합니다. 이러한 경우에 데이터 변환 엑시트는 송신자 채널이 정의된 시스템에 상주해야 합니다.

MQGET 호출은 애플리케이션에서 직접 발행됩니다. MQMD의 CodedCharSetId 및 Encoding 필드를 필요한 문자 세트 및 인코딩으로 설정하십시오. 애플리케이션이 큐 관리자와 동일한 문자 세트 및 인코딩을 사용하는 경우, CodedCharSetId를 MQCCSI_Q_MGR로 설정하고 Encoding을 MQENC_NATIVE로 설정하십시오. MQGET 호출이 완료된 후 이러한 필드는 리턴된 메시지 데이터에 적절한 값을 가집니다. 이러한 값은 변환이 성공적이지 않은 경우 필요한 값과 다를 수 있습니다. 애플리케이션은 이러한 필드를 각 MOGET 호출 전에 필요한 값으로 재설정해야 합니다.

MQGET의 MQGET 호출에 대해 데이터 변환 엑시트 호출에 필요한 조건이 정의되어 있습니다.

데이터 변환 엑시트에 전달된 매개변수에 대한 설명 및 자세한 사용 참고사항은 MQ_DATA_CONV_EXIT 호출 및 MQDXP 구조의 데이터 변환을 참조하십시오.

다른 시스템 인코딩과 CCSID 사이의 애플리케이션 데이터를 변환시키는 프로그램은 IBM MQ 데이터 변환 인터페이스(DCI)에 따라야만 합니다.

멀티캐스트 클라이언트의 경우, API 엑시트 및 데이터 변환 엑시트는 일부 메시지가 큐 관리자를 통해 전달되지 않을 수 있기 때문에 클라이언트 측에서 실행될 수 있어야 합니다. 다음 라이브러리는 서버 패키지와 마찬가지로 클라이언트 패키지의 일부입니다.

표 119. 클라이언트 및 서버 패키지에 있는 라이브러리	
운영 체제	라이브러리
 AIX	32비트 및 64비트: libmqm.a 및 libmqm_r.a
 IBM i	LIBMQM 및 LIBMQM_R
 Linux	32비트 및 64비트: libmqm.so 및 libmqm_r.so
 Windows	32비트 및 64비트: mqm.dll 및 mqm.pdb

데이터 변환 엑시트 호출

데이터 변환 엑시트는 MQGET 호출을 처리하는 동안 제어를 수신하는 사용자 작성 엑시트입니다.

다음 명령문이 true이면 엑시트가 호출됩니다.

- MQGMO_CONVERT 옵션이 MQGET 호출에 지정됩니다.
- 메시지 데이터의 일부 또는 전부가 요청된 문자 세트 또는 인코딩에 없습니다.
- 메시지와 연관된 MQMD 구조의 *Format* 필드가 MQFMT_NONE이 아닙니다.
- MQGET 호출에 지정된 *BufferLength*가 0이 아닙니다.
- 메시지 데이터 길이가 0이 아닙니다.
- 메시지에 사용자 정의 형식의 데이터가 포함되어 있습니다. 사용자 정의 형식이 전체 메시지에 사용되거나, 하나 이상의 내장 형식이 선행될 수 있습니다. 예를 들면, 사용자 정의 형식 앞에 MQFMT_DEAD_LETTER_HEADER 형식이 표시될 수 있습니다. 사용자 정의 형식만 변환하도록 엑시트가 호출되므로, 큐 관리자는 사용자 정의 형식에 선행하는 모든 내장 형식을 변환합니다.

또한 사용자 작성 엑시트는 내장 형식을 변환하기 위해 호출될 수 있지만, 내장 변환 루틴이 내장 형식을 정상적으로 변환시킬 수 없는 경우에만 발생합니다.

기타 몇 가지 조건이 있으며, 이들은 [MQ_DATA_CONV_EXIT](#)에서 MQ_DATA_CONV_EXIT 호출의 사용 시 참고사항에 충분히 설명되어 있습니다.

MQGET 호출의 세부사항은 [MQGET](#)의 내용을 참조하십시오. 데이터 변환 엑시트는 MQXCNVC 이외의 MQI 호출을 사용할 수 없습니다.

애플리케이션이 큐 관리자에 연결된 이후에 해당 *Format*을 사용하는 첫 번째 메시지를 검색하려고 시도할 때 엑시트의 새 사본이 로드됩니다. 또한 큐 관리자가 이전에 로드된 사본을 제거한 경우, 다른 시간에 새 사본을 로드할 수 있습니다.

데이터 변환 엑시트는 MQGET 호출을 발행한 프로그램의 환경과 같은 환경에서 실행됩니다. 사용자 애플리케이션과 마찬가지로, 프로그램은 메시지 변환을 지원하지 않는 목적지 큐 관리자에 메시지를 보내는 메시지 채널 에이전트(MCA)일 수 있습니다. 환경은 해당되는 경우 주소 공간 및 사용자 프로파일을 포함합니다. 엑시트는 큐 관리자의 환경에서 실행되지 않기 때문에 큐 관리자의 무결성을 손상시키지 않습니다.

z/OS의 데이터 변환



z/OS에서는 다음에 유의하십시오.

- 엑시트 프로그램은 어셈블리 언어로만 작성할 수 있습니다.
- 엑시트 프로그램은 재입력 가능해야 하고, 스토리지의 임의의 위치에서 실행 가능해야 합니다.
- 엑시트 프로그램은 종료 시 환경을 시작 시 환경으로 복원해야 하고 확보된 스토리지가 있으면 비워야 합니다.
- 엑시트 프로그램은 대기(WAIT)하거나 ESTAE 또는 SPIE를 발행해서는 안 됩니다.
- 엑시트 프로그램은 일반적으로 다음 상태에서 z/OS LINK에 의한 것처럼 호출됩니다.
 - 비승인 문제점의 프로그램 상태
 - 1차 주소 공간 제어 모드

- 비교차 메모리 모드
 - 비엑세스 등록 모드
 - 31비트 주소 지정 모드
 - TCB-PRB 모드
 - CICS 애플리케이션에서 사용될 때 엑시트는 EXEC CICS LINK에 의해 호출되고 CICS 프로그래밍 변환을 따라야 합니다. 매개변수는 CICS 통신 영역(COMMAREA)에서 포인터(주소)에 의해 전달됩니다.
- 권장되지는 않지만, 사용자 엑시트 프로그램은 다음 주의사항에 따라 CICS API 호출을 사용할 수도 있습니다.
- 결과가 MCA에서 선언한 작업 단위에 영향을 미칠 수 있으므로, 동기점을 발행하지 마십시오.
 - CICS Transaction Server에 의해 제어되는 자원을 포함하여 IBM MQ for z/OS이외의 자원 관리자에 의해 제어되는 자원을 갱신하지 마십시오.
- CONVERT=YES인 채널의 경우, 엑시트는 CSQXLIB DD문에서 참조하는 데이터 세트에서 로드됩니다. IBM MQ CICS 브릿지에 대한 MQ 제공 엑시트 CSQCBDCI 및 CSQCBDCO는 SCSQAUTH에 있습니다.

IBM i IBM i의 데이터 변환 엑시트 프로그램 작성

데이터 변환 종료 프로그램을 IBM MQ 작성할 때 고려해야 할 IBM i단계들.

다음 단계를 수행하십시오.

1. 메시지 형식의 이름을 지정하십시오. 이 이름은 MQMD의 *Format* 필드에 맞아야 합니다. *Format* 이름은 앞에 임베드된 공백이 없어야 하며, 뒤 공백은 무시됩니다. 오브젝트의 이름에는 8자를 초과하지 않는 공백이 아닌 문자가 있어야 하는데, 그 이유는 *Format*의 길이가 8자이기 때문입니다. 메시지를 송신할 때마다 이 이름을 사용하는 것을 기억하십시오. (예제에서 사용된 이름은 *Format*입니다.)
 2. 메시지를 나타내는 구조를 작성하십시오. 예제는 올바른 구문을 참조하십시오.
 3. CVTMQMMDTA 명령을 통해 이 구조를 실행하여 데이터 변환 엑시트에 대한 코드 단편을 작성하십시오.
- CVTMQMMDTA 명령에 의해 생성된 함수는 파일 QMQM/H(AMQSVMHA)로 배송되는 매크로를 사용합니다. 이러한 매크로는 모든 구조가 팩형이라고 가정하여 기록되며, 이에 해당되지 않는 경우 해당 매크로가 수정됩니다.
4. 제공된 스켈레톤 소스 파일, QMQMSAMP/QCSRC(AMQSVFC4)의 사본을 작성하고 해당 이름을 바꾸십시오. (예제에서 사용된 이름은 EXIT_MOD입니다.)
 5. 소스 파일에서 다음 주석 상자를 찾아 설명된 대로 코드를 삽입하십시오.
 - a. 소스 파일의 끝 부분에서 주석 상자가 다음으로 시작됩니다.

```
/* Insert the functions produced by the data-conversion exit */
```

여기에 731 페이지의 『3』 단계에서 생성된 코드 단편을 삽입하십시오.

- b. 소스 파일의 중간 근처에서 주석 상자가 다음으로 시작됩니다.

```
/* Insert calls to the code fragments to convert the format's */
```

이 다음에는 ConverttagSTRUCT 함수에 대한 주석 처리된 호출이 따릅니다.

함수의 이름을 731 페이지의 『5.a』 단계에서 추가한 함수의 이름으로 변경하십시오. 주석 문자를 제거하여 함수를 활성화하십시오. 여러 함수가 있을 경우 각 함수에 대한 호출을 작성하십시오.

- c. 소스 파일의 시작 부분 근처에서 주석 상자가 다음으로 시작됩니다.

```
/* Insert the function prototypes for the functions produced by */
```

여기에 731 페이지의 『5.a』 단계에서 추가된 함수의 함수 프로토타입 명령문을 삽입하십시오.

메시지에 문자 데이터가 포함되어 있으면 생성되는 코드가 MQXCNCV를 호출합니다. 이는 서비스 프로그램 QMQM/LIBMQM을 바인딩하여 해결할 수 있습니다.

6. 다음과 같이 소스 모듈 EXIT_MOD를 컴파일하십시오.

```
CRTCMOD MODULE(library/EXIT_MOD) +
SRCFILE(QCSRC) +
TERASPACE(*YES *TSIFC)
```

7. 프로그램을 작성/링크하십시오.

비스레드 애플리케이션의 경우 다음을 사용하십시오.

```
CRTPGM PGM(library/Format) +
MODULE(library/EXIT_MOD) +
BNDSRVPGM(QMQM/LIBMQM) +
ACTGRP(QMQM) +
USRPRF(*USER)
```

기본 환경에 대한 데이터 변환 엑시트를 작성하는 것 외에도 다른 엑시트가 스레드 환경에서 필요합니다. 이 로드 가능한 오브젝트 뒤에 **_R**을 표시해야 합니다. MQXCNVNC에 대한 호출을 해결하려면 **LIBMQM_R** 라이브러리를 사용하십시오. 로드 가능한 두 오브젝트 모두 스레드 환경에 필요합니다.

```
CRTPGM PGM(library/Format_R) +
MODULE(library/EXIT_MOD) +
BNDSRVPGM(QMQM/LIBMQM_R) +
ACTGRP(QMQM) +
USRPRF(*USER)
```

8. IBM MQ 작업에 대한 라이브러리 목록에 출력을 배치하십시오. 프로덕션을 위해 데이터 변환 엑시트 프로그램을 QSYS에 저장하는 것이 좋습니다.

참고:

1. CVTMQMDTA가 팩형 구조를 사용하는 경우, 모든 IBM MQ 애플리케이션은 **_Packed** 규정자를 사용해야 합니다.
2. 데이터 변환 엑시트 프로그램은 재입력 가능해야 합니다.
3. MQXCNVNC는 데이터 변환 엑시트에서 발행할 수 있는 MQI 호출입니다.
4. 엑시트가 사용자의 권한으로 실행되도록 하려면, 사용자 프로파일 컴파일러 옵션을 ***USER**로 설정하여 엑시트 프로그램을 컴파일하십시오.
5. IBM MQ for IBM i의 모든 사용자 엑시트에 대해서는 테라스페이스 메모리를 사용해야 합니다. CRTCMOD 및 CRTBNDC 명령에서 TERASPACE(*YES *TSIFC)를 지정하십시오.

IBM MQ for z/OS의 데이터 변환 엑시트 프로그램 작성

.NET 데이터 변환 종료 IBM MQ for z/OS 프로그램 작성 시 고려해야 할 단계

다음 단계를 수행하십시오.

1. 제공된 소스 스켈레톤 CSQ4BAX9(비CICS 환경의 경우) 또는 CSQ4CAX9(CICS의 경우)를 시작점으로 사용하십시오.
2. CSQUCVX 유틸리티를 실행하십시오.
3. 변환시킬 메시지에서 구조가 발생하는 순서대로 CSQUCVX 유틸리티에 의해 생성된 루틴을 통합하려면 CSQ4BAX9 또는 CSQ4CAX9의 프롤로그에 있는 지시사항을 따르십시오.
4. 유틸리티는 데이터 구조가 팩형이 아니고 데이터를 암시적으로 맞출 수 있으며 구조가 필요에 따라 바이트를 건너뛰고(올바른 구문의 예제에서 ID와 VERSION 사이처럼) 완전 단어 경계에서 시작된다고 가정합니다. 구조가 팩형이면 생성되는 CMQXCALA 매크로를 생략하십시오. 따라서 모든 필드 이름이 지정되고 바이트를 건너뛰지 않도록 구조를 선언하는 방안을 고려하십시오. 즉, 올바른 구문의 예제에서 ID와 VERSION 사이에 "MQBYTE DUMMY;" 필드를 추가하십시오.
5. 입력 버퍼가 변환할 메시지 형식보다 짧으면 제공된 엑시트는 오류를 리턴합니다. 엑시트가 가능한 많은 완료 필드를 변환하더라도 오류로 인해 변환되지 않은 메시지가 애플리케이션으로 리턴됩니다. 짧은 입력 버퍼가 최대한 변환될 수 있게 하려면 CSQXCDFD 매크로의 TRUNC= 값을 YES로 변경하십시오. 이렇게 하면 오류가 리턴되지 않으므로, 애플리케이션은 변환된 메시지를 수신합니다. 애플리케이션은 잘림을 핸들링해야 합니다.
6. 필요한 다른 특수 처리 코드를 추가하십시오.
7. 프로그램 이름을 데이터 형식 이름으로 바꾸십시오.

8. 배치 애플리케이션 프로그램과 같은 프로그램을 컴파일하고 링크 편집하십시오(CICS 애플리케이션과 함께 사용하지 않을 경우). 유틸리티에 의해 생성된 코드의 매크로는 **thlqual.SCSQMACS** 라이브러리에 있습니다.

메시지에 문자 데이터가 포함되어 있으면 생성된 코드는 MQXCNCV를 호출합니다. 엑시트가 이 호출을 사용하는 경우 엑시트 스텝 프로그램 CSQASTUB으로 링크 편집하십시오. 스텝은 언어 독립적이고 환경 독립적입니다. 또는 동적 호출 이름 CSQXCNCV를 사용하여 스텝을 동적으로 로드할 수 있습니다. 자세한 정보는 963 페이지의 『동적으로 IBM MQ 스텝 호출』의 내용을 참조하십시오.

링크 편집된 모듈을 애플리케이션 로드 라이브러리와, 채널 시작기로 시작된 태스크 프로시저의 CSQXLIB DD문에서 참조하는 데이터 세트에 배치하십시오.

9. 엑시트를 CICS 애플리케이션에서 사용하는 경우 필요하면 CSQASTUB을 포함하여 CICS 애플리케이션 프로그램과 같이 컴파일하고 링크 편집하십시오. CICS 애플리케이션 프로그램 라이브러리에 배치하십시오. 일반적인 방법으로 CICS에 프로그램을 정의하고 정의에는 EXECKEY(CICS)를 지정하십시오.

참고: LE/370 런타임 라이브러리가 CSQUCVX 유틸리티를 실행하는 데에는 필요하지만(732 페이지의 『2』 단계 참조), 데이터 변환 엑시트 자체를 링크 편집하거나 실행하는 데에는 필요하지 않습니다(733 페이지의 『8』 단계 및 733 페이지의 『9』 단계 참조).

- IMSIBM MQ 브리지 내 데이터 변환에 관한 정보는 를 참조하십시오 75 페이지의 『IMS 브릿지 애플리케이션 작성』.

IBM MQ for AIX or Linux 시스템에 대한 데이터 변환 엑시트 작성

시스템용 IBM MQ for AIX or Linux 데이터 변환 종료 프로그램 작성 시 고려해야 할 단계.

다음 단계를 수행하십시오.

1. 메시지 형식의 이름을 지정하십시오. 이 이름은 MQMD의 *Format* 필드에 맞아야 하며 대문자여야 합니다 (예: MYFORMAT). *Format* 이름에는 앞 공백이 없어야 합니다. 뒤 공백은 무시됩니다. 오브젝트의 이름에는 8자를 초과하지 않는 공백이 아닌 문자가 있어야 하는데, 그 이유는 *Format*의 길이가 8자이기 때문입니다. 메시지를 송신할 때마다 이 이름을 사용하는 것을 기억하십시오.

데이터 변환 엑시트가 스레드 환경에서 사용된 경우 로드 가능한 오브젝트가 스레드 버전임을 나타내려면 해당 오브젝트 뒤에 _r을 표시해야 합니다.
2. 메시지를 나타내는 구조를 작성하십시오. 예제는 올바른 구문을 참조하십시오.
3. crtmcvix 명령을 통해 이 구조를 실행하여 데이터 변환 엑시트에 대한 코드 단편을 작성하십시오.

crtmcvix 명령에 의해 생성된 함수는 모든 구조가 팩형이라고 가정하는 매크로를 사용하며, 이에 해당되지 않는 경우 해당 매크로를 수정합니다.
4. 제공된 스켈레톤 소스 파일을 복사하고 이름을 733 페이지의 『1』 단계에서 설정한 메시지 형식의 이름으로 바꾸십시오. 스켈레톤 소스 파일 및 해당 사본은 읽기 전용입니다.

스켈레톤 소스 파일은 amqsvfc0.c라고 합니다.
5. IBM MQ for AIX에서는 amqsvfc.exp라는 스켈레톤 내보내기 파일도 제공됩니다. 이 파일을 복사하고 이름을 MYFORMAT.EXP로 바꾸십시오.
6. 스켈레톤은 MQ_INSTALLATION_PATH/inc 디렉토리에 샘플 헤더 파일 amqsvmha.h를 포함시킵니다. 여기서 MQ_INSTALLATION_PATH는 IBM MQ가 설치되는 상위 레벨 디렉토리를 나타냅니다. 이 파일을 선택하려면 포함 경로가 이 디렉토리를 가리키는지 확인하십시오.

amqsvmha.h 파일에는 crtmcvix 명령에 의해 생성된 코드에 사용되는 매크로가 포함되어 있습니다. 변환할 구조에 문자 데이터가 포함된 경우, 이러한 매크로는 MQXCNCV를 호출합니다.
7. 소스 파일에서 다음 주석 상자를 찾아 설명된 대로 코드를 삽입하십시오.
 - a. 소스 파일의 끝 부분에서 주석 상자가 다음으로 시작됩니다.

```
/* Insert the functions produced by the data-conversion exit */
```

여기에 733 페이지의 『3』 단계에서 생성된 코드 단편을 삽입하십시오.

- b. 소스 파일의 중간 근처에서 주석 상자가 다음으로 시작됩니다.

```
/* Insert calls to the code fragments to convert the format's */
```

이 다음에는 ConverttagSTRUCT 함수에 대한 주석 처리된 호출이 따릅니다.

함수의 이름을 733 페이지의 『7.a』 단계에서 추가한 함수의 이름으로 변경하십시오. 주석 문자를 제거하여 함수를 활성화하십시오. 여러 함수가 있을 경우 각 함수에 대한 호출을 작성하십시오.

c. 소스 파일의 시작 부분 근처에서 주석 상자가 다음으로 시작됩니다.

```
/* Insert the function prototypes for the functions produced by */
```

여기에 733 페이지의 『3』 단계에서 추가된 함수의 함수 프로토타입 명령문을 삽입하십시오.

8. MQStart를 시작점으로 사용하여 엑시트를 공유 라이브러리로 컴파일하십시오. 이를 수행하려면 734 페이지의 『AIX and Linux 시스템의 데이터 변환 엑시트 컴파일』의 내용을 참조하십시오.
9. 엑시트 디렉토리에 출력을 배치하십시오. 기본 엑시트 디렉토리는 32비트 시스템의 경우 /var/mqm/exits이고 64비트 시스템의 경우 /var/mqm/exits64입니다. qm.ini 또는 mqclient.ini 파일에서 이러한 디렉토리를 변경할 수 있습니다. 이 경로는 각 큐 관리자에 대해 설정할 수 있으며 엑시트는 해당 경로에서만 검색할 수 있습니다.

참고:

1. crtmcvcx가 팩형 구조를 사용하는 경우, 모든 IBM MQ 애플리케이션을 이런 방식으로 컴파일해야 합니다.
2. 데이터 변환 엑시트 프로그램은 재입력 가능해야 합니다.
3. MQXCNCV는 데이터 변환 엑시트에서 발행할 수 있는 MQI 호출입니다.

 AIX and Linux 시스템의 데이터 변환 엑시트 컴파일
AIX and Linux 시스템에서 데이터 변환 엑시트를 컴파일하는 방법의 예입니다.

필수 라이브러리

사용 가능한 32비트 및 64비트 라이브러리 목록은 [외부 라이브러리](#) 항목을 참조하십시오.

모든 플랫폼에서 모듈의 시작점은 MQStart입니다.

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

AIX



다음 명령 중 하나를 실행하여 엑시트 소스 코드를 컴파일하십시오.

32비트 애플리케이션 스레드되지 않음

```
cc -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT \
MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

스레드됨

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

64비트 애플리케이션 스레드되지 않음

```
cc -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT \
MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

스레드됨

```
xlc_r -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT_r \
MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

Linux

Linux

다음 명령 중 하나를 실행하여 엑시트 소스 코드를 컴파일하십시오.

32비트 애플리케이션

스레드되지 않음

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c  
-I MQ_INSTALLATION_PATH/inc
```

스레드됨

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c  
-I MQ_INSTALLATION_PATH/inc
```

64비트 애플리케이션

스레드되지 않음

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c  
-I MQ_INSTALLATION_PATH/inc
```

스레드됨

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c  
-I MQ_INSTALLATION_PATH/inc
```

Windows

IBM MQ for Windows의 데이터 변환 엑시트 작성

.NET 데이터 변환 종료 IBM MQ for Windows 프로그램 작성 시 고려해야 할 단계

다음 단계를 수행하십시오.

1. 메시지 형식의 이름을 지정하십시오. 이 이름은 MQMD의 *Format* 필드에 맞아야 합니다. *Format* 이름에는 앞 공백이 없어야 합니다. 뒤 공백은 무시됩니다. 오브젝트의 이름에는 8자를 초과하지 않는 공백이 아닌 문자가 있어야 하는데, 그 이유는 *Format*의 길이가 8자이기 때문입니다.

샘플 디렉터리에는 'amqsvfcn.def'라는 DEF 파일도 함께 제공됩니다.

MQ_INSTALLATION_PATH\Tools\C\Samples *MQ_INSTALLATION_PATH*는 IBM MQ가 설치된 디렉토리입니다. 이 파일을 복사한 다음 이름을 바꿉니다(예: MYFORMAT.DEF로 변경). 작성할 DLL의 이름과 MYFORMAT.DEF에 지정된 이름이 동일한지 확인하십시오. MYFORMAT.DEF의 이름 FORMAT1을 새로운 형식 이름으로 덮어쓰십시오.

메시지를 송신할 때마다 이 이름을 사용하는 것을 기억하십시오.

2. 메시지를 나타내는 구조를 작성하십시오. 예제는 올바른 구문을 참조하십시오.
3. crtmqcvx 명령을 통해 이 구조를 실행하여 데이터 변환 엑시트에 대한 코드 단편을 작성하십시오.

CRTMQCVX 명령에 의해 생성된 함수는 모든 구조가 팩형이라고 가정하여 기록된 매크로를 사용하며, 이에 해당되지 않는 경우 해당 매크로를 수정합니다.

4. 제공된 스케레톤 소스 파일인 amqsvfc0.c를 복사하고 해당 이름을 735 페이지의 『1』 단계에서 설정한 메시지 형식의 이름으로 바꾸십시오.

amqsvfc0.c 는 에 있습니다 `MQ_INSTALLATION_PATH\Tools\C\Samples`. 여기서 `MQ_INSTALLATION_PATH` 는 IBM MQ 가 설치된 디렉터리입니다. (기본 설치 디렉터리는 `C:\Program Files\IBM\MQ`입니다.)

이 스킴레톤에는 `MQ_INSTALLATION_PATH\Tools\C\include` 디렉토리 내에 샘플 헤더 파일 `amqsvmha.h` 이 포함되어 있습니다. 이 파일을 선택하려면 포함 경로가 이 디렉토리를 가리키는지 확인하십시오.

`amqsvmha.h` 파일에는 `CRTMQCVX` 명령에 의해 생성된 코드에 사용되는 매크로가 포함되어 있습니다. 변환할 구조에 문자 데이터가 포함된 경우, 이러한 매크로는 `MQXCNCVC`를 호출합니다.

5. 소스 파일에서 다음 주석 상자를 찾아 설명된 대로 코드를 삽입하십시오.

- a. 소스 파일의 끝 부분에서 주석 상자가 다음으로 시작됩니다.

```
/* Insert the functions produced by the data-conversion exit */
```

여기에 735 페이지의 『3』 단계에서 생성된 코드 단편을 삽입하십시오.

- b. 소스 파일의 중간 근처에서 주석 상자가 다음으로 시작됩니다.

```
/* Insert calls to the code fragments to convert the format's */
```

이 다음에는 `ConverttagSTRUCT` 함수에 대한 주석 처리된 호출이 따릅니다.

함수의 이름을 736 페이지의 『5.a』 단계에서 추가한 함수의 이름으로 변경하십시오. 주석 문자를 제거하여 함수를 활성화하십시오. 여러 함수가 있을 경우 각 함수에 대한 호출을 작성하십시오.

- c. 소스 파일의 시작 부분 근처에서 주석 상자가 다음으로 시작됩니다.

```
/* Insert the function prototypes for the functions produced by */
```

여기에 735 페이지의 『3』 단계에서 추가된 함수의 함수 프로토타입 명령문을 삽입하십시오.

6. 다음 명령 파일을 작성하십시오.

```
cl -I MQ_INSTALLATION_PATH\Tools\C\Include -Tp \
MYFORMAT.C
```

```
MYFORMAT.DEF
```

여기서 `MQ_INSTALLATION_PATH`는 IBM MQ가 설치되어 있는 디렉토리입니다.

7. 명령 파일을 실행하여 엑시트를 DLL 파일로 컴파일하십시오.
8. IBM MQ 데이터 디렉토리 아래의 엑시트 서브디렉토리에 출력을 배치하십시오. 32비트 시스템에 엑시트를 설치하기 위한 기본 디렉토리는 `MQ_DATA_PATH\Exits`이고 64비트 시스템의 경우에는 `MQ_DATA_PATH\Exits64`입니다.

데이터 변환 엑시트 검색에 사용된 경로는 레지스트리에 제공됩니다. 다음은 레지스트리 폴더입니다.

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere
MQ\Installation\MQ_INSTALLATION_NAME\Configuration\ClientExitPath\
```

그리고 레지스트리 키는 `ExitsDefaultPath`입니다. 이 경로는 각 큐 관리자에 대해 설정할 수 있으며 엑시트는 해당 경로에서만 검색할 수 있습니다.

참고:

1. `CRTMQCVX`가 팩형 구조를 사용하는 경우, 모든 IBM MQ 애플리케이션을 이런 방식으로 컴파일해야 합니다.
2. 데이터 변환 엑시트 프로그램은 재입력 가능해야 합니다.
3. `MQXCNCVC`는 데이터 변환 엑시트에서 발행할 수 있는 MQI 호출입니다.

Windows Windows 운영 체제의 엑시트 및 스위치 로드 파일

IBM WebSphere MQ for Windows 7.5 큐 관리자는 32비트를 처리합니다. 결과적으로, 64비트 애플리케이션을 사용할 경우 일부 유형의 엑시트 및 XA 스위치 로드 파일에는 큐 관리자에서 사용할 수 있는 32비트 버전도 있어야 합니다. 엑시트 또는 XA 스위치 로드 파일의 32비트 버전이 필요하지만 사용 불가능한 경우에는 관련 API 호출 또는 명령이 실패합니다.

ExitPath에 대해 qm.ini file에서 두 개의 속성이 지원됩니다. 이는 ExitsDefaultPath= MQ_INSTALLATION_PATH\exits 및 ExitsDefaultPath64= MQ_INSTALLATION_PATH\exits64입니다. MQ_INSTALLATION_PATH는 IBM MQ가 설치되어 있는 상위 레벨 디렉토리를 나타냅니다. 해당 속성을 사용하여 적절한 라이브러리를 찾을 수 있습니다. IBM MQ 클러스터에서 엑시트를 사용하면 원격 시스템에서 적절한 라이브러리를 찾을 수 있습니다.

다음 표에서는 다양한 유형의 엑시트 및 스위치 로드 파일을 나열하고, 사용 중인 애플리케이션 버전이 32비트인지 64 비트인지에 따라 32비트 또는 64비트 버전 또는 둘 모두가 필요한지 여부를 나타냅니다.

파일 유형	32비트 애플리케이션	64비트 애플리케이션
API 엑시트(API exit)	32비트 및 64비트	64비트
데이터 변환 엑시트	32비트	64비트
서버 채널 엑시트(모든 유형)	64비트	64비트
클라이언트 채널 엑시트(모든 유형)	32비트	64비트
설치 가능 서비스 엑시트	64비트	64비트
클러스터 WLM 엑시트	64비트	64비트
발행/구독 라우팅 엑시트	64비트	64비트
데이터베이스 스위치 로드 파일	32비트 및 64비트	64비트
외부 트랜잭션 관리자 AX 라이브러리	32비트	64비트
사전 연결 엑시트	32비트	64비트

저장소의 사전 연결 엑시트를 사용하여 연결 정의 참조

IBM MQ MQI clients는 사전 연결 엑시트 라이브러리를 사용하여 연결 정의를 확보하기 위해 저장소를 검색하도록 구성할 수 있습니다.

소개

클라이언트 애플리케이션은 클라이언트 채널 정의 테이블(CCDT)을 사용하여 큐 관리자에 연결할 수 있습니다. 일반적으로 CCDT 파일은 중앙 네트워크 파일 서버에 위치하며 이를 참조하는 클라이언트가 포함되어 있습니다. CCDT 파일을 참조하는 다양한 클라이언트 애플리케이션을 관리하는 것이 어렵기 때문에 LDAP 디렉토리, WebSphere Registry and Repository 및 기타 저장소와 같은 글로벌 저장소에 클라이언트 정의를 저장하는 유연한 접근법을 사용합니다. 클라이언트 연결 정의를 저장소에 저장하면 클라이언트 연결 정의를 보다 쉽게 관리할 수 있으므로, 애플리케이션이 올바른 최신 클라이언트 연결 정의에 액세스할 수 있습니다.

MQCONN/X 호출을 실행하는 동안 IBM MQ MQI client는 애플리케이션이 지정한 사전 연결 엑시트 라이브러리를 로드하고 연결 정의를 검색하기 위한 엑시트 함수를 호출합니다. 그런 다음 검색된 연결 정의를 사용하여 큐 관리자에 대한 연결을 설정합니다. 호출할 엑시트 라이브러리 및 함수의 세부사항은 mqclient.ini 구성 파일에 지정되어 있습니다.

구문

```
void MQ_PRECONNECT_EXIT(pExitParms, pQMgrName, ppConnectOpts, pCompCode, pReason);
```

매개변수

pExitParms

유형: PMQNXIP 입력/출력

PreConnection 엑시트 매개변수 구조입니다.

구조는 엑시트의 호출자에 의해 할당되고 유지보수됩니다.

pQMgrName

유형: PMQCHAR 입출력(I/O)

큐 관리자의 이름.

입력 시, 이 매개변수는 **QMgrName** 매개변수를 통해 MQCONN API 호출에 제공되는 필터 문자열입니다. 이 필드는 공백이거나 명시적일 수 있으며 특정 와일드카드 문자를 포함할 수 있습니다. 필드는 엑시트에 의해 변경됩니다. 엑시트가 MQXR_TERM으로 호출된 경우 매개변수는 NULL입니다.

ppConnectOpts

유형: ppConnectOpts 입출력(I/O)

MQCONN의 조치를 제어하는 옵션입니다.

MQCONN API 호출의 조치를 제어하는 MQCNO 연결 옵션 구조에 대한 포인터입니다. 엑시트가 MQXR_TERM으로 호출된 경우 매개변수는 NULL입니다. MQI 클라이언트는 원래 애플리케이션에서 제공하지 않은 경우에도 항상 MQCNO 구조를 엑시트에 제공합니다. 애플리케이션이 MQCNO 구조를 제공하는 경우, 클라이언트는 수정된 엑시트에 전달하기 위해 복제본을 작성합니다. 이 클라이언트가 MQCNO의 소유권을 보유합니다.

MQCNO를 통해 참조되는 MQCD는 배열을 통해 제공된 모든 연결 정의에서 우선순위를 갖습니다. 클라이언트는 MQCNO 구조를 사용하여 큐 관리자에 연결하며 나머지는 무시됩니다.

pCompCode

유형: PMQLONG 입력/출력

완료 코드.

엑시트 완료 코드를 수신하는 MQLONG에 대한 포인터입니다. 다음 값 중 하나여야 합니다.

- MQCC_OK - 성공적인 완료
- MQCC_WARNING - 경고(부분 완료)
- MQCC_FAILED - 호출 실패

pReason

유형: PMQLONG 입력/출력

pCompCode를 규정하는 이유.

엑시트 이유 코드를 수신하는 MQLONG에 대한 포인터입니다. 완료 코드가 MQCC_OK인 경우 유일하게 올바른 값은 다음과 같습니다.

- MQRC_NONE - (0, x'000') 이유가 보고되지 않습니다.

완료 코드가 MQCC_FAILED 또는 MQCC_WARNING인 경우 엑시트 함수는 이유 코드 필드를 올바른 MQRC_* 값으로 설정할 수 있습니다.

C 호출

```
void MQ_PRECONNECT_EXIT (&ExitParms, &QMgrName, &pConnectOpts, &CompCode, &Reason);
```

Parameter

PMQNXIP	pExitParms	/*PreConnect exit parameter structure*/
PMQCHAR	pQMgrName	/*Name of the queue manager*/
PPMQCNO	ppConnectOpts	/*Options controlling the action of MQCONN*/
PMQLONG	pCompCode	/*Completion code*/
PMQLONG	pReason	/*Reason qualifying pCompCode*/

발행 엑시트 작성 및 컴파일

발행된 메시지를 구독자가 수신하기 전에 해당 메시지의 콘텐츠를 변경하도록 큐 관리자에서 발행 엑시트를 구성할 수 있습니다. 또한 메시지 헤더를 변경하거나 구독에 메시지를 전달하지 않을 수 있습니다.

참고: 발행 엑시트는 z/OS에서 지원되지 않습니다.

발행 엑시트를 사용하여 구독자에 전달된 메시지를 검사하고 대체할 수 있습니다.

- 각 구독자에게 발행된 메시지의 콘텐츠 조사
- 각 구독자에게 발행된 메시지의 콘텐츠 수정
- 메시지를 넣은 큐 대체
- 구독자에 대한 메시지 전달 중지

발행 엑시트 작성

엑시트를 작성하고 컴파일하는 데 도움을 받으려면 [688 페이지의 『AIX, Linux, and Windows에 엑시트 및 설치 가능 서비스 작성』](#)의 단계를 사용하십시오.

발행 엑시트의 제공자는 엑시트가 수행하는 작업을 정의합니다. 하지만 엑시트는 [MQPSXP](#)에 정의된 규칙을 준수해야 합니다.

IBM MQ는 MQ_PUBLISH_EXIT 시작점 구현을 제공하지 않습니다. C 언어 typedef 선언을 제공합니다. typedef를 사용하여 사용자 작성 엑시트에 대한 매개변수를 올바르게 선언하십시오. 다음 예제는 typedef 선언을 사용하는 방법을 보여줍니다.

```
#include "cmqec.h"

MQ_PUBLISH_EXIT MyPublishExit;

void MQENTRY MyPublishExit( PMQPSXP pExitParms,
                             PMQPBC  pPubContext,
                             PMQSBC  pSubContext )
{
    /* C language statements to perform the function of the exit */
}
```

발행 엑시트는 다음 조작의 결과로 큐 관리자 프로세스 내에서 실행됩니다.

- 하나 이상의 구독자에게 메시지를 전달하는 발행 조작
- 하나 이상의 보유된 메시지를 전달하는 구독 조작
- 하나 이상의 보유된 메시지를 전달하는 구독 요청 조작

연결에 대해 발행 엑시트를 호출하는 경우, 처음 호출할 때 MQXR_INIT의 *ExitReason* 코드가 설정됩니다. 발행 엑시트를 사용한 후 연결이 끊기기 전에 MQXR_TERM의 *ExitReason* 코드로 엑시트가 호출됩니다.

발행 엑시트가 구성되었지만, 큐 관리자를 시작할 때 이를 로드할 수 없으면 큐 관리자에서 발행/구독 메시지 조작이 금지됩니다. 발행/구독 메시징을 다시 사용 가능하게 하기 전에 문제점을 수정하거나 큐 관리자를 재시작해야 합니다.

발행 엑시트를 필요로 하는 각 IBM MQ 연결은 엑시트를 로드하거나 초기화하는 데 실패할 수 있습니다. 엑시트가 로드 또는 초기화에 실패한 경우, 해당 연결에 대한 발행 엑시트를 필요로 하는 발행/구독 조작이 사용 불가능합니다. 조작은 IBM MQ 이유 코드 MQRC_PUBLISH_EXIT_ERROR와 함께 실패합니다.

발행 엑시트가 호출되는 컨텍스트는 애플리케이션에서 큐 관리자로의 연결입니다. 발행 조작을 수행하는 각 연결에 대한 사용자 데이터 영역은 큐 관리자에서 유지보수됩니다. 엑시트는 각 연결의 사용자 데이터 영역에 정보를 보유할 수 있습니다.

발행 엑시트는 일부 MQI 호출을 사용할 수 있습니다. 메시지 특성을 조작하는 MQI 호출만 사용할 수 있습니다. 호출은 다음과 같습니다.

- MQBUFMH
- MQCRTMH
- MQDLTMH

- MQDLTMP
- MQMHBUF
- MQINQMP
- MQSETMP

발행 엑시트가 목적지 큐 관리자 또는 큐 이름을 변경하면 새 권한 검사가 수행되지 않습니다.

발행 엑시트 컴파일

발행 엑시트는 동적으로 로드된 라이브러리이며, 채널 엑시트로 간주할 수 있습니다. 엑시트 컴파일에 대한 정보는 688 페이지의 『AIX, Linux, and Windows에 엑시트 및 설치 가능 서비스 작성』의 내용을 참조하십시오.

샘플 발행 엑시트

샘플 엑시트 프로그램은 amqspse0.c입니다. 이는 초기화, 발행 또는 종료 조작에서 엑시트가 호출되었는지 여부에 따라 다른 메시지를 로그 파일에 씁니다. 또한 엑시트 사용자 영역 필드를 사용하여 스토리지를 적절히 할당하고 비우는 방법을 설명합니다.

발행 엑시트 구성

특정 속성을 정의하여 발행 엑시트를 구성해야 합니다.

Windows 및 Linux에서는 IBM MQ 탐색기를 사용하여 속성을 정의할 수 있습니다. 속성은 발행/구독 아래의 큐 관리자 특성 페이지에서 정의됩니다.

AIX and Linux 시스템의 qm.ini 파일에서 발행 엑시트를 구성하려면 PublishSubscribe라는 스탠자를 작성하십시오. PublishSubscribe 스탠자의 속성은 다음과 같습니다.

PublishExitPath=[path][module_name]

발행 엑시트 코드가 포함된 모듈 이름 및 경로. 이 필드의 최대 길이는 MQ_EXIT_NAME_LENGTH입니다. 기본값은 발행 엑시트 없음입니다.

PublishExitFunction= function_name

발행 엑시트 코드가 포함된 모듈에 대한 함수 시작점 이름입니다. 이 필드의 최대 길이는 MQ_EXIT_NAME_LENGTH입니다.

 IBM i에서 프로그램을 사용하는 경우 PublishExitFunction을 생략하십시오.

PublishExitData= string

큐 관리자가 발행 엑시트를 호출할 경우 MQPSXP 구조를 입력으로 전달합니다. **PublishExitData** 속성을 사용하여 지정된 데이터는 구조의 *ExitData* 필드에 제공됩니다. 문자열의 길이는 최대 MQ_EXIT_DATA_LENGTH자까지 가능합니다. 기본값은 32개의 공백 문자입니다.

클러스터 워크로드 엑시트 작성 및 컴파일

클러스터의 워크로드 관리를 사용자 정의하기 위해 클러스터 워크로드 엑시트 프로그램을 작성합니다. 메시지를 라우팅할 때, 하루에 여러 번 채널을 사용하는 비용이나 메시지 콘텐츠를 고려할 수 있습니다. 이들은 표준 워크로드 관리 알고리즘에서 고려하지 않는 요인입니다.

대부분의 경우 워크로드 관리 알고리즘으로 사용자의 요구를 충분히 충족시킬 수 있습니다. 그러나 IBM MQ는 사용자가 워크로드 관리를 조정하는 고유 사용자 엑시트 프로그램을 제공할 수 있도록 사용자 엑시트인 클러스터 워크로드 엑시트를 포함합니다.

워크로드 밸런싱에 영향을 주기 위해 사용할 수 있는 네트워크 또는 메시지에 관한 특정 정보를 가질 수 있습니다. 어느 것이 고용량 채널인지 또는 저렴한 네트워크 라우트인지 알 수도 있고, 콘텐츠에 따라 메시지를 라우팅할 수도 있습니다. 클러스터 워크로드 엑시트 프로그램을 작성할지 아니면 써드파티가 제공하는 것을 사용할 것인지 결정할 수 있습니다.

클러스터 워크로드 엑시트는 클러스터 큐에 액세스할 때 호출됩니다. MQOPEN, MQPUT1 및 MQPUT에 의해 호출됩니다.

MQOO_BIND_ON_OPEN이 지정되는 경우 MQOPEN 시간에 선택된 대상 큐 관리자가 고정됩니다. 이 경우에는 엑시트가 한 번만 실행됩니다.

대상 큐 관리자가 MQOPEN 시간에 고정되어 있지 않으면 대상 큐 관리자는 MQPUT 호출 시간에 선택됩니다. 대상 큐 관리자가 사용 불가능하거나 메시지가 여전히 전송 큐에 있는 동안에 실패하면 엑시트가 다시 호출됩니다. 새 대상 큐 관리자가 선택됩니다. 메시지 전송 중에 메시지 채널이 실패하고 메시지가 백아웃되면 새 대상 큐 관리자가 선택됩니다.

Multi 다중 플랫폼에서 큐 관리자는 큐 관리자가 다음 번 시작될 때 새 클러스터 워크로드 엑시트를 로드합니다.

큐 관리자 정의가 클러스터 워크로드 엑시트 프로그램 이름을 포함하지 않는 경우 클러스터 워크로드 엑시트가 호출되지 않습니다.

다양한 데이터가 엑시트 매개변수 구조 MQWXP의 클러스터 워크로드 엑시트에 전달됩니다.

- 메시지 정의 구조, MQMD
- 메시지 길이 매개변수
- 메시지의 사본 또는 메시지 일부

비z/OS 플랫폼에서 CLWLMode=FAST를 사용하는 경우, 각 운영 체제 프로세스는 엑시트의 고유 사본을 로드합니다. 큐 관리자에 대한 연결이 서로 다르면 엑시트의 여러 사본이 호출될 수 있습니다. 엑시트가 기본 안전 모드, CLWLMode=SAFE에서 실행되는 경우 엑시트의 단일 사본이 자체의 별도 프로세스에서 실행됩니다.

클러스터 워크로드 엑시트 작성

z/OS z/OS용 클러스터 워크로드 엑시트 작성에 대한 정보는 742 페이지의 『IBM MQ for z/OS에 대한 클러스터 워크로드 엑시트 프로그래밍』의 내용을 참조하십시오.

IBM MQ 9.1.0부터 클러스터 워크로드 엑시트는 큐 관리자 주소 공간 대신 채널 시작기 주소 공간에서 실행합니다. 클러스터 워크로드 엑시트가 있는 경우 큐 관리자 시작된 태스크 프로시저에서 CSQXLIB DD 문을 제거하고 클러스터 워크로드 엑시트가 포함된 데이터 세트를 채널 시작기 시작된 태스크 프로시저의 CSQXLIB 연결에 추가해야 합니다.

Multi 멀티플랫폼의 경우 클러스터 워크로드 엑시트에서는 MQI 호출을 사용하지 않아야 합니다. 다른 측면에서 클러스터 워크로드 엑시트 프로그램 작성 및 컴파일에 대한 규칙은 채널 엑시트 프로그램에 적용되는 규칙과 유사합니다. 688 페이지의 『AIX, Linux, and Windows에 엑시트 및 설치 가능 서비스 작성』의 단계를 수행하고 샘플 프로그램인 741 페이지의 『샘플 클러스터 워크로드 엑시트』를 사용하여 엑시트 작성 및 컴파일의 도움을 받으십시오.

채널 엑시트에 대한 자세한 정보는 712 페이지의 『채널 엑시트 프로그램 작성』의 내용을 참조하십시오.

클러스터 워크로드 엑시트 구성

ALTER QMGR 명령에 클러스터 워크로드 엑시트 속성을 지정하여 큐 관리자 정의에서 클러스터 워크로드 엑시트의 이름을 지정합니다. 예를 들면, 다음과 같습니다.

```
ALTER QMGR CLWLEXIT(myexit)
```

관련 참조

[클러스터 워크로드 엑시트 호출 및 데이터 구조](#)

샘플 클러스터 워크로드 엑시트

IBM MQ는 샘플 클러스터 워크로드 엑시트 프로그램을 포함합니다. 샘플을 복사하여 사용자 프로그램의 기초로 사용할 수 있습니다.

z/OS IBM MQ for z/OS

샘플 클러스터 워크로드 종료 프로그램은 어셈블러와, 형태로 제공됩니다 C. 어셈블러 버전은 라고 불리며 라이브러리 CSQ4BAF1 에서 찾을 thlqual.SCSQASMS 수 있습니다. C 버전은 CSQ4BCF1이고 라이브러리 thlqual.SCSQC37S에 있습니다. thlqual은 설치한 IBM MQ 데이터 세트에 대한 대상 라이브러리 상위 레벨 규정자입니다.

Multi IBM MQ for Multiplatforms

샘플 클러스터 워크로드 엑시트 프로그램은 C로 제공되며 이름이 amqswlm0.c입니다. 이 프로그램 위치는 다음과 같습니다.

표 120. 멀티플랫폼의 샘플 클러스터 워크로드 엑시트 프로그램 위치	
플랫폼	Filepath
 AIX	MQ_INSTALLATION_PATH/samp
 Windows	MQ_INSTALLATION_PATH\Tools\c\Samples
 IBM i	qmqm 라이브러리

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

이 샘플 엑시트는 큐 관리자가 사용 불가능하게 되는 경우 외에는 모든 메시지를 특정 큐 관리자로 라우팅합니다. 또한 메시지를 다른 큐 관리자로 라우팅하여 큐 관리자 실패에 대처합니다.

메시지를 송신할 큐 관리자를 나타냅니다. 큐 관리자 정의의 CLWLDATA 속성에서 클러스터-수신자 채널의 이름을 제공하십시오. 예를 들면, 다음과 같습니다.

```
ALTER QMGR CLWLDATA(' my-cluster-name. my-queue-manager ')
```

엑시트를 사용하려면 CLWLEXIT 속성에 전체 경로 및 이름을 제공하십시오.

  AIX and Linux에서:

```
ALTER QMGR CLWLEXIT(' path /amqswlm(cwlFunction)')
```

 Windows에서:

```
ALTER QMGR CLWLEXIT(' path \amqswlm(cwlFunction)')
```

 z/OS에서:

```
ALTER QMGR CLWLEXIT(CSQ4BxF1)
```

여기서 x는 'A' 또는 'C'이며, 사용 중인 버전의 프로그래밍 언어에 따라 다릅니다.

 IBM i에서는 다음 명령을 사용하십시오.

- MQSC 명령을 사용하십시오.

```
ALTER QMGR CLWLEXIT('AMQSWLM library ')
```

프로그램 이름 및 라이브러리 이름은 모두 10자이며 필요한 경우 오른쪽에 공백으로 채워집니다.

- CL 명령을 사용하십시오.

```
CHGMQM MQMNAME( qmgrname ) CLWLEXIT(' library /AMQSWLM')
```

이제 IBM MQ는 제공된 워크로드 관리 알고리즘을 사용하지 않고 이 엑시트를 호출하여 모든 메시지를 선택된 큐 관리자로 라우팅합니다.

IBM MQ for z/OS에 대한 클러스터 워크로드 엑시트 프로그래밍

클러스터 워크로드 엑시트는 z/OS LINK 명령에 의한 것처럼 호출됩니다. 엑시트는 엄격한 여러 프로그래밍 규칙을 따릅니다. 대기를 포함한 대부분의 SVC 명령을 사용하거나 워크로드 엑시트에서 STAE 또는 ESTAE를 사용하는 것을 삼가하십시오.

클러스터 워크로드 엑시트는 z/OS **LINK**에 의한 것처럼 호출됩니다:

- 비승인 문제점의 프로그램 상태
- 1차 주소 공간 제어 모드
- 비교차 메모리 모드
- 비액세스 등록 모드
- 31비트 주소 지정 모드
- 스토리지 키 8
- 프로그램 키 마스크 8
- TCB 키 8

채널 시작기의 시작된 태스크 프로시저의 CSQXLIB DD 문으로 지정된 데이터 세트에 링크 편집된 모듈을 넣으십시오. 로드 모듈 이름은 큐 관리자 정의에서 워크로드 엑시트 이름으로 지정됩니다.

IBM MQ for z/OS에 대한 채널 엑시트를 작성할 때 다음 규칙이 적용됩니다.

- 어셈블러 또는 C로 엑시트를 작성해야 합니다. C를 사용하는 경우 *z/OS C/C++ Programming Guide*, SC09-4765에 설명된 시스템 엑시트에 대한 C 시스템 프로그래밍 환경을 준수해야 합니다.
- MQXCLWLN 호출을 사용하는 경우, *thlqual*.SCSQLLOAD에 제공된 CSQMFLW로 링크 편집하십시오.
- 엑시트는 CSQXLIB DD문으로 정의된 비승인 라이브러리에서 로드됩니다. CSQXLIB에 DISP=SHR이 있는 경우, 큐 관리자가 실행 중일 때 큐 관리자가 시작하는 다음 MQCONN 스레드에서 사용되는 새 버전으로 엑시트를 업데이트할 수 있습니다.
- 엑시트는 재입력 가능해야 하고 가상 스토리지의 임의의 위치에서 실행할 수 있어야 합니다.
- 엑시트는 리턴 시 환경을 시작 시 환경으로 재설정해야 합니다.
- 엑시트는 확보된 스토리지를 비우거나, 후속 엑시트 호출로 스토리지가 비워졌는지 확인해야 합니다.
- MQI 호출은 허용되지 않습니다.
- 대기 시 큐 관리자의 성능이 크게 저하되므로, 엑시트는 대기를 야기하는 시스템 서비스를 사용하지 않아야 합니다. 따라서 되도록이면 SVC, PC 또는 I/O를 사용하지 마십시오.
- 엑시트는 첨부된 하위 태스크에서 발행하는 경우를 제외하고는 ESTAE 또는 SPIE를 발행하면 안 됩니다.

참고: 엑시트에서 수행할 수 있는 작업에 대한 절대적인 제한사항은 없습니다. 하지만 대부분의 SVC에는 대기가 포함되므로 다음 명령을 제외하고는 SVC를 사용하지 마십시오.

- **GETMAIN / FREEMAIN**
- **LOAD / DELETE**

ESTAE 및 ESPIE의 오류 핸들링은 IBM MQ에서 수행되는 오류 핸들링을 방해할 수 있으므로 사용하지 마십시오. IBM MQ가 오류를 복구하지 못할 수도 있고, 엑시트 프로그램이 모든 오류 정보를 수신하지 못할 수도 있습니다.

시스템 매개변수 EXITLIM은 엑시트가 실행되는 시간을 제한합니다. EXITLIM의 기본값은 30초입니다. 리턴 코드 MQRC_CLUSTER_EXIT_ERROR, 2266 X'8DA'가 표시되는 경우 엑시트가 루프 처리 중일 수 있습니다. 엑시트에서 완료하는 데 30초가 넘는 시간이 필요한 경우, EXITLIM 값을 늘리십시오.

큐잉을 위한 프로시저 애플리케이션 작성

이 정보를 사용하여 큐잉 애플리케이션 작성, 큐 관리자에 연결 및 연결 끊기, 발행/구독 및 오브젝트 열기 및 닫기에 대해 알아보십시오.

애플리케이션 작성에 대한 자세한 정보를 얻으려면 다음 링크를 사용하십시오.

- [744 페이지의 『MQI\(Message Queue Interface\) 개요』](#)
- [756 페이지의 『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』](#)
- [763 페이지의 『오브젝트 열기 및 닫기』](#)
- [772 페이지의 『큐에 메시지 넣기』](#)
- [786 페이지의 『큐에서 메시지 가져오기』](#)

- 822 페이지의 『발행/구독 애플리케이션 작성』
- 861 페이지의 『오브젝트 속성 조회 및 설정』
- 863 페이지의 『작업 단위 커밋 및 백아웃』
- 873 페이지의 『트리거를 사용한 IBM MQ 애플리케이션 시작』
- 891 페이지의 『MQI 및 클러스터에 대한 작업』
-  895 페이지의 『IBM MQ for z/OS의 애플리케이션 사용 및 작성』
-  71 페이지의 『IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션』

관련 개념

6 페이지의 『애플리케이션 개발 개념』

사용자는 원하는 절차적 또는 객체 지향 언어를 사용하여 IBM MQ 애플리케이션을 작성할 수 있습니다. IBM MQ 애플리케이션을 디자인하고 작성하기 전에 기본 IBM MQ 개념을 숙지하십시오.

5 페이지의 『IBM MQ용 애플리케이션 개발』

메시지를 송신하고 수신하며, 큐 관리자와 관련 자원을 관리하기 위한 애플리케이션을 개발할 수 있습니다. IBM MQ는 많은 다양한 언어와 프레임워크로 작성된 애플리케이션을 지원합니다.

53 페이지의 『IBM MQ 애플리케이션에 대한 설계 고려사항』

애플리케이션에서 사용 가능한 플랫폼과 환경을 이용할 수 있는 방법을 결정한 경우 IBM MQ에서 제공한 기능의 사용 방법을 결정해야 합니다.

917 페이지의 『클라이언트 프로시저 애플리케이션 작성』

프로시저 언어를 사용하여 IBM MQ에서 클라이언트 애플리케이션을 작성할 때 알아야 할 사항입니다.

937 페이지의 『프로시저 애플리케이션 빌드』

여러 프로시저 언어 중 하나로 IBM MQ 애플리케이션을 작성하고 여러 다른 플랫폼에서 애플리케이션을 실행할 수 있습니다.

1133 페이지의 『절차에 따른 프로그램 오류 핸들링』

이 정보는 호출할 때 또는 메시지를 최종 목적지에 전달할 때 애플리케이션 MQI 호출과 관련된 오류를 설명합니다.

관련 태스크

971 페이지의 『멀티플랫폼용 샘플 절차적 프로그램 사용』

이 샘플 프로시저 프로그램은 제품과 함께 전달됩니다. 샘플은 C 및 COBOL로 작성되며 MQI(Message Queue Interface)의 일반 사용을 보여줍니다.

1079 페이지의 『다음의 샘플 절차적 프로그램을 사용하여 z/OS』

IBM MQ for z/OS와 함께 전달된 샘플 프로시저 애플리케이션은 MQI(Message Queue Interface)의 일반적인 사용에 대해 설명합니다.

MQI(Message Queue Interface) 개요

MQI(Message Queue Interface) 컴포넌트에 대해 학습합니다.

MQI(Message Queue Interface)는 다음으로 구성됩니다.

- 프로그램이 큐 관리자 및 해당 기능에 액세스할 수 있도록 하는 호출
- 프로그램이 큐 관리자로(부터) 데이터를 전달하고 가져오는 데 사용하는 구조
- 큐 관리자로(부터) 데이터를 전달하고 가져오기 위한 요소 데이터 유형

 IBM MQ for z/OS는 또한 다음을 제공합니다.

- z/OS 배치 프로그램에서 변경사항을 커밋하고 백아웃할 수 있는 두 개의 추가 호출
- 데이터 정의 파일 (때로는 복사 파일, 매크로, 포함 파일, 헤더 파일로도 알려짐)은 IBM MQ for z/OS와 함께 제공되는 상수의 값을 정의합니다.
- 애플리케이션에 링크 편집할 스탭 프로그램.
- z/OS 플랫폼에서 MQI를 사용하는 방법을 보여주는 샘플 프로그램 스위트. 해당 샘플에 대한 추가 정보는 1079 페이지의 『다음의 샘플 절차적 프로그램을 사용하여 z/OS』의 내용을 참조하십시오.

IBM MQ for IBM i는 또한 다음을 제공합니다.

- 데이터 정의 파일 (때로는 복사 파일, 매크로, 포함 파일, 헤더 파일로도 알려짐)은 IBM MQ for IBM i과 함께 제공되는 상수의 값을 정의합니다.
- ILE C, ILE COBOL 및 ILE RPG 애플리케이션을 링크 편집하기 위한 세 가지 스텝 프로그램
- IBM i 플랫폼에서 MQI를 사용하는 방법을 보여주는 샘플 프로그램 스위트.

AIX, Linux, and Windows 시스템은 다음도 제공합니다.

- IBM MQ for AIX, Linux, and Windows 시스템 프로그램이 변경사항을 커밋하고 백아웃할 수 있는 호출
- 이러한 플랫폼에서 제공되는 상수 값을 정의하는 포함 파일
- 애플리케이션을 링크하는 라이브러리 파일
- 이러한 플랫폼에서 MQI를 사용하는 방법을 보여주는 샘플 프로그램의 스위트. 해당 샘플에 대한 추가 정보는 971 페이지의 [『멀티플랫폼용 샘플 절차적 프로그램 사용』](#)의 내용을 참조하십시오.
- 외부 트랜잭션 관리자에 대한 바인딩을 위한 샘플 소스 및 실행 가능 코드

MQI에 대한 자세한 정보를 얻으려면 다음 링크를 사용하십시오.

- 746 페이지의 [『MQI 호출』](#)
- 746 페이지의 [『동기점 호출』](#)
- 747 페이지의 [『데이터 변환, 데이터 유형, 데이터 정의 및 구조』](#)
- 748 페이지의 [『IBM MQ 스텝 프로그램 및 라이브러리 파일』](#)
- 753 페이지의 [『모든 호출에 공통적인 매개변수』](#)
- 753 페이지의 [『버퍼 지정』](#)
-  753 페이지의 [『z/OS 배치 고려사항』](#)
- 754 페이지의 [『AIX and Linux 신호 핸들링』](#)

관련 개념

756 페이지의 [『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』](#)

IBM MQ 프로그래밍 서비스를 사용하려면 프로그램을 큐 관리자에 연결해야 합니다. 이 정보를 사용하여 큐 관리자에 연결하고 큐 관리자에서 연결을 끊는 방법에 대해 알아보십시오.

763 페이지의 [『오브젝트 열기 및 닫기』](#)

이 정보는 IBM MQ 오브젝트의 열기 및 닫기 조작에 대한 통찰력을 제공합니다.

772 페이지의 [『큐에 메시지 넣기』](#)

이 정보를 사용하여 메시지를 큐에 넣는 방법에 대해 알아보십시오.

786 페이지의 [『큐에서 메시지 가져오기』](#)

이 정보를 사용하여 큐에서 메시지를 가져오는 방법에 대해 알아봅니다.

861 페이지의 [『오브젝트 속성 조회 및 설정』](#)

속성은 IBM MQ 오브젝트의 특성을 정의하는 특성입니다.

863 페이지의 [『작업 단위 커밋 및 백아웃』](#)

이 정보는 작업 단위에서 발생한 복구 가능한 가져오기 및 넣기 조작을 커밋하고 백아웃하는 방법을 설명합니다.

873 페이지의 [『트리거를 사용한 IBM MQ 애플리케이션 시작』](#)

트리거와 트리거를 사용하여 IBM MQ 애플리케이션을 시작하는 방법에 대해 학습하십시오.

891 페이지의 [『MQI 및 클러스터에 대한 작업』](#)

클러스터와 관련된 호출 및 리턴 코드에 대한 특수 옵션이 있습니다.

895 페이지의 [『IBM MQ for z/OS의 애플리케이션 사용 및 작성』](#)

IBM MQ for z/OS 애플리케이션은 여러 다른 환경에서 실행되는 프로그램으로 구성할 수 있습니다. 이는 둘 이상의 환경에서 사용 가능한 기능을 활용할 수 있음을 의미합니다.

71 페이지의 [『IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션』](#)

이 정보는 IBM MQ(를) 사용하여 IMS 애플리케이션을 작성하는 데 도움이 됩니다.

MQI 호출

이 정보를 사용하여 MQI(Message Queue Interface)의 호출에 대해 알아보십시오.

MQI에서의 호출은 다음과 같이 그룹화할 수 있습니다.

MQCONN, MQCONNX 및 MQDISC

프로그램을 큐 관리자에 연결하고(옵션 사용 또는 미사용), 큐 관리자에서 프로그램의 연결을 끊으려면 이러한 명령을 사용하십시오.

를 위한 프로그램을 CICSz/OS 작성하는 경우, 이러한 호출을 사용할 필요가 없습니다. 그러나 애플리케이션을 다른 플랫폼으로 포팅하려는 경우 이러한 호출 사용이 권장됩니다.

MQOPEN 및 MQCLOSE

큐와 같은 오브젝트를 열고 닫으려면 이러한 호출을 사용하십시오.

MQPUT 및 MQPUT1

메시지를 큐에 넣으려면 이러한 호출을 사용하십시오.

MQGET

큐에서 메시지를 찾아보거나 큐에서 메시지를 제거하려면 이 호출을 사용하십시오.

MQSUB, MQSUBRQ

토픽에 subscription을 등록하고 subscription에 일치하는 publication을 요청하려면 이러한 호출을 사용하십시오.

MQINQ

오브젝트의 속성에 대해 조회하려면 이 호출을 사용하십시오.

MQSET

큐의 속성 중 일부를 설정하려면 이 호출을 사용하십시오. 다른 유형의 오브젝트 속성은 설정할 수 없습니다.

MQBEGIN, MQCMIT 및 MQBACK

IBM MQ가 작업 단위의 코디네이터인 경우 이러한 호출을 사용하십시오. MQBEGIN은 작업 단위를 시작합니다. MQCMIT 및 MQBACK은 작업 단위 중에 작성된 업데이트를 커밋하거나 롤백하는 작업 단위를 종료합니다.  IBM i 커밋 제어기는 IBM MQ for IBM i에서 글로벌 작업 단위를 조정하는 데 사용됩니다. 고유 시작 커밋 제어, 커밋 및 롤백 명령이 사용됩니다.

MQCRTMH, MQBUFMH, MQMHBUF, MQDLTMH

메시지 핸들을 작성하고, 메시지 핸들을 버퍼 또는 메시지 핸들 버퍼로 변환하고 메시지 핸들을 삭제하려면 이러한 호출을 사용하십시오.

MQSETMP, MQINQMP, MQDLTMP

이러한 호출을 사용하여 메시지 핸들에서 메시지 특성을 설정하고, 메시지 특성에 대해 조회하고, 메시지 핸들에서 특성을 삭제하십시오.

MQCB, MQCB_FUNCTION, MQCTL

콜백 기능을 등록 또는 제어하려면 이러한 호출을 사용하십시오.

MQSTAT

이전의 비동기 Put 조작에 대한 상태 정보를 검색하려면 이 호출을 사용하십시오.

MQI 호출에 대한 설명은 [호출 설명](#)을 참조하십시오.

동기점 호출

이 정보를 사용하여 다른 플랫폼에서 동기점 호출에 대해 알아보십시오.

동기점 호출은 다음과 같이 사용 가능합니다.

IBM MQ for z/OS 호출



IBM MQ for z/OS는 MQCMIT 및 MQBACK 호출을 제공합니다.

z/OS 배치 프로그램의 이러한 호출을 사용하여 마지막 동기점 이후 모든 MQGET 및 MQPUT 조작이 영구적이 되거나(커밋됨) 백아웃된다고 큐 관리자에게 알립니다. 다른 환경에서 변경사항을 커밋하고 백아웃하려면,

CICS

EXEC CICS SYNCPOINT 및 EXEC CICS SYNCPOINT ROLLBACK과 같은 명령을 사용하십시오.

IMS

GU(get unique)와 같은 IMS 동기점 기능을 IOPCB, CHKP(체크포인트) 및 ROLB(롤백) 호출에 사용하십시오.

RRS

MQCMIT 및 MQBACK 또는 SRRCMIT 및 SRRBACK 중에 적절한 값을 사용하십시오. (867 페이지의 『트랜잭션 관리 및 복구 가능한 자원 관리자 서비스』의 내용을 참조하십시오.)

참고: SRRCMIT 및 SRRBACK은 고유 RRS 명령이며 MQI 호출이 아닙니다.

IBM i 호출

IBM i

IBM MQ for IBM i는 MQCMIT 및 MQBACK 명령을 제공합니다. 또한 IBM i COMMIT 및 ROLLBACK 명령, IBM i 커밋 제어 기능(예: EXEC CICS SYNCPOINT)을 시작하는 다른 명령 또는 호출을 사용할 수도 있습니다.

AIX, Linux, and Windows 플랫폼의 IBM MQ 호출

ALW

IBM MQ for AIX, Linux, and Windows에서는 MQCMIT 및 MQBACK 호출을 제공합니다.

프로그램의 동기점 호출을 사용하여 마지막 동기점 이후 모든 MQGET 및 MQPUT 조작이 영구적이 되거나(커밋됨) 백아웃된다고 큐 관리자에게 알립니다. CICS 환경에서 변경사항을 커밋하고 백아웃하려면 EXEC CICS SYNCPOINT 및 EXEC CICS SYNCPOINT ROLLBACK과 같은 명령을 사용하십시오.

데이터 변환, 데이터 유형, 데이터 정의 및 구조

이 정보를 사용하여 MQI(Message Queue Interface)를 사용할 때 데이터 변환, 기본 데이터 유형, IBM MQ 데이터 정의 및 구조에 대해 알아보십시오.

데이터 변환

MQXCNCVC(변환 문자) 호출은 하나의 문자 세트에서 다른 문자 세트로 메시지 문자 데이터를 변환합니다. 이 호출은 데이터 변환 종료 시에만 사용됩니다. 단, 다음의 경우는 예외입니다 IBM MQ for z/OS.

MQXCNCVC 호출과 함께 사용된 구문은 MQXCNCVC - 문자 변환의 내용을 참조하고 데이터 변환 엑시트 작성 및 호출에 대한 자세한 내용은 729 페이지의 『데이터 변환 엑시트 작성』의 내용을 참조하십시오.

기본 데이터 유형

지원되는 프로그래밍 언어의 경우 MQI는 기본 데이터 유형 또는 비구조화된 필드를 제공합니다.

이러한 데이터 유형은 기본 데이터 유형에서 자세히 설명됩니다.

IBM MQ 데이터 정의

z/OS IBM MQ for z/OS는 COBOL 복사 파일, 어셈블리 언어 매크로, 단일 PL/I 포함 파일, 단일 C 언어 포함 파일 및 C++ 언어 포함 파일의 양식으로 데이터 정의를 제공합니다.

IBM i IBM MQ for IBM i는 COBOL 복사 파일, RPG 복사 파일, C 언어 포함 파일 및 C++ 언어 포함 파일의 양식으로 데이터 정의를 제공합니다.

IBM MQ와 함께 제공되는 데이터 정의 파일에는 다음이 포함되어 있습니다.

- 모든 IBM MQ 상수 및 리턴 코드의 정의
- IBM MQ 구조 및 데이터 유형의 정의
- 구조 초기화를 위한 상수 정의
- 각 호출에 대한 함수 프로토타입(PL/I 및 C 언어 전용)

IBM MQ 데이터 정의 파일의 자세한 정의는 677 페이지의 『IBM MQ 데이터 정의 파일』의 내용을 참조하십시오.

구조

746 페이지의 『MQI 호출』에 나열된 MQI 호출과 함께 사용되는 구조는 지원되는 각 프로그래밍 언어에 대한 데이터 정의 파일에 제공됩니다.

구조체 유형에 대한 요약은 구조체 데이터 유형을 참조하십시오.

  IBM MQ for z/OS 그리고 이러한 구조체의 일부 필드를 완성할 때 IBM MQ for IBM i 사용할 수 있는 상수들이 포함된 파일을 제공합니다. 해당 파일에 대한 자세한 정보는 IBM MQ 데이터 정의를 참조하십시오.

IBM MQ 스텝 프로그램 및 라이브러리 파일

각 플랫폼별로 제공되는 스텝 프로그램 및 라이브러리 파일이 여기에 나열됩니다.

실행 가능한 애플리케이션을 빌드할 때 스텝 프로그램 및 라이브러리 파일을 사용하는 방법에 대한 자세한 정보는 937 페이지의 『프로시저 애플리케이션 빌드』의 내용을 참조하십시오. C++ 라이브러리 파일로 연결에 대한 자세한 정보는 C++ 사용 IBM MQ C++ 사용의 내용을 참조하십시오.

IBM MQ for AIX 라이브러리 파일

IBM MQ for AIX에서 프로그램을 애플리케이션을 실행 중인 환경에 제공되는 MQI 라이브러리 파일 및 운영 체제에서 제공하는 파일에 링크해야 합니다.

비스레드 애플리케이션에서 다음 라이브러리 중 하나에 링크하십시오.

표 121. 비스레드 AIX 애플리케이션에 대한 라이브러리 파일	
라이브러리 파일	환경
libmqm.a	C용 서버
libmqic.a 및 libmqm.a	C용 클라이언트
libmqmzf.a	C용 설치 가능 서비스 엑시트
libmqmxa.a	서버 XA 인터페이스
libmqmxa64.a	서버 대체 XA 인터페이스
libmqcxa.a	클라이언트 XA 인터페이스
libmqcxa64.a	클라이언트 대체 XA 인터페이스
libmqmcbt.o	Micro Focus COBOL용 IBM MQ 런타임 라이브러리 지원
libmqmcb.a	COBOL용 서버
libmqicb.a	COBOL용 클라이언트
libimqc23ia. ^{a1}	C++용 클라이언트 (x1c 16)
libimqs23ia. ^{a1}	C++용 서버 (x1c 16)
libimqc23ca. ^{a2}	C++용 클라이언트 (x1c 17)
libimqs23ca. ^{a2}	C++용 서버 (x1c 17)

표 참고사항:

- 이름에 "ia"가 포함된 라이브러리는 16 AIX 컴파일러용 IBM XL C/C++ 로 빌드되었습니다.  이 컴파일러에 대한 지원은 (IBM MQ 9.4.0 Long Term Support 릴리스의 경우, 그리고 IBM MQ 9.3.5 릴리스의 Continuous Delivery 경우) 더 이상 권장되지 않습니다.   이 컴파일러는 에서 제거되었습니다 IBM MQ 9.4.2.
- 이름에 "ca"가 포함된 라이브러리는 17 AIX 컴파일러용 IBM Open XL C/C++ 로 빌드되었습니다.

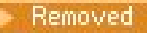
스레드 애플리케이션에서 다음 라이브러리 중 하나에 링크하십시오.

표 122. 스레드 AIX 애플리케이션에 대한 라이브러리 파일.

라이브러리 파일 및 각 라이브러리 파일의 환경을 나열하는 두 개의 열로 구성된 표입니다.

라이브러리 파일	환경
libmqm_r.a	C용 서버
libmqic_r.a 및 libmqm_r.a	C용 클라이언트
libmqmzf_r.a	C용 설치 가능 서비스 엑시트
libmqmxa_r.a	서버 XA 인터페이스
libmqmxa64_r.a	서버 대체 XA 인터페이스
libmqcxa_r.a	클라이언트 XA 인터페이스
libmqcxa64_r.a	클라이언트 대체 XA 인터페이스
libimqc23ia_r. ^{a1}	C++용 클라이언트 (xlc 16)
libimqs23ia_r. ^{a1}	C++용 서버 (xlc 16)
libimqc23ca_r. ^{a2}	C++용 클라이언트 (xlc 17)
libimqs23ca_r. ^{a2}	C++용 서버 (xlc 17)

표 참고사항:

- 이름에 "ia"가 포함된 라이브러리는 16 AIX 컴파일러용 IBMXL C/C++ 로 빌드되었습니다.  이 컴파일러에 대한 지원은 (IBM MQ 9.4.0 Long Term Support 릴리스의 경우, 그리고 IBM MQ 9.3.5 릴리스의 Continuous Delivery 경우) 더 이상 권장되지 않습니다.   이 컴파일러는 에서 제거되었습니다 IBM MQ 9.4.2.
- 이름에 "ca"가 포함된 라이브러리는 17 AIX 컴파일러용 IBMOpen XL C/C++ 로 빌드되었습니다.

참고: 둘 이상의 라이브러리에 링크할 수 없습니다. 즉, 동시에 스레드 및 비스레드 라이브러리 둘 모두에 링크할 수는 없습니다.

IBM MQ for IBM i 라이브러리 파일

IBM MQ for IBM i에서 운영 체제에서 제공하는 파일뿐만 아니라 애플리케이션을 실행 중인 환경에 제공된 MQI 라이브러리 파일에 프로그램을 링크하십시오.

비스레드 애플리케이션의 경우:

표 123. 비스레드 IBM i 애플리케이션에 대한 라이브러리 파일	
라이브러리 파일	환경
LIBMQM	서버 및 클라이언트 서비스 프로그램
LIBMQIC	클라이언트 서비스 프로그램
IMQB23I4	C++ 기반 서비스 프로그램
IMQS23I4	C++ 서버 서비스 프로그램
LIBMQMZF	C용 설치 가능 엑시트

스레드 애플리케이션의 경우:

표 124. 스레드 IBM i 애플리케이션에 대한 라이브러리 파일	
라이브러리 파일	환경
LIBMQM_R	서버 및 클라이언트 서비스 프로그램
IMQB23I4_R	C++ 기반 서비스 프로그램
IMQS23I4_R	C++ 서버 서비스 프로그램
LIBMQMZF_R	C용 설치 가능 엑시트
LIBMQIC_R	클라이언트 서비스 프로그램

IBM MQ for IBM i에서 C++로 애플리케이션을 작성할 수 있습니다. C++ 애플리케이션을 링크하는 방법 및 C++ 사용의 모든 측면에 대한 자세한 정보를 보려면 [C++ 사용](#)의 내용을 참조하십시오.

Linux 라이브러리 파일의 경우 IBM MQ

Linux의 IBM MQ 경우, 운영 체제에서 제공하는 라이브러리 파일 외에도, 애플리케이션을 실행하는 환경에 맞춰 제공되는 MQI 라이브러리 파일들을 프로그램에 링크해야 합니다.

비스레드 애플리케이션에서 다음 라이브러리 중 하나에 링크하십시오.

표 125. 비스레드 Linux 애플리케이션에 대한 라이브러리 파일	
라이브러리 파일	환경
libmqm.so	C용 서버
libmqic.so 및 libmqm.so	C용 클라이언트
libmqmzf.so	C용 설치 가능 서비스 엑시트
libmqmxa.so	서버 XA 인터페이스
libmqmxa64.so	서버 대체 XA 인터페이스
libmqcxa.so	클라이언트 XA 인터페이스
libmqcxa64.so	클라이언트 대체 XA 인터페이스
libimqc23gl.so	C++용 클라이언트
libimqs23gl.so	C++용 서버

스레드 애플리케이션에서 다음 라이브러리 중 하나에 링크하십시오.

표 126. 스레드 Linux 애플리케이션에 대한 라이브러리 파일	
라이브러리 파일	환경
libmqm_r.so	C용 서버
libmqic_r.so and libmqm_r.so	C용 클라이언트
libmqmzf_r.so	C용 설치 가능 서비스 엑시트
libmqmxa_r.so	서버 XA 인터페이스
libmqmxa64_r.so	서버 대체 XA 인터페이스
libmqcxa_r.so	클라이언트 XA 인터페이스
libmqcxa64_r.so	클라이언트 대체 XA 인터페이스
libimqc23gl_r.so	C++용 클라이언트
libimqs23gl_r.so	C++용 서버

참고: 둘 이상의 라이브러리에 링크할 수 없습니다. 즉, 동시에 스레드 및 비스레드 라이브러리 둘 모두에 링크할 수는 없습니다.

Windows IBM MQ for Windows 라이브러리 파일

IBM MQ for Windows에서는, 운영 체제에서 제공하는 파일 외에 애플리케이션을 실행 중인 환경에 대해 제공된 MQI 라이브러리 파일에 프로그램을 링크해야 합니다.

표 127. Windows 애플리케이션에 대한 라이브러리 파일	
라이브러리 파일	환경
<code>MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib</code>	C용 서버(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib</code>	C용 클라이언트(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\mqmxa.lib</code>	C용 서버 XA 인터페이스(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\mqcxa.lib</code>	C용 클라이언트 XA 인터페이스(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib</code>	C용 클라이언트 MTS(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\mqmcics4.lib32</code>	C용 서버 TXSeries CICS 지원(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\mqccics4.lib32</code>	C용 클라이언트 TXSeries CICS 지원(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\mqmzf.lib</code>	C용 설치 가능 서비스 엑시트(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb.lib</code>	IBM COBOL용 서버(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\mqmcb.lib</code>	Micro Focus COBOL용 서버(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\mqicbb.lib</code>	IBM COBOL용 클라이언트(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\mqiccb.lib</code>	Micro Focus COBOL용 클라이언트(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\imqs23vn.lib</code>	C++용 서버(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\imqc23vn.lib</code>	C++용 클라이언트(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\imqb23vn.lib</code>	C++용 기본(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib\imqx23vn.lib</code>	C++용 클라이언트 MTS(32비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib</code>	C용 서버(64비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib</code>	C용 클라이언트(64비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib64\mqmxa.lib</code>	C용 서버 XA 인터페이스(64비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib64\mqcxa.lib</code>	C용 클라이언트 XA 인터페이스(64비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib</code>	C용 클라이언트 MTS(64비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb.lib</code>	IBM COBOL용 서버(64비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb.lib</code>	Micro Focus COBOL용 서버(64비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib64\mqicbb.lib</code>	IBM COBOL용 클라이언트(64비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib64\mqiccb.lib</code>	Micro Focus COBOL용 클라이언트(64비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib64\imqs23vn.lib</code>	C++용 서버(64비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib64\imqc23vn.lib</code>	C++용 클라이언트(64비트)

표 127. Windows 애플리케이션에 대한 라이브러리 파일 (계속)	
라이브러리 파일	환경
<code>MQ_INSTALLATION_PATH\Tools\Lib64\imqb23vn.lib</code>	C++용 기본(64비트)
<code>MQ_INSTALLATION_PATH\Tools\Lib64\imqx23vn.lib</code>	C++용 클라이언트 MTS(64비트)

`MQ_INSTALLATION_PATH`은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

`amqmdnet.dll`을(를) .NET 프로그램 컴파일에 사용하십시오. 자세한 정보는 [520 페이지](#)의 『.NET 애플리케이션 개발』 섹션 내에 있는 [578 페이지](#)의 『IBM MQ .NET 프로그램 컴파일』의 내용을 참조하십시오.

이전 릴리스와의 호환성을 위해 다음 파일이 제공됩니다.

```
mqic32.lib
mqic32xa.lib
```

▶ **z/OS** IBM MQ for z/OS 스텝 프로그램

IBM MQ for z/OS로 작성된 프로그램을 실행할 수 있으려면 애플리케이션을 실행 중인 환경에 대해 IBM MQ for z/OS와 함께 제공된 스텝 프로그램에 대해 링크 편집해야 합니다.

스텝 프로그램은 IBM MQ for z/OS에서 처리할 수 있는 요청에 호출 처리의 첫 번째 단계를 제공합니다.

IBM MQ for z/OS는 다음 스텝 프로그램을 제공합니다.

CSQBSTUB

z/OS 배치 프로그램에 대한 스텝 프로그램

CSQBRRSI

MQI를 거쳐 RRS을 사용하는 z/OS 배치 프로그램에 대한 스텝 프로그램

CSQBRSTB

RRS을 직접 사용하는 z/OS 배치 프로그램에 대한 스텝 프로그램

CSQCSTUB

CICS 프로그램에 대한 스텝 프로그램

CSQQSTUB

IMS 프로그램에 대한 스텝 프로그램

CSQXSTUB

분산 큐잉 비CICS 엑시트에 대한 스텝 프로그램

CSQASTUB

데이터 변환 엑시트에 대한 스텝 프로그램



주의: 특정 환경에 대해 나열된 프로그램 외에 스텝 프로그램을 사용하는 경우 예상치 못한 결과가 발생할 수 있습니다.

참고: CSQBRSTB 스텝 프로그램을 사용하는 경우 SYS1.CSSLIB의 ATRSCSS로 링크 편집하십시오. (또한 SYS1.CSSLIB은 호출 가능한 서비스 라이브러리로 알려져 있습니다.) RRS에 대한 자세한 정보는 [867 페이지](#)의 『트랜잭션 관리 및 복구 가능한 자원 관리자 서비스』의 내용을 참조하십시오.

또는 프로그램 내에서 스텝을 동적으로 호출할 수 있습니다. 이 기술은 [963 페이지](#)의 『동적으로 IBM MQ 스텝 호출』에서 설명됩니다.

IMS에서 IBM MQ에서 제공하는 특수 언어 인터페이스 모듈을 사용해야 할 수도 있습니다.

동일한 IMS MPP 영역에서 CSQBSTUB 및 CSQQSTUB로 링크 편집된 애플리케이션을 실행하지 마십시오. 이로 인해 DFS3607I 또는 CSQQ005E 메시지와 같은 문제점이 발생할 수 있습니다. 주소 공간에서 첫 번째 MQCONN 호출은 사용되는 인터페이스를 판별하므로 CSQQSTUB 및 CSQBSTUB 트랜잭션을 다른 IMS 메시지 영역에서 실행해야 합니다.

모든 호출에 공통적인 매개변수

모든 호출에 공통적인 매개변수의 두 가지 유형(핸들 및 리턴 코드)이 있습니다.

핸들 사용

모든 MQI 호출은 하나 이상의 핸들을 사용합니다. 호출에 적절하게 큐 관리자, 큐 또는 기타 오브젝트, 메시지 또는 구독을 식별하십시오.

프로그램이 큐 관리자와 통신하기 위해 프로그램에는 해당 큐 관리자를 인지하는 고유 ID가 있어야 합니다. 이 ID는 연결 핸들이라고 하며 때때로 *Hconn*이라고 부릅니다. CICS 프로그램의 경우 연결 핸들은 항상 0입니다. 다른 모든 플랫폼 또는 프로그램 스타일에 대해 프로그램이 큐 관리자에 연결하는 경우 MQCONN 또는 MQCONNEX 호출에서 연결 핸들을 리턴합니다. 프로그램에서 다른 호출을 사용하는 경우 입력 매개변수로 연결 핸들을 전달합니다.

프로그램이 IBM MQ 오브젝트에 대해 작업하려면, 프로그램에 해당 오브젝트를 인지하는 고유 ID가 있어야 합니다. 이 ID는 오브젝트 핸들이라고 하며 때때로 *Hobj*라고 부릅니다. 프로그램에서 작업하기 위해 오브젝트를 여는 경우 MQOPEN 호출에서 핸들을 리턴합니다. 프로그램에서 후속 MQPUT, MQGET, MQINQ, MQSET 또는 MQCLOSE 호출을 사용하는 경우 입력 매개변수로 오브젝트 핸들을 전달합니다.

마찬가지로, MQSUB 호출은 후속 MQGET, MQCB 또는 MQSUBRQ 호출에서 구독을 식별하는 데 사용되는 구독 핸들 또는 *Hsub*을 리턴하고 메시지 특성을 처리하는 특정 호출은 메시지 핸들 또는 *Hmsg*를 사용합니다.

리턴 코드 이해

완료 코드 및 이유 코드가 각 호출의 출력 매개변수로 리턴됩니다. 이러한 코드는 전체적으로 리턴 코드라고 알려져 있습니다.

호출에 성공했는지 표시하기 위해 각 호출은 호출이 완료될 때 완료 코드를 리턴합니다. 완료 코드는 일반적으로 성공을 나타내는 MQCC_OK 또는 실패를 나타내는 MQCC_FAILED 중 하나입니다. 일부 호출은 중간 상태인 부분 성공을 나타내는 MQCC_WARNING을 리턴할 수 있습니다.

각 호출은 또한 호출의 부분 성공 또는 실패에 대해서는 이유를 표시하는 이유 코드를 리턴합니다. 큐가 가득 차거나, 큐에 허용되지 않은 조작을 가져오거나 큐 관리자에 정의되지 않은 특정 큐와 같은 상황을 처리하는 여러 가지 이유 코드가 있습니다. 프로그램은 진행 방법을 결정하는 데 이유 코드를 사용할 수 있습니다. 예를 들어, 사용자에게 입력 데이터를 변경하도록 프롬프트를 표시한 후 다시 호출을 작성하거나, 사용자에게 오류 메시지를 리턴할 수 있습니다.

완료 코드가 MQCC_OK인 경우 이유 코드는 항상 MQRC_NONE입니다.

각 호출에 대한 완료 코드 및 이유 코드가 해당 호출의 설명에 나열됩니다. [호출 설명](#)의 내용을 참조하고 목록에서 적절한 호출을 선택하십시오.

정정 조치에 대한 아이디어를 포함하여 자세한 정보는 다음을 참조하십시오.

- ▶ **z/OS** IBM MQ for z/OS 메시지, 완료 상태 및 사유 코드 for IBM MQ for z/OS
- 기타 모든 IBM MQ 플랫폼의 경우 [메시지 및 오류 코드](#)

버퍼 지정

큐 관리자는 필요한 경우에만 버퍼를 참조합니다. 호출 시 버퍼가 필요하지 않거나 버퍼 길이가 0인 경우 널 포인터를 버퍼에 사용할 수 있습니다.

필요한 버퍼의 크기를 지정하는 경우 항상 데이터 길이를 사용하십시오.

호출의 출력을 보관하기 위해 버퍼를 사용하는 경우(예를 들어, MQGET 호출에 대한 메시지 데이터 또는 MQINQ 호출에서 조회하는 속성 값을 보유하기 위해) 지정하는 버퍼가 올바르지 않거나 읽기 전용 스토리지인 경우 큐 관리자는 이유 코드를 리턴하려고 시도합니다. 그러나 항상 이유 코드를 리턴할 수 있는 것은 아닙니다.

▶ **z/OS** z/OS 배치 고려사항

z/OS MQI를 호출하는 배치 프로그램은 감독자 또는 문제 상태에 있을 수 있습니다.

그러나 다음 조건을 충족해야 합니다.

- 서비스 요청 차단(SRB) 모드가 아닌 태스크 모드여야 합니다.

- 액세스 등록 주소 공간 제어(ASC) 모드가 아닌 1차 ASC 모드여야 합니다.
- 교차 메모리 모드가 아니어야 합니다. 1차 주소 공간 번호(ASN)는 2차 ASN 및 홈 ASN과 같아야 합니다.
- MPF 엑시트 프로그램으로 사용되지 않아야 합니다.
- z/OS 잠금을 보유할 수 없습니다.
- 기능 복원 루틴(FRR) 스택에 FRR이 있을 수 없습니다.
- 모든 프로그램 상태 단어(PSW) 키는 MQCONN 또는 MQCONNX 호출에 대해 강제 실행될 수 있지만(키가 TCB 키에 있는 사용 스토리지와 호환된다는 가정에 따라), MQCONN 또는 MQCONNX에 의해 리턴된 연결 핸들을 사용하는 후속 호출에는 다음이 적용됩니다.
 - MQCONN 또는 MQCONNX 호출에 사용된 것과 동일한 PSW 키가 있어야 합니다.
 - 동일한 PSW 키 아래에 (적절한 곳에서 쓰기를 위해) 액세스 가능한 매개변수가 있어야 합니다.
 - 태스크의 모든 하위 태스크가 아닌 동일한 태스크(TCB) 아래에서 발행되어야 합니다.
- 24비트 또는 31비트 주소 지정 모드일 수 있습니다. 그러나 24비트 주소 지정 모드가 강제 실행되는 경우, 매개변수 주소는 올바른 31비트 주소로 해석되어야 합니다.

이러한 조건 중 하나라도 충족되지 않으면 프로그램 검사가 실행될 수 있습니다. 일부 경우에 호출이 실패하고 이 유 코드가 리턴됩니다.

Linux AIX AIX and Linux 고려사항

AIX and Linux 애플리케이션 개발 시 알아야 하는 고려사항입니다.

Linux AIX AIX and Linux 시스템의 분기 시스템 호출

IBM MQ 애플리케이션에서 분기 시스템 호출을 사용할 때 다음 고려사항을 참고하십시오.

애플리케이션이 `fork`를 사용하려는 경우, 해당 애플리케이션의 상위 프로세스는 IBM MQ 호출 (예: MQCONN)을 작성하거나 **ImqQueueManager**를 사용하여 IBM MQ 오브젝트를 작성하기 전에 `fork`를 호출해야 합니다.

애플리케이션이 IBM MQ 호출을 작성한 후에 하위 프로세스를 작성하려는 경우, 애플리케이션 코드는 `fork()`를 `exec()`와 함께 사용하여 하위가 상위의 사본이 아닌 새 인스턴스인지 확인해야 합니다.

애플리케이션이 `exec()`를 사용하지 않는 경우, 하위 프로세스 내에서 작성된 IBM MQ API 호출은 `MQRC_ENVIRONMENT_ERROR`를 리턴합니다.

Linux AIX AIX and Linux 신호 핸들링

일반적으로 AIX and Linux 시스템은 비스레드 (프로세스) 환경에서 멀티스레드 환경으로 이동되었습니다. 많은 경우에 신호 및 신호 핸들링은 지원되기는 하지만, 멀티스레드 환경에 그다지 적합하지 않으며 다양한 제한사항이 존재합니다.

일반적으로 AIX and Linux 시스템은 비스레드 (프로세스) 환경에서 멀티스레드 환경으로 이동되었습니다. 대부분의 애플리케이션은 신호 및 신호 핸들링을 인식할 필요가 없지만, 비스레드 환경에서 일부 기능은 신호를 사용해야지만 구현할 수 있습니다. 멀티스레드 환경에서, 스레드 기반 기본요소는 신호를 사용하는 비스레드 환경에서 구현하는 데 사용되는 일부 기능을 지원합니다.

많은 경우에 신호 및 신호 핸들링은 지원되기는 하지만, 멀티스레드 환경에 그다지 적합하지 않으며 다양한 제한사항이 존재합니다. 각 스레드가 신호를 핸들링하려고 하는 멀티스레드 환경에서, 애플리케이션 코드를 서로 다른 미들웨어 라이브러리(애플리케이션의 일부로 실행)와 통합 중인 경우 문제가 발생할 수 있습니다. 프로세스 내에 하나의 실행 스레드만 있을 때 효과가 있는, 신호 핸들러의 저장 및 복원을 위한 기존 접근법(프로세스별로 정의됨)은 멀티스레드 환경에서는 적합하지 않습니다. 그 이유는 많은 실행 스레드가 프로세스 전역의 자원을 저장 및 복원하려고 시도하여 예측할 수 없는 결과가 생길 수 있기 때문입니다.

Linux AIX 비스레드 애플리케이션

각 MQI 함수는 다음 신호에 대해 고유의 신호 핸들러를 설정합니다. 이러한 신호에 대한 사용자의 핸들러는 MQI 함수 호출 동안 대체됩니다. 다른 신호는 사용자가 작성한 핸들러에 의해 정상적인 방법으로 감지됩니다.

각 MQI 함수는 다음 신호에 대해 고유의 신호 핸들러를 설정합니다.

SIGALRM
SIGBUS

SIGFPE
SIGSEGV
SIGILL

이러한 신호에 대한 사용자의 핸들러는 MQI 함수 호출 동안 대체됩니다. 다른 신호는 사용자가 작성한 핸들러에 의해 정상적인 방법으로 감지됩니다. 핸들러를 설치하지 않는 경우 기본 조치(예: 무시, 코어 덤프 또는 종료)는 제자리에 남습니다.

IBM MQ는 동기 신호(SIGSEGV, SIGBUS, SIGFPE, SIGILL)를 핸들링한 후, MQI 함수 호출을 작성하기 전에 신호를 등록된 모든 신호 핸들러로 전달하려고 합니다.

Linux AIX 스레드 애플리케이션

스레드는 MQCONN(또는 MQCONNX)부터 MQDISC까지 IBM MQ에 연결되는 것으로 간주됩니다.

동기 신호

동기 신호는 특정 스레드에서 발생합니다.

AIX and Linux 시스템은 전체 프로세스 동안 이런 신호에 대해 신호 핸들러를 안전하게 설정하도록 지원합니다. 그러나 IBM MQ는 스레드가 IBM MQ에 연결되어 있는 동안 애플리케이션 프로세스에서 다음 신호에 대해 고유의 핸들러를 설정합니다.

SIGBUS
SIGFPE
SIGSEGV
SIGILL

멀티스레드 애플리케이션을 작성 중인 경우, 각 신호에 대해 오직 하나의 프로세스 전역 신호 핸들러가 있습니다. IBM MQ는 고유의 신호 핸들러를 설정할 때 각 신호에 대해 이전에 등록된 모든 핸들러를 저장합니다. IBM MQ에서 나열된 신호 중 하나를 처리한 후 IBM MQ는 프로세스 내에서 첫 번째 IBM MQ 연결 시 적용된 신호 핸들러를 호출하려고 합니다. 이전에 등록된 핸들러는 모든 애플리케이션 스레드가 IBM MQ에서 연결이 끊겼을 때 복원됩니다.

신호 핸들러는 IBM MQ에 의해 저장 및 복원되기 때문에, 동일한 프로세스의 다른 스레드가 IBM MQ에 역시 연결되어 있는 동안 애플리케이션 스레드는 이러한 신호에 대해 신호 핸들러를 설정하지 않아야 합니다.

참고: 스레드가 IBM MQ에 연결되어 있는 동안 애플리케이션 또는 미들웨어 라이브러리(애플리케이션의 일부로 실행)가 신호 핸들러를 설정하는 경우, 애플리케이션의 신호 핸들러는 해당 신호를 처리하는 동안 상응하는 IBM MQ 핸들러를 호출해야 합니다.

신호 핸들러의 설정 및 복원 시 일반 원칙은 저장될 마지막 신호 핸들러가 복원될 첫 번째 신호 핸들러여야 한다는 것입니다.

- IBM MQ에 연결한 후에 애플리케이션이 신호 핸들러를 설정하는 경우, 이전 신호 핸들러는 애플리케이션이 IBM MQ에서 연결을 끊기 전에 복원되어야 합니다.
- IBM MQ에 연결하기 전에 애플리케이션이 신호 핸들러를 설정하는 경우, 해당 신호 핸들러를 복원하기 전에 애플리케이션은 IBM MQ에서 연결을 끊어야 합니다.

참고: 저장될 마지막 신호 핸들러가 복원될 첫 번째 신호 핸들러여야 한다는 일반 원칙을 지키지 못하면, 애플리케이션에서 예상치 못한 신호 핸들링이 발생하고 잠재적으로 애플리케이션에 의해 신호가 유실될 수 있습니다.

비동기 신호

IBM MQ는 스레드 애플리케이션이 클라이언트 애플리케이션이 아닌 경우 스레드 애플리케이션에서 비동기 신호를 사용하지 않습니다.

스레드 클라이언트 애플리케이션에 대한 추가적인 고려사항

IBM MQ는 서버에 대한 I/O 동안 다음 신호를 핸들링합니다. 이러한 신호는 통신 스택에 의해 정의됩니다. 스레드가 큐 관리자에 연결되어 있는 동안 애플리케이션은 이러한 신호에 대해 신호 핸들러를 설정하지 않아야 합니다.



MQI에서 AIX and Linux 신호 핸들링 사용 시 추가적인 고려사항

AIX and Linux에서 신호 처리에 MQI를 사용 중인 경우, 최단경로 애플리케이션, 신호 핸들러 내의 MQI 기능 호출, MQI 호출 도중의 신호, 사용자 엑시트 및 설치 가능 서비스, VMS 엑시트 핸들러에 대해 추가로 고려해야 하는 사항이 있습니다.

빠른 경로(신뢰할 수 있는) 애플리케이션

빠른 경로 애플리케이션은 IBM MQ와 동일한 프로세스로 실행되므로 멀티스레드 환경에서 실행됩니다.

이 환경에서 IBM MQ는 동기 신호 SIGSEGV, SIGBUS, SIGFPE 및 SIGILL을 핸들링합니다. 다른 모든 신호는 IBM MQ에 연결되어 있는 동안 빠른 경로 애플리케이션으로 전달되지 않아야 합니다. 그 대신 애플리케이션에 의해 차단 또는 핸들링되어야 합니다. 빠른 경로 애플리케이션이 이런 이벤트를 가로채는 경우, 큐 관리자가 중지된 후 다시 시작되거나 정의되지 않은 상태로 남아 있을 수 있습니다. MQCONNX 아래의 빠른 경로 애플리케이션에 대한 제한사항의 전체 목록은 759 페이지의 『MQCONNX 호출을 사용하여 큐 관리자에 연결』의 내용을 참조하십시오.

신호 핸들러 내의 MQI 함수 호출

신호 핸들러에 있는 동안에는 MQI 함수를 호출하지 마십시오.

다른 MQI 함수가 활성 상태인 동안 신호 핸들러에서 MQI 함수를 호출하려고 하면 MQRC_CALL_IN_PROGRESS가 리턴됩니다. 활성 상태인 다른 MQI 함수가 없는 경우 신호 핸들러에서 MQI 함수를 호출하려고 하면, 선택적 호출만이 핸들러에서 또는 핸들러 내에서 발행될 수 있는 운영 체제 제한사항으로 인해 조작 동안 어느 시점에 일반적으로 실패합니다.

프로그램 엑시트 중에 자동으로 호출될 수 있는 C++ 소멸자 메소드의 경우, 호출 중인 MQI 함수를 중지하지 못할 수 있습니다. MQRC_CALL_IN_PROGRESS와 관련된 모든 오류는 무시하십시오. 신호 핸들러가 exit()를 호출하는 경우, 일반적으로 IBM MQ는 동기점의 커밋되지 않은 메시지를 백아웃하고 모든 열린 큐를 닫습니다.

MQI 호출 동안의 신호

MQI 함수는 코드 EINTR 또는 이와 상응한 코드를 애플리케이션 프로그램으로 리턴하지 않습니다.

신호가 MQI 호출 동안 발생하고 핸들러가 return을 호출하면, 마치 신호가 발생하지 않은 것처럼 호출이 계속 실행됩니다. 특히, MQGET은 애플리케이션으로 제어를 즉시 리턴하기 위해 신호에 의해 인터럽트될 수 없습니다. MQGET을 벗어나려는 경우 큐를 GET_DISABLED로 설정하십시오. 다른 방법으로, 유한 시간 만기(gmo.WaitInterval 세트가 포함된 MQGMO_WAIT)가 있는 MQGET 호출 주위에 루프를 사용하고, 신호 핸들러(비스레드 환경) 또는 스레드 환경의 상응하는 함수를 사용하여 루프를 벗어나는 플래그를 설정하십시오.



AIX 환경에서, IBM MQ는 신호에 의해 인터럽트된 시스템 호출을 다시 시작하도록 요청합니다. sigaction(2)를 사용하여 고유의 신호 핸들러를 설정할 때 SA_RESTART 플래그를 새 조치 구조의 sa_flags 필드에 설정하십시오. 그렇지 않으면 IBM MQ가 신호에 의해 인터럽트된 호출을 완료하지 못할 수 있습니다.

사용자 엑시트 및 설치 가능 서비스

멀티스레드 환경에서 IBM MQ 프로세스의 일부로 실행되는 사용자 엑시트 및 설치 가능 서비스는 빠른 경로 애플리케이션과 동일한 제한사항을 갖고 있습니다. 사용자 엑시트 및 설치 가능 서비스를 IBM MQ에 영구적으로 연결하고, 신호 또는 비스레드세이프 운영 체제 호출을 사용하지 않는 것을 고려하십시오.

큐 관리자에 연결 및 큐 관리자에서 연결 끊기

IBM MQ 프로그래밍 서비스를 사용하려면 프로그램을 큐 관리자에 연결해야 합니다. 이 정보를 사용하여 큐 관리자에 연결하고 큐 관리자에서 연결을 끊는 방법에 대해 알아보십시오.

연결이 작성되는 방법은 프로그램이 작동 중인 플랫폼 및 환경에 따라 다릅니다.

Multi IBM MQ for Multiplatforms

이러한 환경에서 실행되는 프로그램은 MQCONN MQI 호출을 사용하여 큐 관리자에 연결하고, MQDISC 호출을 사용하여 큐 관리자에서 연결을 끊을 수 있습니다. 또는 프로그램이 MQCONNX 호출을 사용할 수 있습니다.

z/OS IBM MQ for z/OS 배치

이 환경에서 실행되는 프로그램은 MQCONN MQI 호출을 사용하여 큐 관리자에 연결하고, MQDISC 호출을 사용하여 큐 관리자에서 연결을 끊을 수 있습니다. 또는 프로그램이 MQCONNX 호출을 사용할 수 있습니다.

z/OS 배치 프로그램은 동일한 TCB에 있는 다중 큐 관리자에 연속적으로 또는 동시에 연결할 수 있습니다.

z/OS IMS

IMS 제어 영역은 시작 시 하나 이상의 큐 관리자에 연결됩니다. 이 연결은 IMS 명령에 의해 제어됩니다.

z/OS에서 IMS 어댑터를 제어하는 방법에 대한 정보는 [관리 IBM MQ for z/OS](#)의 내용을 참조하십시오. 그러나 메시지 큐잉 IMS 프로그램의 작성자는 MQCONN MQI 호출을 사용하여 연결하려는 큐 관리자를 지정해야 합니다. MQDISC 호출을 사용하면 해당 큐 관리자에서 연결을 끊을 수 있습니다.

동기점을 설정하는 IMS 호출을 따르고 다른 사용자를 위해 메시지를 처리하기 전에 IMS 어댑터는 애플리케이션이 핸들을 닫고 큐 관리자에서 연결을 끊었는지 확인합니다. [866 페이지의 『IMS 애플리케이션의 동기점』](#)의 내용을 참조하십시오.

IMS 프로그램은 동일한 TCB에 있는 다중 큐 관리자에 연속적으로 또는 동시에 연결할 수 있습니다.

z/OS 용 CICS Transaction Server

CICS 프로그램은 CICS 시스템 자체가 연결되기 때문에 큐 관리자에 연결하기 위한 어떤 작업도 수행할 필요가 없습니다. 이 연결은 보통 초기화 시 자동으로 작성되지만 IBM MQ for z/OS와 함께 제공되는 CKQC 트랜잭션도 사용할 수 있습니다. CKQC에 대한 자세한 정보는 [관리 IBM MQ for z/OS](#)의 내용을 참조하십시오.

CICS 태스크는 CICS 리전이 연결되는 큐 관리자에만 연결할 수 있습니다.

CICS 프로그램은 MQI 연결 및 연결 끊기 호출(MQCONN 및 MQDISC)을 사용할 수도 있습니다. 이러한 애플리케이션을 최소의 레코딩으로 비CICS 환경으로 포팅할 수 있도록 이 작업을 수행할 수 있습니다. 그러나 이러한 호출은 항상 CICS 환경에서 성공적으로 완료됩니다. 이는 리턴 코드가 큐 관리자에 대한 연결의 실제 상태를 반영하지 않을 수 있음을 의미합니다.

Windows 및 개방 시스템용 TXSeries

이러한 프로그램은 CICS 시스템 자체가 연결되기 때문에 큐 관리자에 연결하기 위한 어떤 작업도 수행할 필요가 없습니다. 따라서 한 번에 하나의 연결만 지원됩니다. CICS 애플리케이션은 MQCONN 호출을 발행하여 연결 핸들을 확보하고 종료하기 전에 MQDISC 호출을 발행해야 합니다.

다음 링크를 사용하여 큐 관리자에 연결 및 큐 관리자에서 연결 끊기에 대해 자세히 알아보십시오.

- [758 페이지의 『MQCONN 호출을 사용하여 큐 관리자에 연결』](#)
- [759 페이지의 『MQCONNX 호출을 사용하여 큐 관리자에 연결』](#)
- [762 페이지의 『MQDISC를 사용하여 큐 관리자에서 프로그램 연결 끊기』](#)

관련 개념

[744 페이지의 『MQI\(Message Queue Interface\) 개요』](#)

MQI(Message Queue Interface) 컴포넌트에 대해 학습합니다.

[763 페이지의 『오브젝트 열기 및 닫기』](#)

이 정보는 IBM MQ 오브젝트의 열기 및 닫기 조작에 대한 통찰력을 제공합니다.

[772 페이지의 『큐에 메시지 넣기』](#)

이 정보를 사용하여 메시지를 큐에 넣는 방법에 대해 알아보십시오.

[786 페이지의 『큐에서 메시지 가져오기』](#)

이 정보를 사용하여 큐에서 메시지를 가져오는 방법에 대해 알아보십시오.

[861 페이지의 『오브젝트 속성 조회 및 설정』](#)

속성은 IBM MQ 오브젝트의 특성을 정의하는 특성입니다.

[863 페이지의 『작업 단위 커밋 및 백아웃』](#)

이 정보는 작업 단위에서 발생한 복구 가능한 가져오기 및 넣기 조작을 커밋하고 백아웃하는 방법을 설명합니다.

873 페이지의 『트리거를 사용한 IBM MQ 애플리케이션 시작』

트리거와 트리거를 사용하여 IBM MQ 애플리케이션을 시작하는 방법에 대해 학습하십시오.

891 페이지의 『MQI 및 클러스터에 대한 작업』

클러스터와 관련된 호출 및 리턴 코드에 대한 특수 옵션이 있습니다.

895 페이지의 『IBM MQ for z/OS의 애플리케이션 사용 및 작성』

IBM MQ for z/OS 애플리케이션은 여러 다른 환경에서 실행되는 프로그램으로 구성할 수 있습니다. 이는 둘 이상의 환경에서 사용 가능한 기능을 활용할 수 있음을 의미합니다.

71 페이지의 『IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션』

이 정보는 IBM MQ(를) 사용하여 IMS 애플리케이션을 작성하는 데 도움이 됩니다.

MQCONN 호출을 사용하여 큐 관리자에 연결

이 정보를 사용하여 MQCONN 호출을 통해 큐 관리자에 연결하는 방법을 알아보십시오.

일반적으로, 특정 큐 관리자 또는 기본 큐 관리자에 연결할 수 있습니다.

- ▶ **z/OS** IBM MQ for z/OS의 경우, 배치 환경에서 기본 큐 관리자는 CSQBDEFV 모듈에 지정됩니다.
- ▶ **Multi** IBM MQ for Multiplatforms의 경우 기본 큐 관리자는 mqs.ini 파일에서 지정됩니다.

▶ **z/OS** 또는 z/OS MVS 배치, TSO 및 RRS 환경에서 큐 공유 그룹 내에 있는 하나의 큐 관리자에 연결할 수 있습니다. MQCONN 또는 MQCONNX 요청은 그룹의 활성 멤버 중에 하나를 선택합니다.

큐 관리자에 연결할 때 큐 관리자는 태스크에 대해 로컬이어야 합니다. 즉, IBM MQ 애플리케이션과 동일한 시스템에 속해야 합니다.

▶ **z/OS** IMS 환경에서, 큐 관리자는 IMS 제어 영역에 연결되고 프로그램이 사용하는 종속 영역에 연결되어야 합니다. 기본 큐 관리자는 IBM MQ for z/OS가 설치될 때 CSQQDEFV 모듈에 지정됩니다.

CICS/ASX 환경에서 TXSeriesCICS, 그리고 TXSeries 및 에 대해 Windows, 큐 관리자는 에 대한 XA 리소스로 정의되어야 합니다.

기본 큐 관리자에 연결하려면, 전체적으로 공백으로 구성되거나 널(X'00') 문자로 시작하는 이름을 지정하여 MQCONN을 호출하십시오.

큐 관리자에 성공적으로 연결하려면 애플리케이션에 권한을 부여해야 합니다. 자세한 정보는 [보안 설정](#)을 참조하십시오.

MQCONN의 출력은 다음과 같습니다.

- 연결 핸들(Hconn)
- 완료 코드
- 이유 코드

후속 MQI 호출에서 연결 핸들을 사용하십시오.

이유 코드에 애플리케이션이 이미 해당 큐 관리자에 연결된 것으로 표시되는 경우, 리턴되는 연결 핸들은 애플리케이션이 처음 연결될 때 리턴된 것과 동일한 핸들입니다. 이 경우 호출 애플리케이션이 연결을 유지하는 것으로 예측되기 때문에 애플리케이션은 MQDISC 호출을 발행하지 않아야 합니다.

연결 핸들의 범위는 오브젝트 핸들의 범위와 동일합니다(764 페이지의 『MQOPEN 호출을 사용하여 오브젝트 열기』 참조).

매개변수에 대한 설명은 MQCONN에서 MQCONN 호출의 설명 부분에 있습니다.

MQCONN 호출을 발행할 때 큐 관리자가 정지 중 상태이거나 큐 관리자가 종료 중인 경우 호출이 실패합니다.

MQCONN 또는 MQCONNX의 범위

MQCONN 또는 MQCONNX 호출의 범위는 일반적으로 호출을 발행한 스레드입니다. 즉, 호출에서 리턴된 연결 핸들은 호출을 발행한 스레드 내에서만 유효합니다. 핸들을 사용하면 한 번에 하나의 호출만 작성할 수 있습니다. 다른 스레드에서 사용되는 경우 유효하지 않은 것으로 간주되어 거부됩니다. 애플리케이션에 다중 스레드가 있고 각각이 IBM MQ 호출을 사용하게 하려면, 각 호출이 MQCONN 또는 MQCONNX를 발행해야 합니다.

프로세스가 다중 MQCONN 호출을 작성하는 경우 각 호출을 동일한 큐 관리자에 대해 작성할 필요는 없습니다. 그러나 한 번에 하나의 IBM MQ 연결만 스레드에서 작성할 수 있습니다. 또는 760 페이지의 『MQCONN을 사용한 공유(스레드 독립) 연결』을 고려하여 단일 스레드의 다중 IBM MQ 연결과 모든 스레드의 단일 IBM MQ 연결을 사용하도록 허용하십시오.⁷

애플리케이션이 클라이언트로서 실행 중인 경우, 스레드 내에 있는 둘 이상의 큐 관리자에 연결할 수 있습니다.

MQCONN을 사용하여 큐 관리자에 연결

MQCONN 호출은 MQCONN 호출과 유사하지만, 호출의 작동 방법을 제어하기 위한 옵션이 포함되어 있습니다.

MQCONN에 대한 입력으로, 큐 관리자 이름  또는 z/OS 공유 큐 시스템의 큐 공유 그룹 이름을 제공할 수 있습니다. 큐 매니저에 대한 연결 방식을 제어하는 옵션은 MQCNO 라는 구조체에 제공됩니다.

MQCONN의 출력은 다음과 같습니다.

- 연결 핸들(Hconn)
- 완료 코드
- 이유 코드

후속 MQI 호출에서 연결 핸들을 사용합니다.

MQCNO 구조체의 연결 옵션 필드에 Options 설정된 연결 옵션은 연결의 여러 속성을 제어할 수 있게 합니다. 특히 주목할 만한 옵션 그룹은 다음과 같습니다:

- 바인딩 옵션을 통해 신뢰할 수 있는 애플리케이션을 생성할 수 있습니다. 신뢰할 수 있는 애플리케이션은 애플리케이션과 IBM MQ 로컬 큐 관리자 에이전트가 동일한 프로세스가 됨을 의미합니다. 에이전트 프로세스가 큐 관리자에 액세스하는 데 더 이상 인터페이스를 사용할 필요가 없으므로, 이러한 애플리케이션은 큐 관리자의 확장이 됩니다. 이 동작은 MQCNO_FASTPATH_BINDING 옵션을 지정하여 요청됩니다. 신뢰할 수 있는 애플리케이션에 적용되는 제한 사항에 대한 자세한 내용은 [759 페이지의 『신뢰할 수 있는 애플리케이션의 제한사항』](#).
- 핸들 공유 옵션을 통해 공유 연결을 생성할 수 있습니다. 공유 연결은 동일한 프로세스 내의 서로 다른 스레드 간에 핸들을 공유할 수 있습니다. 공유 연결에 대한 자세한 내용은 [760 페이지의 『MQCONN을 사용한 공유\(스레드 독립\) 연결』](#).

MQCNO는 또한 애플리케이션이 큐 관리자와의 연결 인증 방식을 제어할 수 있도록 합니다. 인증 자격 증명은 MQCNO 구조체에서 참조되는 MQCSP 구조체에 지정될 수 있습니다.

MQCONN 호출의 매개변수 및 제어 가능한 연결 속성에 대한 자세한 내용은 [MQCONN - 큐 관리자 연결\(확장\)](#)을 참조하십시오.

신뢰할 수 있는 애플리케이션의 제한사항

신뢰할 수 있는 애플리케이션에 적용되는 제한 사항. 일부 제한 사항은 모든 플랫폼에 적용되며, 다른 제한 사항은 플랫폼별로 다릅니다.

T

- 큐 관리자에서 신뢰할 수 있는 애플리케이션의 연결을 명확히 끊어야 합니다.
- 명령어로 **endmqm** 큐 관리자를 종료하기 전에 신뢰할 수 있는 애플리케이션을 중지해야 합니다.
- 비동기 신호 및 타이머 인터럽트(예: **sigkill**)를 MQCNO_FASTPATH_BINDING과 함께 사용하지 않아야 합니다.
- 모든 플랫폼에서, 신뢰할 수 있는 애플리케이션 내의 스레드는 동일한 프로세스에 있는 다른 스레드가 다른 큐 관리자에 연결되어 있는 동안 큐 관리자에 연결할 수 없습니다.
-   AIX and Linux 시스템의 경우, mqm을 모든 MQI 호출에 유효한 사용자 ID 및 그룹 ID로 사용해야 합니다. 인증이 필요한 비MQI 호출(예: 파일 열기)을 작성하기 전에 이들 ID를 변경할 수 있지만 그 다음 MQI 호출을 작성하기 전에 이를 다시 mqm으로 변경해야 합니다.

⁷ IBM MQ for AIX or Linux 시스템에 멀티스레드 애플리케이션을 사용하는 경우 애플리케이션의 스택 크기가 스레드에 충분하지 확인해야 합니다. 멀티스레드 애플리케이션이 자체적으로 또는 기타 신호 핸들러(예: CICS)를 사용하여 MQI 호출을 수행할 때에는 256KB 이상의 스택 크기를 사용할 것을 고려하십시오.

• **IBM i** IBM i의 경우:

1. 신뢰할 수 있는 애플리케이션은 QMQM 사용자 프로파일 아래에서 실행해야 합니다. 사용자 프로파일이 QMQM 그룹의 구성원이 되거나 프로그램에서 QMQM 권한을 채택하는 것만으로는 부족합니다. QMQM 사용자 프로파일을 대화식 작업에 사인온하는 데 사용하거나 신뢰할 수 있는 애플리케이션을 실행하는 작업의 작업 설명에 지정하는 것은 불가능할 수 있습니다. 이 경우 한 가지 접근 방식은 프로파일 전환 API IBM i 함수인 QSYGETPH, QWTSETP 및 QSYRLSPH를 사용하여 프로그램이 IBM MQ 실행되는 동안 작업의 현재 사용자를 일시적으로 QMQM으로 변경하는 것입니다. 이러한 함수의 세부 사항과 사용 예시는 애플리케이션 프로그래밍 IBM i 인터페이스 문서의 [보안 API](#) 섹션에서 확인할 수 있습니다.
2. ENDJOB을 사용하여 실행 중인 작업을 종료하거나 시스템 요청 옵션 2를 사용하여 신뢰할 수 있는 애플리케이션을 취소하지 마십시오.

• **ALW** AIX, Linux, and Windows에서 신뢰할 수 있는 32비트 애플리케이션은 지원되지 않습니다. 신뢰할 수 있는 32비트 애플리케이션을 실행하려는 경우, 표준 바인드 연결로 다운그레이드됩니다.

Multi MQCONN를 사용한 공유(스레드 독립) 연결

이 정보를 사용하여 MQCONN를 사용한 공유 연결과 고려할 몇 가지 사용 시 참고사항에 대해 알아보십시오.

참고: **z/OS** IBM MQ for z/OS에서 지원되지 않습니다.

멀티플랫폼에서는 로 생성된 연결은 해당 연결을 MQCONN 생성한 스레드에서만 사용할 수 있습니다. MQCONN 호출의 옵션을 사용하면 프로세스의 모든 스레드에서 공유할 수 있는 연결을 작성할 수 있습니다. 애플리케이션이 MQI 호출을 동일한 스레드에서 발행해야 하는 트랜잭션 환경에서 실행 중인 경우, 다음과 같은 기본 옵션을 사용해야 합니다.

MQCNO_HANDLE_SHARE_NONE

비공유 연결을 작성합니다.

대부분의 다른 환경에서는 다음과 같은 스레드 독립의 공유 연결 옵션 중 하나를 사용할 수 있습니다.

MQCNO_HANDLE_SHARE_BLOCK

공유 연결을 작성합니다. MQCNO_HANDLE_SHARE_BLOCK 연결에서, 해당 연결이 현재 다른 스레드의 MQI 호출에서 사용되고 있는 경우 MQI 호출은 현재 MQI 호출이 완료될 때까지 대기합니다.

MQCNO_HANDLE_SHARE_NO_BLOCK

공유 연결을 작성합니다. MQCNO_HANDLE_SHARE_NO_BLOCK 연결에서, 해당 연결이 현재 다른 스레드의 MQI 호출에서 사용되고 있는 경우 MQI 호출은 MQRC_CALL_IN_PROGRESS의 이유로 즉시 실패합니다.

MTS(Microsoft Transaction Server) 환경을 제외하고, 기본값은 MQCNO_HANDLE_SHARE_NONE입니다. MTS 환경에서 기본값은 MQCNO_HANDLE_SHARE_BLOCK입니다.

연결 핸들이 MQCONN 호출에서 리턴됩니다. 프로세스에 있는 모든 스레드의 후속 MQI 호출을 MQCONN에서 리턴된 핸들과 연관시키고 해당 호출에서 핸들을 사용할 수 있습니다. 단일 공유 핸들을 사용하는 MQI 호출은 스레드 전체에서 직렬화됩니다.

예를 들어, 공유 핸들을 사용하여 다음 활동 순서가 가능합니다.

1. 스레드 1이 MQCONN를 발행하고 공유 핸들 h1을 가져옵니다.
2. 스레드 1이 큐를 열고 h1을 사용하여 가져오기 요청을 발행합니다.
3. 스레드 2가 h1을 사용하여 넣기 요청을 발행합니다.
4. 스레드 3이 h1을 사용하여 넣기 요청을 발행합니다.
5. 스레드 2가 h1을 사용하여 MQDISC를 발행합니다.

핸들이 스레드에서 사용 중인 동안에는 연결 액세스를 다른 스레드에 사용할 수 없습니다. 스레드가 다른 스레드의 모든 이전 호출이 완료될 때까지 대기하도록 허용되는 환경에서는 MQCONN를 옵션 MQCNO_HANDLE_SHARE_BLOCK과 함께 사용하십시오.

그러나 차단은 문제를 일으킬 수 있습니다. 760 페이지의 『2』 단계에서 스레드 1이 아직 도착하지 않았을 수 있는 메시지를 대기하는 가져오기(대기를 포함한 가져오기) 요청을 발행한다고 가정하십시오. 이 경우, 스레드 2와 3도 스레드 1에서의 가져오기 요청이 걸리는 시간만큼 대기 상태를 유지합니다(차단됩니다). 다른 MQI 호출

이 핸들에서 이미 실행 중인 경우 MQI 호출이 오류와 함께 리턴되는 것을 선호하는 경우 MQCONN를 옵션 MQCNO_HANDLE_SHARE_NO_BLOCK과 함께 사용하십시오.

공유 연결 사용 시 참고사항

1. 오브젝트를 열어서 작성된 모든 오브젝트 핸들(Hobj)은 Hconn과 연관됩니다. 따라서 공유 Hconn의 경우, Hconn을 사용하는 모든 스레드에서 Hobj를 공유 및 사용할 수 있습니다. 마찬가지로, Hconn 아래에서 시작된 작업 단위도 해당 Hconn과 연관되므로 작업 단위 역시 공유 Hconn을 사용하여 스레드 전체에서 공유됩니다.
2. 모든 스레드는 MQDISC를 호출하여 해당 MQCONN을 호출한 스레드뿐만 아니라 공유 Hconn의 연결을 끊을 수 있습니다. MQDISC는 Hconn을 종료하여 모든 스레드에서 사용 불가능하게 만듭니다.
3. 단일 스레드는 다중 공유 Hconn을 연속으로 사용할 수 있습니다. 예를 들어, 각 조작을 다른 로컬 작업 단위 아래에서 수행되게 하고, MQPUT을 사용하여 하나의 공유 Hconn 아래에 하나의 메시지를 넣은 후 다른 공유 Hconn을 사용하여 다른 메시지를 넣으십시오.
4. 공유 Hconn은 글로벌 작업 단위 내에서 사용할 수 없습니다.

Multi MQ_CONNECT_TYPE과 함께 MQCONN 호출 옵션 사용
이 정보를 활용하여 다양한 MQCONN 호출 옵션과 환경 MQ_CONNECT_TYPE 변수와 함께 사용하는 방법을 이해하십시오.

참고: MQ_CONNECT_TYPE 표준 바인딩에만 효과가 있습니다. 다른 바인딩의 경우, MQ_CONNECT_TYPE 는 무시됩니다.

에서 IBM MQ for Multiplatforms 환경 변수 를 MQCONN 호출 시 사용되는 MQCNO 구조체의 필드에 Options 지정된 바인딩 유형과 함께 사용할 수 MQ_CONNECT_TYPE 있습니다.

표 128. MQCONN 콜 옵션이 환경 MQ_CONNECT_TYPE 변수와 함께 사용되는 방법		
MQCONN 호출 옵션	MQ_CONNECT_TYPE 환경 변수	결과
STANDARD	UNDEFINED	STANDARD
STANDARD	STANDARD	STANDARD
STANDARD	FASTPATH	STANDARD
STANDARD	CLIENT	CLIENT
STANDARD	로컬	STANDARD

MQCNO_STANDARD_BINDING이 지정되지 않은 경우 MQCNO_STANDARD_BINDING의 기본값인 MQCNO_NONE을 사용할 수 있습니다.

MQCONN 및 MQCONN을 위한 인증 및 신원 확인

이 작업을 통해 애플리케이션이 연결 시 인증에 사용되는 자격 증명을 IBM MQ 제공하는 방법을 학습하십시오.

기본 사용자 신원

애플리케이션이 메시지 큐 인터페이스(MQI)를 사용하여 MQCONN 또는 MQCONN으로 연결할 IBM MQ 경우, 항상 사용자 신원이 설정되어 해당 연결과 연결됩니다.

기본적으로 초기 사용자 ID는 항상 애플리케이션이 실행되는 운영 체제 프로세스의 ID입니다. 이 초기 신원 정보는 로컬에 바인딩되거나 신뢰된 애플리케이션 연결에는 충분할 수 있습니다.

애플리케이션이 MQCONN 호출로 큐 관리자에 연결할 때, 애플리케이션은 기본 사용자 ID를 수정할 수 없습니다. 그러나 다음 메커니즘은 연결과 연관된 사용자 ID를 변경할 수 있습니다:

- 클라이언트 측 또는 서버 측 보안 종료.
- 큐 매니저의 채널 인증 규칙.
- TLS 상호 인증 과정에서 설정된 클라이언트 사용자 ID.

MQCONN를 사용하여 자격 증명 제공

MQCONN는 애플리케이션이 연결과 연관된 신원 정보에 대해 더 많은 제어권을 부여합니다. 응용 프로그램은 MQCONN **ConnectOpts** 매개변수에 지정된 연결 옵션의 일부로 MQCSP 구조를 제공할 수 있습니다. MQCSP 구조는 사용자 신원을 확인하는 데 사용되는 자격 증명을 포함할 수 있습니다. IBM MQ MQCSP 구조에서 다음 자격 증명을 지원합니다:

- 사용자 ID와 비밀번호.
- 에서 IBM MQ 9.3.4, 애플리케이션이 또는 Linux AIX 시스템에서 실행되는 큐 관리자에 연결하는 경우 인증 토큰.

큐 관리자의 연결 인증 및 채널 인증 구성은 애플리케이션이 제공하는 자격 증명 처리되는 방식을 제어합니다. 예를 들어, 이 구성은 다음 측면에 영향을 미칩니다:

- MQCSP 구조 내 자격 증명 검증되는지 여부 및 검증 방식.
- MQCSP 구조 내 자격 증명서의 사용자 ID가 다른 사용자 ID에 매핑되는지 여부.
- 인증된 사용자가 이후 애플리케이션의 컨텍스트로 채택되는지 여부.

연결 인증에 대한 자세한 내용은 [연결 인증](#)을 참조하십시오. 채널 인증에 대한 자세한 정보는 [채널 인증 레코드](#)를 참조하십시오.

MQI 를 사용하는 C 언어로 작성된 몇 가지 예제 프로그램은 MQCSP 구조체를 활용하여 인증 자격 증명을 제공하는 방법을 보여줍니다. 자세한 내용은 다음 샘플 프로그램을 참조하십시오:

- [1001 페이지의 『Get 샘플 프로그램』](#)
- [1013 페이지의 『Put 샘플 프로그램』](#)
- [990 페이지의 『브라우저 샘플 프로그램』](#)
- [1028 페이지의 『TLS 샘플 프로그램』](#)

관련 정보

[MQCSP 구조를 사용하여 사용자 식별 및 인증](#)

[MQCSP - 보안 매개변수](#)

[사용자 식별 및 인증](#)

MQDISC를 사용하여 큐 관리자에서 프로그램 연결 끊기

이 정보를 사용하여 MQDISC를 통해 큐 관리자에서 프로그램의 연결을 끊는 방법에 대해 알아보십시오.

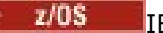
MQCONN 또는 MQCONNX 호출을 사용하여 연결된 프로그램이 큐 관리자와 모든 상호작용을 완료하면 다음을 제외하고 MQDISC 호출을 사용하여 연결을 끊습니다.

-  **z/OS** 트랜잭션 서버 CICS 애플리케이션에서는 z/OS MQCONN이 사용되지 않은 경우 호출이 선택 사항이며, 애플리케이션 종료 전에 연결 태그를 해제하려는 경우에 해당합니다.
-  **IBM i** 운영 체제에서 사인오프할 때 암시적 MQDISC 호출이 작성되는 IBM MQ for IBM i에서.

MQDISC 호출에 대한 입력으로, 큐 관리자에 연결되었을 때 MQCONN 또는 MQCONNX에서 리턴된 연결 핸들 (Hconn)을 제공해야 합니다.

다중 플랫폼에서 실행할 CICS 때, MQDISC가 호출된 후에는 연결 핸들(Hconn)이 더 이상 유효하지 않으며, MQCONN 또는 MQCONNX를 다시 호출할 때까지 더 이상 MQI 호출을 수행할 수 없습니다. MQDISC는 이 핸들을 사용하여 아직 열려 있는 모든 오브젝트에 대해 암시적 MQCLOSE를 수행합니다.

 **z/OS** z/OS에 연결된 클라이언트의 경우, MQDISC 호출이 발행될 때 암시적 커미트가 일어나지만 아직 열려 있는 모든 큐 핸들은 채널이 실제로 종료될 때까지 닫히지 않습니다.

 **z/OS** IBM MQ for z/OS에서 MQCONNX를 사용하여 연결하는 경우, MQDISC는 MQCONNX에 의해 설정된 연결 태그의 범위도 종료합니다. 그러나 CICS, IMS 또는 RRS 애플리케이션에서 연결 태그와 연관된 활성 복구 단위가 있는 경우, MQDISC는 이유 코드 MQRC_CONN_TAG_NOT_RELEASED로 거부됩니다.

매개변수에 대한 설명은 [MQDISC](#)에서 MQDISC 호출의 설명 부분에 있습니다.

MQDISC가 발행되지 않는 시기

표준 비공유 연결(Hconn)은 스레드 작업이 종료될 때 정리됩니다. 공유 연결은 전체 프로세스가 종료될 때에만 암시적으로 백아웃되고 연결이 끊어집니다. 공유 Hconn을 작성한 스레드가 종료되지만 Hconn이 여전히 존재하는 경우 Hconn은 계속 사용 불가능합니다.

권한 검사

MQCLOSE 및 MQDISC 호출은 일반적으로 권한 검사를 수행하지 않습니다.

일반적인 이벤트 과정에서, IBM MQ 오브젝트를 열거나 연결할 권한이 있는 작업은 해당 오브젝트를 닫거나 오브젝트에서 연결을 끊습니다. IBM MQ 오브젝트를 열거나 오브젝트에 연결된 작업의 권한이 취소되는 경우에도 MQCLOSE 및 MQDISC 호출은 승인됩니다.

오브젝트 열기 및 닫기

이 정보는 IBM MQ 오브젝트의 열기 및 닫기 조작에 대한 통찰력을 제공합니다.

다음 조작을 수행하려면 먼저 관련 IBM MQ 오브젝트를 열어야 합니다.

- 큐에 메시지 넣기
- 큐에서 메시지 가져오기(찾아보기 또는 검색)
- 오브젝트의 속성 설정
- 모든 오브젝트의 속성 조회

오브젝트를 열려면 오브젝트에 대해 수행하려는 작업을 지정하는 호출 옵션과 함께 MQOPEN 호출을 사용하십시오. 유일한 예외는 단일 메시지를 큐에 넣은 후 큐를 즉시 닫으려는 경우입니다. 이 경우 MQPUT1 호출을 사용하여 열기 단계를 무시할 수 있습니다([780 페이지의 『MQPUT1 호출을 사용하여 큐에 하나의 메시지 넣기』](#) 참조).

MQOPEN 호출을 사용하여 오브젝트를 열기 전에 프로그램을 큐 관리자에 연결해야 합니다. 모든 환경에서 이와 관련된 자세한 내용은 [756 페이지의 『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』](#)에서 설명됩니다.

다음과 같이 사용자가 열 수 있는 IBM MQ 오브젝트의 유형은 네 가지입니다.

- 큐
- 이름 목록
- 프로세스 정의
- 큐 관리자

MQOPEN 호출을 사용하는 방법과 유사하게 이러한 모든 오브젝트를 열 수 있습니다. IBM MQ 오브젝트에 대한 자세한 정보는 [오브젝트 유형](#)을 참조하십시오.

동일한 오브젝트를 두 번 이상 열 수 있으며, 그 때마다 새 오브젝트 핸들을 가져옵니다. 한 핸들을 사용하여 큐에서 메시지를 찾아보고, 다른 핸들을 사용하여 동일한 큐에서 메시지를 제거할 수 있습니다. 이렇게 하면 동일한 오브젝트를 닫고 다시 열기 위한 자원이 절약됩니다. 또한 메시지에 대해 찾아보기 및 제거 조작을 동시에 수행하기 위해 큐를 열 수 있습니다.

이 밖에도, 단일 MQOPEN을 사용하여 다중 오브젝트를 열고 MQCLOSE를 사용하여 닫을 수 있습니다. 이 방법에 대한 정보는 [781 페이지의 『분배 목록』](#)의 내용을 참조하십시오.

오브젝트를 열려고 하는 경우 큐 관리자는 MQOPEN 호출에 지정한 옵션으로 해당 오브젝트를 열 권한이 사용자에게 있는지 검사합니다.

오브젝트는 프로그램이 큐 관리자에서 연결이 끊어질 때 자동으로 닫힙니다. IMS 환경에서, 연결 끊기는 GU(get unique) IMS 호출 뒤에 프로그램이 새 사용자를 위한 처리를 시작할 때 강제 실행됩니다. IBM i 플랫폼에서, 오브젝트는 작업이 종료될 때 자동으로 닫힙니다.

이미 연 오브젝트를 닫는 것은 좋은 프로그래밍 사례입니다. MQCLOSE 호출을 사용하여 이 작업을 수행하십시오.

다음 링크를 사용하여 오브젝트 열기 및 닫기에 대해 자세히 알아보십시오.

- [764 페이지의 『MQOPEN 호출을 사용하여 오브젝트 열기』](#)

- [771 페이지의 『동적 큐 작성』](#)
- [771 페이지의 『리모트 큐 열기』](#)
- [772 페이지의 『MQCLOSE 호출을 사용하여 오브젝트 닫기』](#)

관련 개념

[744 페이지의 『MQI\(Message Queue Interface\) 개요』](#)

MQI(Message Queue Interface) 컴포넌트에 대해 학습합니다.

[756 페이지의 『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』](#)

IBM MQ 프로그래밍 서비스를 사용하려면 프로그램을 큐 관리자에 연결해야 합니다. 이 정보를 사용하여 큐 관리자에 연결하고 큐 관리자에서 연결을 끊는 방법에 대해 알아보십시오.

[772 페이지의 『큐에 메시지 넣기』](#)

이 정보를 사용하여 메시지를 큐에 넣는 방법에 대해 알아보십시오.

[786 페이지의 『큐에서 메시지 가져오기』](#)

이 정보를 사용하여 큐에서 메시지를 가져오는 방법에 대해 알아봅니다.

[861 페이지의 『오브젝트 속성 조회 및 설정』](#)

속성은 IBM MQ 오브젝트의 특성을 정의하는 특성입니다.

[863 페이지의 『작업 단위 커밋 및 백아웃』](#)

이 정보는 작업 단위에서 발생한 복구 가능한 가져오기 및 넣기 조작을 커밋하고 백아웃하는 방법을 설명합니다.

[873 페이지의 『트리거를 사용한 IBM MQ 애플리케이션 시작』](#)

트리거와 트리거를 사용하여 IBM MQ 애플리케이션을 시작하는 방법에 대해 학습하십시오.

[891 페이지의 『MQI 및 클러스터에 대한 작업』](#)

클러스터와 관련된 호출 및 리턴 코드에 대한 특수 옵션이 있습니다.

[895 페이지의 『IBM MQ for z/OS의 애플리케이션 사용 및 작성』](#)

IBM MQ for z/OS 애플리케이션은 여러 다른 환경에서 실행되는 프로그램으로 구성할 수 있습니다. 이는 둘 이상의 환경에서 사용 가능한 기능을 활용할 수 있음을 의미합니다.

[71 페이지의 『IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션』](#)

이 정보는 IBM MQ(를) 사용하여 IMS 애플리케이션을 작성하는 데 도움이 됩니다.

MQOPEN 호출을 사용하여 오브젝트 열기

이 정보를 사용하여 MQOPEN 호출을 통해 오브젝트를 여는 방법을 알아보십시오.

MQOPEN 호출에 대한 입력으로 다음을 제공해야 합니다.

- 연결 핸들. CICS 애플리케이션에서는 상수 MQHC_DEF_HCONN(값이 0인 상수)을 지정하거나, z/OSMQCONN 또는 MQCONN 호출이 반환하는 연결 핸들을 사용할 수 있습니다. 다중 플랫폼의 경우 항상 MQCONN 또는 MQCONN 호출이 반환하는 연결 핸들을 사용하십시오.
- 오브젝트 디스크립터 구조(MQOD)를 사용하여 열려는 오브젝트의 설명.
- 호출의 조치를 제어하는 하나 이상의 옵션.

MQOPEN의 출력은 다음과 같습니다.

- 오브젝트에 대한 액세스 권한을 나타내는 오브젝트 핸들. 모든 후속 MQI 호출에 대한 입력으로 이 핸들을 사용하십시오.
- 플랫폼에서 지원되는 동적 큐를 작성 중인 경우 수정된 오브젝트 디스크립터 구조.
- 완료 코드.
- 이유 코드.

오브젝트 핸들의 범위

오브젝트 핸들(Hobj)의 범위는 연결 핸들(Hconn)의 범위와 동일합니다.

이 내용은 [758 페이지의 『MQCONN 또는 MQCONN의 범위』](#) 및 [760 페이지의 『MQCONN을 사용한 공유\(스레드 독립\) 연결』](#)에 설명되어 있습니다. 그러나 일부 환경에는 추가적인 고려사항이 있습니다.

CICS

CICS 프로그램에서, MQOPEN 호출을 작성한 태스크와 동일한 CICS 태스크 내에서만 핸들을 사용할 수 있습니다.

IMS 그리고 z/OS 배치

환경에서는 IMS 그리고 z/OS 배치 동일한 작업 내에서 핸들을 사용할 수 있지만, 하위 작업 내에서는 사용할 수 없습니다.

MQOPEN 호출의 매개변수에 대한 설명은 [MQOPEN](#)에 제공됩니다.

다음 절에서는 MQOPEN에 대한 입력으로 제공해야 하는 정보를 설명합니다.

오브젝트 식별(MQOD 구조)

열려는 오브젝트를 식별하려면 MQOD 구조를 사용하십시오. 이 구조는 MQOPEN 호출에 대한 입력 매개변수입니다. (이 구조는 MQOPEN 호출을 사용하여 동적 큐를 작성할 때 큐 관리자에 의해 수정됩니다.)

MQOD 구조에 대한 전체 세부사항은 [MQOD](#)의 내용을 참조하십시오.

분배 목록에 대해 MQOD 구조를 사용하는 방법과 관련된 정보는 [781 페이지](#)의 『분배 목록』 아래에 있는 [783 페이지](#)의 『MQOD 구조 사용』의 내용을 참조하십시오.

이름 해석

큐 관리자가 큐 및 큐 관리자 이름을 해결하는 방법.

참고: 큐 관리자 알리어스는 RNAME 필드가 없는 리모트 큐 정의입니다.

MQOPEN 및 MQPUT1 호출이 큐 및 큐 관리자 이름을 어떻게 해결하는지

IBM MQ 큐를 열면 MQOPEN 호출이 사용자가 지정하는 큐 이름에 대해 이름 해석 기능을 수행합니다. 이 기능은 큐 관리자가 후속 조작을 수행하는 큐를 판별합니다. 이는 오브젝트 디스크립터(MQOD)에 알리어스 큐 또는 리모트 큐의 이름을 지정할 때 호출이 로컬 큐 또는 전송 큐로 이름을 해석하는 것을 의미합니다. 큐가 입력, 찾아보기 또는 설정 중 어떤 유형으로든 열려진 경우, 로컬 큐가 있으면 로컬 큐로 해석하고 없으면 실패합니다. 출력 전용, 조회 전용 또는 출력 및 조회 전용으로 열려진 경우에만 로컬이 아닌 큐로 해석합니다. 이름 해석 프로세스의 개요는 [765 페이지](#)의 표 129의 내용을 참조하십시오. *ObjectQMGrName*에 제공하는 이름은 *ObjectName*의 이름보다 먼저 해석됩니다.

[765 페이지](#)의 표 129는 리모트 큐의 로컬 정의를 사용하여 큐 관리자의 이름에 대한 알리어스를 정의할 수 있는 방법을 보여줍니다. 이 방법으로 리모트 큐에 메시지를 넣을 때 사용되는 전송 큐를 선택할 수 있습니다. 예를 들어, 여러 리모트 큐 관리자로 목적지가 정해진 메시지에 대해 단일 전송 큐를 사용할 수 있습니다.

다음 테이블을 사용하려면 먼저 왼쪽의 두 개 열을 읽고 **MQOD 입력** 표제 아래에서 적절한 경우를 선택하십시오. 그런 다음 지시사항에 따라 해당되는 행을 읽으십시오. **해석된 이름** 열의 지시사항에 따라 **MQOD에 대한 입력** 열로 돌아가거나 지시된 값을 삽입할 수 있습니다. 또는 결과가 표시된 테이블을 종료할 수 있습니다. 예를 들어, *ObjectName*을 입력하도록 요청될 수 있습니다.

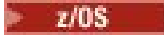
표 129. MQOPEN 및 'MQPUT1' 사용 시 큐 이름 해결				
MQOD에 대한 입력	MQOD에 대한 입력	해석된 이름	해석된 이름	해석된 이름
<i>ObjectQMGrName</i>	<i>ObjectName</i>	<i>ObjectQMGrName</i>	<i>ObjectName</i>	Transmission queue
공백 또는 로컬 큐 관리자	CLUSTER 속성을 포함하지 않은 로컬 큐	로컬 큐 관리자	<i>ObjectName</i> 입력	적용할 수 없음(로컬 큐가 사용됨)
공백 큐 관리자	CLUSTER 속성을 포함한 로컬 큐	워크로드 관리에서 선택한 클러스터 큐 관리자 또는 PUT 실행 시 선택된 특정 클러스터 큐 관리자	<i>ObjectName</i> 입력	SYSTEM.CLUSTER.TRANSMIT.QUEUE 및 사용된 로컬 큐 SYSTEM.QSG.TRANSMIT.QUEUE(참고 참조)

표 129. MQOPEN 및 'MQPUT1' 사용 시 큐 이름 해결 (계속)

MQOD에 대한 입력	MQOD에 대한 입력	해석된 이름	해석된 이름	해석된 이름
로컬 큐 관리자	CLUSTER 속성을 포함한 로컬 큐	로컬 큐 관리자	ObjectName 입력	적용할 수 없음(로컬 큐가 사용됨)
공백 또는 로컬 큐 관리자	모델 큐	로컬 큐 관리자	생성된 이름	적용할 수 없음(로컬 큐가 사용됨)
공백 또는 로컬 큐 관리자	CLUSTER 속성을 포함하거나 포함하지 않은 알리어스 큐	ObjectQMgrName을 변경하지 않고 이름 해석을 다시 수행하고 알리어스 큐 정의 오브젝트에서 BaseQName으로 설정된 ObjectName을 입력하십시오. ObjectQMgrName이 지정된 경우 로컬에 정의된 알리어스로 해석하지 않아야 하지만, ObjectQMgrName이 공백인 경우 클러스터된 알리어스(다른 큐 관리자에서 호스팅됨)로 해석할 수 있습니다.		
로컬 큐 관리자	CLUSTER 속성을 포함한 알리어스 큐	알리어스는 로컬로 정의되지 않은 클러스터 큐 또는 알리어스와 동일한 ObjectName을 갖는 클러스터 큐로 해석하지 않아야 합니다.		
공백 큐 관리자	CLUSTER 속성을 포함한 알리어스 큐	이 알리어스는 자신과 동일한 ObjectName을 갖는 클러스터 큐로 해석할 수 있습니다.		
공백 또는 로컬 큐 관리자	리모트 큐의 로컬 정의	ObjectQMgrName을 RemoteQMgrName으로 설정하고 ObjectName을 RemoteQName으로 설정하여 이름 해석을 다시 수행하십시오. 리모트 큐로 해석하지 않아야 합니다.		공백이 아닌 경우 XmitQName 속성의 이름, 그렇지 않으면 리모트 큐 정의 오브젝트의 RemoteQMgrName입니다. SYSTEM.QSG.TRANSMIT.QUEUE(참고 참조)
공백 큐 관리자	일치하는 로컬 오브젝트가 없음. 클러스터 큐를 찾음	워크로드 관리에서 선택한 클러스터 큐 관리자 또는 PUT 실행 시 선택된 특정 클러스터 큐 관리자	ObjectName 입력	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE(참고 참조)

표 129. MQOPEN 및 'MQPUT1' 사용 시 큐 이름 해결 (계속)				
MQOD에 대한 입력	MQOD에 대한 입력	해석된 이름	해석된 이름	해석된 이름
공백 또는 로컬 큐 관리자	일치하는 로컬 오브젝트가 없음. 클러스터 큐를 찾을 수 없음		오류, 큐를 찾을 수 없음	적용할 수 없음
로컬 큐 관리자와 동일한 큐 공유 그룹의 큐 관리자 이름	로컬 공유 큐	로컬 큐 관리자	ObjectName 입력	적용할 수 없음
로컬 전송 큐의 이름	(해석되지 않음)	ObjectQMgrName 입력	ObjectName 입력	ObjectQMgrName 입력 SYSTEM.QSG.TRANSMIT.QUEUE(참고 참조)
큐 관리자 알리어스 정의 (RemoteQMgrName이 로컬 큐 관리자일 수 있음)	(해석되지 않음, 리모트 큐)	ObjectQMgrName을 RemoteQMgrName으로 설정하고 이름 해석을 다시 수행하십시오. 리모트 큐로 해석하지 않아야 합니다.	ObjectName 입력	공백이 아닌 경우 XmitQName 속성의 이름, 그렇지 않으면 리모트 큐 정의 오브젝트의 RemoteQMgrName입니다. SYSTEM.QSG.TRANSMIT.QUEUE(참고 참조)
큐 관리자가 로컬 오브젝트의 이름이 아님. 클러스터 큐 관리자 또는 큐 관리자 알리어스를 찾음	(해석되지 않음)	ObjectQMgrName 또는 PUT 실행 시 선택된 특정 클러스터 큐 관리자	ObjectName 입력	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE(참고 참조)
큐 관리자가 로컬 오브젝트의 이름이 아님. 클러스터 오브젝트를 찾을 수 없음	(해석되지 않음)	ObjectQMgrName 입력	ObjectName 입력	DefXmitQName이 지원되는 큐 관리자의 DefXmitQName 속성. SYSTEM.QSG.TRANSMIT.QUEUE(참고 참조)

참고:

1. BaseQName 은 별칭 큐의 정의에서 기본 큐의 이름입니다.
2. RemoteQName은 리모트 큐의 로컬 정의에서 리모트 큐의 이름입니다.
3. RemoteQMgrName은 리모트 큐의 로컬 정의에서 리모트 큐 관리자의 이름입니다.
4. XmitQName은 리모트 큐의 로컬 정의에서 전송 큐의 이름입니다.
5.  큐 공유 그룹(QSG)의 일부인 큐 관리자의 IBM MQ for z/OS 경우, 로컬 큐 관리자 이름 대신 큐 공유 그룹의 이름을 사용할 수 있습니다 765 페이지의 표 129. 로컬 큐 관리자가 대상 큐를 열거나 메시지를 큐에 넣을 수 없는 경우, 메시지가 인트라 그룹 큐잉 또는 IBM MQ 채널을 통해 지정된 ObjectQMgrName으로 전송됩니다.
6. 테이블의 ObjectName 열에서 CLUSTER는 큐의 CLUSTER 및 CLUSNL 속성을 모두 나타냅니다.
7. 로컬 및 리모트 큐 관리자가 동일한 큐 공유 그룹에 있는 경우 SYSTEM.QSG.TRANSMIT.QUEUE가 사용되고 그룹 내 큐잉이 사용하도록 설정됩니다.
8. 각 클러스터 송신자 채널에 서로 다른 클러스터 전송 큐를 지정한 경우, SYSTEM.CLUSTER.TRANSMIT.QUEUE가 클러스터 전송 큐의 이름이 아닐 수 있습니다. 다중 클러스터 전송 큐에 대한 자세한 정보는 [클러스터링: 클러스터 전송 큐를 구성하는 방법 계획](#)의 내용을 참조하십시오.

9. 큐 관리자가 로컬 오브젝트의 이름이 아니고 클러스터 큐 관리자 또는 큐 관리자 알리어스를 찾은 경우입니다.

ObjectQMgrName을 사용하여 큐 관리자 이름을 제공하고 해당 목적지에 도달할 수 있는 로컬 큐 관리자에 알려진 다른 클러스터 이름을 가진 여러 채널이 있는 경우, 이러한 채널은 목적지 큐의 클러스터 이름과 상관없이 메시지를 이동하는 데 사용할 수 있습니다.

이는 해당 큐만을 위한 메시지가 큐와 동일한 클러스터 이름을 가진 채널을 통해 송신될 것으로 생각한 경우 예상치 못한 상황일 수 있습니다.

그러나 이 경우에 **ObjectQMgrName**이 우선순위를 가지며, 채널이 있는 클러스터 이름과 상관없이 해당 큐 관리자에 도달할 수 있는 모든 채널에 대해 클러스터 워크로드 밸런싱이 고려됩니다.

알리어스 큐를 열면 알리어스가 해석되는 기본 큐가 열리고, 리모트 큐를 열면 전송 큐도 함께 열립니다. 따라서 다른 큐가 열려 있는 동안에는 사용자가 지정하는 큐 또는 이 큐가 해석하는 큐를 삭제할 수 없습니다.

알리어스 큐는 로컬로 정의된 다른 알리어스 큐(클러스터에서 공유되거나 공유되지 않는 큐)로 해석될 수 없는 반면, 리모트로 정의된 클러스터 알리어스 큐로 해석될 수 있으므로 기본 큐로 지정할 수 있습니다.

해석된 큐 이름과 해석된 큐 관리자 이름은 MQOD의 *ResolvedQName* 및 *ResolvedQMgrName* 필드에 저장됩니다.

분산 큐잉 환경에서의 이름 해석에 대한 자세한 정보는 [큐 이름 해석의 정의](#)의 내용을 참조하십시오.

큐 매니저가 리포트 메시지를 배치할 때 큐 및 큐 매니저 이름을 해결하는 특수 사례

보고 메시지의 목적지를 선택할 때 따르는 규칙은 MQOPEN 및 MQPUT1의 규칙과 유사합니다.

보고 메시지의 목적지를 선택하기 위해, 보고 메시지를 생성하는 큐 관리자는 수신 메시지의 메시지 디스크립터 내 및 **ReplyToQMgr ReplyToQ** 필드를 참조합니다. 이는 MQOD가 없기 때문입니다.

메시지를 처음 넣은 큐 관리자가 애플리케이션이 빈 값으로 두었을 경우 **MQMD.ReplyToQMgr** 기본값을 부여한다는 점을 기억하십시오.

보고서 메시지가 생성되는 동안에는 절대 비어 있지 **MQMD.ReplyToQMgr** 않습니다. 따라서 MQOPEN 또는 MQOPEN_REQUIRED(MQPUT1) 시 목적지를 선택하는 것과 비교하여, 보고서 메시지의 목적지를 선택할 때는 더 적은 수의 조건만이 적용됩니다.

보고서 메시지가 생성되는 큐 관리자의 이름이 **MQMD.ReplyToQMgr** 인 경우, 이는 특수한 경우로 처리됩니다.

이 특별한 경우, 이름 해결에는 큐 관리자가 인식하는 클러스터 내에서 생성 중인 보고서 메시지와 일치하는 큐 이름을 찾는 **MQMD.ReplyToQ** 작업이 포함됩니다.

이 특별한 경우에, 클러스터 조회로 **MQMD.ReplyToQ** 를 해결하려고 시도했을 때 결과가 없을 때에만 **MQMD.ReplyToQMgr** 사용됩니다.

보고서 메시지가 생성되는 큐 관리자의 이름이 **MQMD.ReplyToQMgr** 아니라면, 는 **MQMD.ReplyToQ** 무시되고 **MQMD.ReplyToQMgr** 사용하여 대상 위치를 찾으려고 시도합니다.

MQOPEN 호출의 옵션 사용

MQOPEN 호출의 **options** 매개변수에서, 열려는 객체에 대한 접근 권한을 제어하기 위해 하나 이상의 옵션을 선택해야 합니다.

다음 MQOPEN 호출 옵션 중에서 선택할 수 있습니다:

- 큐를 열고, 해당 큐에 넣은 모든 메시지가 동일한 큐 인스턴스로 전달되어야 함을 지정합니다.
- 메시지를 넣을 큐를 엽니다.
- 메시지를 찾아볼 큐를 엽니다.
- 메시지를 제거할 큐를 엽니다.
- 속성을 조회하고 설정할 오브젝트를 엽니다(그러나 큐의 속성만 설정할 수 있음).
- 메시지를 발행할 주제 또는 주제 문자열을 엽니다.
- 컨텍스트 정보를 메시지와 연관시킵니다.

- 보안 검사에 사용할 대체 사용자 ID를 지정합니다.
- 큐 관리자가 정지 중 상태인 경우 호출을 제어합니다.

클러스터 큐에 대한 MQOPEN 옵션

큐를 열고, 해당 큐에 넣는 모든 메시지가 동일한 인스턴스로 전달되어야 한다고 지정합니다.

큐 핸들에 사용되는 바인딩은 큐 **DefBind** 속성에서 가져오며, 이 속성은, MQBND_BIND_NOT_FIXED, 또는 MQBND_BIND_ON_OPEN 값을 가질 MQBND_BIND_ON_GROUP 수 있습니다.

MQPUT을 사용하여 큐에 넣은 모든 메시지를 동일한 라우트를 통해 동일한 큐 관리자로 라우트하려면 MQOPEN 호출 시 MQOO_BIND_ON_OPEN 옵션을 사용하십시오.

MQPUT을 실행할 때, 즉 메시지 기준으로 목적지가 선택되도록 지정하려면 MQOPEN 호출 시 MQOO_BIND_NOT_FIXED 옵션을 사용하십시오.

MQPUT을 사용하여 큐에 넣은 메시지 그룹의 모든 메시지에 동일한 목적지 인스턴스를 할당하려면 MQOPEN 호출 시 MQOO_BIND_ON_GROUP 옵션을 사용하십시오.

클러스터에서 메시지 그룹을 사용할 때는 그룹 내의 모든 메시지가 동일한 대상에서 처리되도록 하기 위해 또는 MQOO_BIND_ON_GROUP 중 MQOO_BIND_ON_OPEN 하나를 지정해야 합니다.

이러한 옵션 중 어느 것도 지정하지 않으면 기본값이 MQOO_BIND_AS_Q_DEF 사용됩니다.

MQOD에서 큐 관리자의 이름을 지정하면 해당 큐 관리자의 큐가 선택됩니다. 큐 관리자 이름이 공백이면 인스턴스를 선택할 수 있습니다. 자세한 정보는 [891 페이지의 『MQOPEN 및 클러스터』](#)의 내용을 참조하십시오.

QALIAS 정의를 사용하여 클러스터 큐를 여는 경우 일부 큐 속성이 기본 큐가 아닌 알리어스 큐에 의해 정의됩니다. 클러스터 속성은 알리어스 큐에 의해 대체된 기본 큐 정의의 속성 중 일부입니다. 예를 들어, 다음 스니펫에서 클러스터 큐는 MQOO_BIND_ON_OPEN이 아닌 MQOO_BIND_NOT_FIXED를 사용하여 열립니다. 클러스터 큐 정의는 클러스터 전체에 걸쳐 광고되며, 별칭 큐 정의는 큐 관리자에 국한됩니다.

```
DEFINE QLOCAL(CLQ1) CLUSTER(MYCLUSTER) DEFBIND(OPEN) REPLACE
DEFINE QALIAS(ACLQ1) TARGET(CLQ1) DEFBIND(NOTFIXED) REPLACE
```

메시지를 넣기 위한 MQOPEN 옵션

메시지를 넣을 수 있도록 큐 또는 토픽을 열려면 MQOO_OUTPUT 옵션을 사용하십시오.

메시지를 찾아보기 위한 MQOPEN 옵션

메시지를 찾아볼 수 있도록 큐를 열려면 MQOPEN 호출에 MQOO_BROWSE 옵션을 사용하십시오.

이 옵션을 지정하면 큐 관리자가 큐의 다음 메시지를 식별하는 데 사용하는 찾아보기 커서가 작성됩니다. 자세한 정보는 [817 페이지의 『큐의 메시지 찾아보기』](#)의 내용을 참조하십시오.

참고:

1. 리모트 큐의 메시지는 찾아볼 수 없습니다. MQOO_BROWSE 옵션을 사용하여 리모트 큐를 열지 마십시오.
2. 분배 목록을 열 때 이 옵션을 지정할 수 없습니다. 분배 목록에 대한 자세한 정보는 [781 페이지의 『분배 목록』](#)의 내용을 참조하십시오.
3. 협업 찾아보기를 사용하는 경우 MQOO_BROWSE와 함께 MQOO_CO_OP를 사용하십시오. [옵션](#)을 참조하십시오.

메시지 제거를 위한 MQOPEN 옵션

세 가지 옵션으로 메시지를 제거하기 위해 큐를 여는 동작을 제어합니다.

모든 MQOPEN 호출에서 옵션 하나만 사용할 수 있습니다. 이러한 옵션은 프로그램이 큐에 독점 액세스 또는 공유 액세스를 갖는지 여부를 정의합니다. 독점 액세스는 큐를 닫을 때까지 사용자만 메시지를 제거할 수 있음을 의미합니다. 다른 프로그램이 메시지를 제거하기 위해 큐를 열려고 하면 해당 MQOPEN 호출이 실패합니다. 공유 액세스는 두 개 이상의 프로그램이 큐에서 메시지를 제거할 수 있음을 의미합니다.

가장 추천되는 접근 방법은 큐가 정의될 때 큐에 사용할 목적으로 작성된 액세스 유형을 승인하는 것입니다. 큐 정의에는 **Shareability** 및 **DefInputOpenOption** 속성에 대한 설정이 포함되었습니다. 이 액세스를 허용하려면 MQOO_INPUT_AS_Q_DEF 옵션을 사용하십시오. 이러한 속성이 이 옵션을 사용할 시기가 지정된 액세스 유형에 어떤 영향을 주는지 확인하려면 [770 페이지의 표 130](#)의 내용을 참조하십시오.

표 130. MQOPEN 호출의 옵션 및 큐 속성이 큐에 대한 액세스에 주는 영향				
큐 속성		MQOPEN 옵션을 사용한 액세스 유형		
Shareability	DefInputOpenOption	AS_Q_DEF	SHARED	EXCLUSIVE
SHAREABLE	SHARED	공용	공용	독점
SHAREABLE	EXCLUSIVE	독점	공용	독점
NOT_SHAREABLE*	SHARED*	독점	독점	독점
NOT_SHAREABLE	EXCLUSIVE	독점	독점	독점

참고: * 이 속성 결합을 갖도록 큐를 정의할 수 있지만, 기본 입력 열기 옵션은 Sareability 속성에 의해 대체됩니다.

다른 방법은 다음과 같습니다.

- 다른 프로그램이 큐에서 메시지를 동시에 제거할 수 있는 경우에도 애플리케이션이 성공적으로 작동할 수 있다는 것을 알고 있는 경우 MQOO_INPUT_SHARED 옵션을 사용하십시오. [770 페이지의 표 130](#)에서는 몇 가지 경우에 이 옵션을 사용할 때에도 큐에 대한 독점 권한이 제공되는 방법을 보여줍니다.
- 다른 프로그램이 큐에서 메시지를 동시에 제거하지 못하게 하는 경우에만 애플리케이션이 성공적으로 작동할 수 있다는 것을 알고 있는 경우 MQOO_INPUT_EXCLUSIVE 옵션을 사용하십시오.

참고:

1. 리모트 큐에서는 메시지를 제거할 수 없습니다. 따라서 MQOO_INPUT_* 옵션을 사용하여 리모트 큐를 열 수 없습니다.
2. 분배 목록을 열 때 이 옵션을 지정할 수 없습니다. 자세한 정보는 [781 페이지의 『분배 목록』](#)의 내용을 참조하십시오.

속성 설정 및 조회를 위한 MQOPEN 옵션

속성을 설정하기 위해 큐를 열려면 MQOO_SET 옵션을 사용하십시오.

다른 유형의 오브젝트에 대한 속성을 설정할 수 없습니다([861 페이지의 『오브젝트 속성 조회 및 설정』](#) 참조).

속성을 조회하기 위해 오브젝트를 열려면 MQOO_INQUIRE 옵션을 사용하십시오.

참고: 분배 목록을 열 때 이 옵션을 지정할 수 없습니다.

메시지 컨텍스트와 관련된 MQOPEN 옵션

메시지를 큐에 넣을 때 컨텍스트 정보를 메시지와 연관시키려면, 큐를 열 때 메시지 컨텍스트 옵션 중 하나를 사용해야 합니다.

이 옵션을 사용하면 메시지를 생성한 사용자와 관련된 컨텍스트 정보와 메시지를 생성한 애플리케이션과 관련된 컨텍스트 정보를 구별할 수 있습니다. 또한 메시지를 큐에 넣을 때 컨텍스트 정보 설정을 선택하거나, 다른 큐 핸들에서 자동으로 가져온 컨텍스트를 갖도록 선택할 수 있습니다.

자세한 내용은 [메시지 컨텍스트](#) 및 [메시지 컨텍스트 정보 제어](#) 항목을 참조하십시오.

대체 사용자 권한에 대한 MQOPEN 옵션

MQOPEN 호출을 사용하여 오브젝트를 열려고 하는 경우 큐 관리자는 해당 오브젝트를 열기 위한 권한이 사용자에게 있는지 검사합니다. 권한이 부여되지 않은 경우 호출이 실패합니다.

그러나 서버 프로그램은 서버의 고유 권한 대신 큐 관리자가 프로그램을 실행하는 사용자의 권한을 검사하기를 원할 수 있습니다. 이를 수행하려면 MQOPEN 호출의 MQOO_ALTERNATE_USER_AUTHORITY 옵션을 사용하고, 대체 사용자 ID를 MQOD 구조의 *AlternateUserId* 필드에 지정해야 합니다. 일반적으로, 서버는 처리 중인 메시지의 컨텍스트 정보에서 사용자 ID를 가져옵니다.

정지 중인 큐 관리자에 대한 MQOPEN 옵션

큐 관리자가 정지 중 상태일 때 MQOPEN 호출을 사용하는 경우 사용 중인 환경에 따라 호출이 실패할 수 있습니다.

환경에서 z/OS CICS, 큐 관리자가 정지 상태일 때 MQOPEN 호출을 사용하면 호출이 항상 실패합니다.

기타 z/OS 및 멀티플랫폼 환경에서는 MQOPEN 호출의 MQOO_FAIL_IF_QUIESCING 옵션을 사용하는 경우에만 큐 관리자가 작업을 거부할 때 호출이 실패합니다.

로컬 큐 이름을 해석하기 위한 MQOPEN 옵션

로컬, 알리어스 또는 모델 큐를 열면 로컬 큐가 리턴됩니다.

그러나 리모트 큐 또는 클러스터 큐를 열면 MQOD 구조의 *ResolvedQName* 및 *ResolvedQMgrName* 필드에는 리모트 큐 정의에 있는 리모트 큐 및 리모트 큐 관리자의 이름 또는 선택한 리모트 클러스터 큐가 입력됩니다.

MQOPEN 호출의 MQOO_RESOLVE_LOCAL_Q 옵션을 사용하여 MQOD 구조의 *ResolvedQName*을(를) 열린 로컬 큐의 이름으로 채우십시오. *ResolvedQMgrName*에는 로컬 큐를 호스팅하는 로컬 큐 관리자의 이름으로 유사하게 입력됩니다. 이 필드는 MQOD 구조의 버전 3에서만 사용할 수 있으며, 구조가 버전 3보다 낮은 경우 오류가 리턴되지 않고 MQOO_RESOLVE_LOCAL_Q가 무시됩니다.

예를 들어, 리모트 큐를 열 때 MQOO_RESOLVE_LOCAL_Q를 지정하는 경우, *ResolvedQName*은(는) 메시지를 넣을 전송 큐의 이름입니다. *ResolvedQMgrName*은(는) 전송 큐를 호스팅하는 로컬 큐 관리자의 이름입니다.

동적 큐 작성

애플리케이션이 종료된 후 큐가 필요하지 않을 때 동적 큐를 사용하십시오.

예를 들어, 응답 대상 큐에 동적 큐를 사용할 수 있습니다. 메시지를 큐에 넣을 때 MQMD 구조의 *ReplyToQ* 필드에 응답 대상 큐의 이름을 지정합니다(774 페이지의 『MQMD 구조를 사용하여 메시지 정의』 참조).

동적 큐를 작성하려면 모델 큐로 알려진 템플릿을 MQOPEN 호출과 함께 사용하십시오. 모델 큐는 IBM MQ 명령이나 조작 및 제어 패널을 사용하여 작성합니다. 사용자가 작성하는 동적 큐는 모델 큐의 속성을 가져옵니다.

MQOPEN을 호출할 때 모델 큐의 이름을 MQOD 구조의 *ObjectName* 필드에 지정하십시오. 호출이 완료되면 *ObjectName* 필드는 작성된 동적 큐의 이름으로 설정됩니다. 또한 *ObjectQMgrName* 필드는 로컬 큐 관리자의 이름으로 설정됩니다.

다음 세 가지 방법으로 작성하는 동적 큐의 이름을 지정할 수 있습니다.

- MQOD 구조의 *DynamicQName* 필드에 원하는 전체 이름을 제공하십시오.
- 이름의 접두부(33자 미만)를 지정하고 큐 관리자가 이름의 나머지 부분을 생성하게 하십시오. 이는 큐 관리자가 고유한 이름을 생성하지만 여전히 얼마간의 제어 권한을 가질 수 있음을 의미합니다. 예를 들어, 각 사용자가 특정 접두부를 사용하게 하거나 특정 접두부를 이름에 넣어 큐에 특수 보안 분류를 제공할 수 있습니다. 이 방법을 사용하려면 *DynamicQName* 필드의 마지막 비공백 문자에 별표(*)를 지정하십시오. 동적 큐 이름에는 단일 별표(*)를 지정하지 마십시오.
- 큐 관리자가 전체 이름을 생성하게 하십시오. 이 방법을 사용하려면 *DynamicQName* 필드의 첫 번째 문자 위치에 별표(*)를 지정하십시오.

이러한 방법에 대한 자세한 정보는 *DynamicQName* 필드의 설명을 참조하십시오.

동적 및 모델 큐에 동적 큐에 대한 자세한 정보가 있습니다.

리모트 큐 열기

리모트 큐는 애플리케이션이 연결되는 큐가 아닌 큐 관리자가 소유하는 큐입니다.

리모트 큐를 열려면 로컬 큐와 마찬가지로 MQOPEN 호출을 사용하십시오. 다음과 같이 큐의 이름을 지정할 수 있습니다.

1. MQOD 구조의 *ObjectName* 필드에서 로컬 큐 관리자에 알려진 대로 리모트 큐의 이름을 지정하십시오.

참고: 이 경우 *ObjectQMGrName* 필드를 공백으로 두십시오.

2. MQOD 구조의 *ObjectName* 필드에서 리모트 큐 관리자에 알려진 대로 리모트 큐의 이름을 지정하십시오. *ObjectQMGrName* 필드에서 다음 중 하나를 지정하십시오.

- 리모트 큐 관리자와 동일한 이름을 가진 전송 큐의 이름. 이름 및 대소문자(대문자, 소문자 또는 대소문자 혼합)가 정확히 일치해야 합니다.
- 목적지 큐 관리자 또는 전송 큐에서 해석되는 큐 관리자 알리어스 오브젝트의 이름.

큐 관리자에게 메시지의 목적지뿐만 아니라 이 목적지에 도달하기 위해 넣어야 하는 전송 큐를 알려줍니다.

3. *DefXmitQname*이(가) 지원되는 경우, MQOD 구조의 *ObjectName* 필드에서 리모트 큐 관리자에 알려진 대로 리모트 큐의 이름을 지정하십시오.

참고: *ObjectQMGrName* 필드를 리모트 큐 관리자의 이름으로 설정하십시오(이 경우 공백으로 둘 수 없음).

MQOPEN을 호출할 때 로컬 이름만 유효성이 검증됩니다. 마지막 검사는 사용될 전송 큐가 있는지 여부를 확인합니다.

이러한 방법은 765 페이지의 표 129에 요약되어 있습니다.

MQCLOSE 호출을 사용하여 오브젝트 닫기

오브젝트를 닫으려면 MQCLOSE 호출을 사용하십시오.

오브젝트가 큐인 경우 다음을 참고하십시오.

- 임시 동적 큐를 닫기 전에 큐를 비울 필요는 없습니다.

임시 동적 큐를 닫으면 큐에 아직 있을 수 있는 모든 메시지와 함께 큐가 삭제됩니다. 이 동작은 큐에 대해 미해결 상태의 커미트되지 않은 MQGET, MQPUT 또는 MQPUT1 호출이 있는 경우 수행됩니다.

-  IBM MQ for z/OS에서, 해당 큐에 대해 미해결 상태의 MQGMO_SET_SIGNAL 옵션이 포함된 MQGET 요청이 있는 경우 해당 요청이 취소됩니다.
- MQOO_BROWSE 옵션을 사용하여 큐를 연 경우 브라우저 커서가 영구 삭제됩니다.

닫기는 동기점과 관련이 없으므로 동기점 앞 또는 뒤의 큐를 닫을 수 있습니다.

MQCLOSE 호출의 입력으로 다음을 제공해야 합니다.

- 연결 핸들. 열 때 사용한 것과 동일한 연결 핸들을 사용하십시오. CICS 애플리케이션에서 z/OS, 대신 상수 MQHC_DEF_HCONN(값이 0인 상수)를 지정할 수 있습니다.
- 닫으려는 오브젝트의 핸들. MQOPEN 호출의 출력에서 이 핸들을 가져오십시오.
- *Options* 필드의 MQCO_NONE(영구적 동적 큐를 닫는 중이 아닌 경우).
- (영구적 동적 큐를 닫을 때) 아직 메시지가 있는 경우에도 큐 관리자가 큐를 삭제해야 하는지 여부를 판별하는 제어 옵션.

MQCLOSE의 출력은 다음과 같습니다.

- 완료 코드
- 이유 코드
- MQHO_UNUSABLE_HOBJ 값으로 재설정된 오브젝트 핸들

MQCLOSE 호출의 매개변수에 대한 설명은 [MQCLOSE](#)에 있습니다.

큐에 메시지 넣기

이 정보를 사용하여 메시지를 큐에 넣는 방법에 대해 알아보십시오.

메시지를 큐에 넣으려면 MQPUT 호출을 사용하십시오. 초기 MQOPEN 호출 후에 MQPUT을 반복적으로 사용하여 많은 메시지를 동일한 큐에 넣을 수 있습니다. 모든 메시지를 큐에 넣는 작업을 완료한 경우 MQCLOSE를 호출하십시오.

단일 메시지를 큐에 넣은 후에 큐를 즉시 닫으려는 경우 MQPUT1 호출을 사용할 수 있습니다. MQPUT1은 다음 호출 순서와 동일한 기능을 수행합니다.

- MQOPEN
- MQPUT
- MQCLOSE

그러나 일반적으로 큐에 넣을 메시지가 두 개 이상인 경우에는 MQPUT 호출을 사용하는 것이 더욱 효율적입니다. 어떤 호출을 사용할 것인지는 메시지의 크기와 작업을 수행 중인 플랫폼에 달려 있습니다.

다음 링크를 사용하여 메시지를 큐에 넣는 방법에 대해 자세히 알아보십시오.

- [773 페이지의 『MQPUT 호출을 사용하여 로컬 큐에 메시지 넣기』](#)
- [778 페이지의 『리모트 큐에 메시지 넣기』](#)
- [778 페이지의 『메시지의 특성 설정』](#)
- [778 페이지의 『메시지 컨텍스트 정보 제어』](#)
- [780 페이지의 『MQPUT1 호출을 사용하여 큐에 하나의 메시지 넣기』](#)
- [781 페이지의 『분배 목록』](#)
- [786 페이지의 『Put 호출이 실패하는 몇 가지 사례』](#)

관련 개념

[744 페이지의 『MQI\(Message Queue Interface\) 개요』](#)
MQI(Message Queue Interface) 컴포넌트에 대해 학습합니다.

[756 페이지의 『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』](#)
IBM MQ 프로그래밍 서비스를 사용하려면 프로그램을 큐 관리자에 연결해야 합니다. 이 정보를 사용하여 큐 관리자에 연결하고 큐 관리자에서 연결을 끊는 방법에 대해 알아보십시오.

[763 페이지의 『오브젝트 열기 및 닫기』](#)
이 정보는 IBM MQ 오브젝트의 열기 및 닫기 조작에 대한 통찰력을 제공합니다.

[786 페이지의 『큐에서 메시지 가져오기』](#)
이 정보를 사용하여 큐에서 메시지를 가져오는 방법에 대해 알아봅니다.

[861 페이지의 『오브젝트 속성 조회 및 설정』](#)
속성은 IBM MQ 오브젝트의 특성을 정의하는 특성입니다.

[863 페이지의 『작업 단위 커밋 및 백아웃』](#)
이 정보는 작업 단위에서 발생한 복구 가능한 가져오기 및 넣기 조작을 커밋하고 백아웃하는 방법을 설명합니다.

[873 페이지의 『트리거를 사용한 IBM MQ 애플리케이션 시작』](#)
트리거와 트리거를 사용하여 IBM MQ 애플리케이션을 시작하는 방법에 대해 학습하십시오.

[891 페이지의 『MQI 및 클러스터에 대한 작업』](#)
클러스터와 관련된 호출 및 리턴 코드에 대한 특수 옵션이 있습니다.

[895 페이지의 『IBM MQ for z/OS의 애플리케이션 사용 및 작성』](#)
IBM MQ for z/OS 애플리케이션은 여러 다른 환경에서 실행되는 프로그램으로 구성할 수 있습니다. 이는 둘 이상의 환경에서 사용 가능한 기능을 활용할 수 있음을 의미합니다.

[71 페이지의 『IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션』](#)
이 정보는 IBM MQ(를) 사용하여 IMS 애플리케이션을 작성하는 데 도움이 됩니다.

MQPUT 호출을 사용하여 로컬 큐에 메시지 넣기

이 정보를 사용하여 MQPUT 호출을 통해 로컬 큐에 메시지를 넣는 방법을 알아보십시오.

MQPUT 호출에 대한 입력으로 다음을 제공해야 합니다.

- 연결 핸들(Hconn).

- 큐 핸들(Hobj).
- 큐에 넣으려는 메시지의 설명. 메시지 디스크립터 구조(MQMD)의 양식으로 되어 있습니다.
- 메시지 넣기 옵션 구조(MQPMO) 양식의 제어 정보.
- 메시지에 포함된 데이터의 길이(MQLONG)
- 메시지 데이터 자체

MQPUT 호출의 출력은 다음과 같습니다.

- 이유 코드(MQLONG)
- 완료 코드(MQLONG)

호출이 성공적으로 완료되는 경우 옵션 구조 및 메시지 디스크립터 구조도 리턴됩니다. 호출은 사용자의 옵션 구조를 수정하여 메시지가 송신된 큐 관리자 및 큐의 이름을 표시합니다. 사용자가 넣고 있는 메시지의 ID에 대해 큐 관리자가 고유한 값을 생성하도록 요청하는 경우(MQMD 구조의 *MsgId* 필드에 2진 0을 지정하여), 호출은 이 구조를 리턴하기 전에 *MsgId* 필드에 값을 삽입합니다. 다른 MQPUT을 발행하기 전에 이 값을 재설정하십시오.

MQPUT 호출에 대한 설명은 [MQPUT](#)에 있습니다.

MQPUT 호출의 입력으로 필요한 정보에 대한 자세한 설명은 다음 링크를 참조하십시오.

- [774 페이지의 『핸들 지정』](#)
- [774 페이지의 『MQMD 구조를 사용하여 메시지 정의』](#)
- [774 페이지의 『MQPMO 구조를 사용하여 옵션 지정』](#)
- [777 페이지의 『메시지의 데이터』](#)
- [778 페이지의 『메시지 넣기: 메시지 핸들 사용』](#)

핸들 지정

연결 핸들(Hconn)을 CICS 지정할 z/OS 때, 상수 MQHC_DEF_HCONN(값은 0)을 사용하거나 MQCONN 또는 MQCONNX 호출이 반환한 연결 핸들을 사용할 수 있습니다. 기타 애플리케이션의 경우, MQCONN 또는 MQCONNX 호출에 의해 리턴되는 연결 핸들을 항상 사용하십시오.

작업을 수행하는 환경과 상관없이 MQOPEN 호출에서 리턴된 동일한 큐 핸들(Hobj)을 사용하십시오.

MQMD 구조를 사용하여 메시지 정의

메시지 디스크립터 구조(MQMD)는 MQPUT 및 MQPUT1 호출에 대한 입/출력 매개변수입니다. 큐에 넣는 중인 메시지를 정의하려면 이 구조를 사용하십시오.

메시지에 MQPRI_PRIORITY_AS_Q_DEF 또는 MQPER_PERSISTENCE_AS_Q_DEF가 지정되고 큐가 클러스터 큐인 경우, 사용되는 값은 MQPUT이 해석하는 큐의 값입니다. 해당 큐가 MQPUT에 대해 사용 불가능한 경우 호출이 실패하게 됩니다. 자세한 정보는 [큐 관리자 클러스터 구성의 내용](#)을 참조하십시오.

참고: 새 메시지를 넣기 전에 MQPMO_NEW_MSG_ID 및 MQPMO_NEW_CORREL_ID를 사용하여 *MsgId* 및 *CorrelId*가 고유한지 확인하십시오. 이러한 필드의 값은 MQPUT이 성공하면 리턴됩니다.

[18 페이지의 『IBM MQ 메시지』](#)에는 MQMD가 설명하는 메시지 특성에 대한 소개가 있고, [MQMD](#)에는 구조 자체에 대한 설명이 있습니다.

MQPMO 구조를 사용하여 옵션 지정

MQPUT 및 MQPUT1 호출로 옵션을 전달하려면 MQPMO(메시지 넣기 옵션) 구조를 사용하십시오.

다음 절에는 이 구조의 필드를 채우는 방법에 대한 도움말이 나와 있습니다. 구조에 대한 설명은 [MQPMO](#)에 있습니다.

구조에는 다음 필드가 포함됩니다.

- *StrucId*
- *Version*

- *Options*
- *Context*
- *ResolvedQName*
- *ResolvedQMgrName*
- *RecsPresent*
- *PutMsgRecsFields*
- *ResponseRecOffset* and *ResponseRecPtr*
- *OriginalMsgHandle*
- *NewMsgHandle*
- *Action*
- *PubLevel*

이러한 필드의 콘텐츠는 다음과 같습니다.

StrucId

구조를 메시지 넣기 옵션 구조로 식별합니다. 4자를 입력하는 필드입니다. 항상 MQPMO_STRUC_ID를 지정하십시오.

버전

구조의 버전 번호를 설명합니다. 기본값은 MQPMO_VERSION_1입니다. MQPMO_VERSION_2를 입력하면 분배 목록을 사용할 수 있습니다(781 페이지의 『분배 목록』 참조). MQPMO_VERSION_3을 입력하면 메시지 핸들 및 메시지 특성을 사용할 수 있습니다. MQPMO_CURRENT_VERSION을 입력하면 애플리케이션이 항상 최신 레벨을 사용하도록 설정됩니다.

옵션

이 필드는 다음을 제어합니다.

- 넣기 조작이 작업 단위에 포함되는지 여부
- 컨텍스트 정보가 메시지와 연관되는 양
- 컨텍스트 정보를 가져오는 위치
- 큐 관리자가 정지 중 상태에 있는 경우 호출이 실패하는지 여부
- 그룹화 및 세그먼트화가 허용되는지 여부
- 새 메시지 ID와 상관 ID의 생성
- 메시지 및 세그먼트를 큐에 넣는 순서
- 로컬 큐 이름을 해석할지 여부

Options 필드를 기본값(MQPMO_NONE)으로 설정된 상태로 두면, 사용자가 넣는 메시지에 해당 메시지와 연관된 컨텍스트 정보가 있게 됩니다.

통화가 동기화 지점과 함께 작동하는 방식은 플랫폼에 의해 결정됩니다. 동기화 지점 제어 기본값은 단일 z/OS 플랫폼의 경우 예, 멀티 플랫폼의 경우 아니오입니다.

컨텍스트

컨텍스트 정보를 복사할 원본 큐 핸들의 이름을 명시합니다(*Options* 필드에서 요청된 경우).

메시지 컨텍스트에 대한 소개는 51 페이지의 『메시지 컨텍스트』의 내용을 참조하십시오. MQPMO 구조를 사용하여 메시지의 컨텍스트 정보를 제어하는 방법에 대한 정보는 778 페이지의 『메시지 컨텍스트 정보 제어』의 내용을 참조하십시오.

ResolvedQName

메시지를 수신하기 위해 열린 큐의 이름(모든 알리어스 이름 해석 후의 이름)을 포함합니다. 이 필드는 출력 필드입니다.

ResolvedQMgrName

*ResolvedQName*에서 큐를 소유하는 큐 관리자의 이름(모든 알리어스 이름 해석 후의 이름)을 포함합니다. 이 필드는 출력 필드입니다.

MQPMO는 분배 목록에 필요한 필드를 수용할 수도 있습니다(781 페이지의 『분배 목록』 참조). 이 기능을 사용하려는 경우 MQPMO 구조의 버전 2가 사용됩니다. 여기에는 다음 필드가 포함됩니다.

RecsPresent

이 필드는 분배 목록에 있는 큐 수, 즉 제공되는 메시지 넣기 레코드(MQPMR) 및 상응하는 응답 레코드(MQRR) 수가 포함됩니다.

입력하는 값은 MQOPEN에서 제공된 오브젝트 레코드 수와 동일할 수 있습니다. 그러나 값이 MQOPEN 호출에서 제공된 오브젝트 레코드 수보다 작거나 메시지 넣기 레코드를 제공하지 않는 경우, 정의되지 않은 큐의 값은 메시지 디스크립터에서 제공된 기본값을 가져옵니다. 또한 값이 제공된 오브젝트 레코드 수보다 크면 초과하는 메시지 넣기 레코드는 무시됩니다.

다음 중 하나를 수행하는 것이 좋습니다.

- 각 목적지에서 보고서 또는 회신을 수신하려는 경우, MQOR 구조에 표시된 것과 동일한 값을 입력하고 *MsgId* 필드를 포함하는 MQPMR을 사용하십시오. 이러한 *MsgId* 필드를 0으로 초기화하거나 MQPMO_NEW_MSG_ID를 지정하십시오.

메시지를 큐에 넣은 경우, 큐 관리자가 작성한 *MsgId* 값은 MQPMR에서 사용할 수 있습니다. 이러한 값을 사용하여 각 보고서 또는 회신과 연관된 목적지를 식별할 수 있습니다.

- 보고서 또는 회신을 수신하지 않으려는 경우 다음 중 하나를 선택하십시오.
 1. 실패한 목적지를 즉시 식별하려는 경우, MQOR 구조에 표시된 것과 동일한 값을 *RecsPresent* 필드에 입력하고 해당 목적지를 식별하기 위해 MQRR을 제공할 수 있습니다. MQPMR은 지정하지 마십시오.
 2. 실패한 목적지를 식별하지 않으려는 경우, *RecsPresent* 필드에 0을 입력하고 MQPMR 또는 MQRR을 제공하지 마십시오.

참고: MQPUT1을 사용 중인 경우 응답 레코드 포인터 및 응답 레코드 오프셋 수는 0이어야 합니다.

메시지 넣기 레코드(MQPMR) 및 응답 레코드(MQRR)에 대한 전체 설명은 MQPMR 및 MQRR의 내용을 참조하십시오.

PutMsgRecFields

각 메시지 넣기 레코드(MQPMR)에 나타나는 필드를 표시합니다. 이러한 필드의 목록은 785 페이지의 『MQPMR 구조 사용』의 내용을 참조하십시오.

PutMsgRecOffset 및 PutMsgRecPtr

포인터(일반적으로 C에서) 및 오프셋(일반적으로 COBOL에서)은 메시지 넣기 레코드를 주소 지정하는 데 사용됩니다(MQPMR 구조에 대한 개요는 785 페이지의 『MQPMR 구조 사용』 참조).

첫 번째 메시지 넣기 레코드에 대한 포인터를 지정하려면 *PutMsgRecPtr* 필드를 사용하거나, 첫 번째 메시지 넣기 레코드의 오프셋을 지정하려면 *PutMsgRecOffset* 필드를 사용하십시오. 이는 MQPMO 시작으로 부터의 오프셋입니다. *PutMsgRecFields* 필드에 따라, *PutMsgRecOffset* 또는 *PutMsgRecPtr* 중 하나에 널이 아닌 값을 입력하십시오.

ResponseRecOffset 및 ResponseRecPtr

또한 포인터 및 오프셋을 사용하여 응답 레코드를 주소 지정할 수 있습니다(응답 레코드에 대한 자세한 정보는 784 페이지의 『MQRR 구조 사용』 참조).

첫 번째 응답 레코드에 대한 포인터를 지정하려면 *ResponseRecPtr* 필드를 사용하거나, 첫 번째 응답 레코드의 오프셋을 지정하려면 *ResponseRecOffset* 필드를 사용하십시오. 이는 MQPMO 구조 시작으로 부터의 오프셋입니다. *ResponseRecOffset* 또는 *ResponseRecPtr* 중 하나에 널이 아닌 값을 입력하십시오.

참고: MQPUT1을 사용하여 메시지를 분배 목록에 넣는 중인 경우, *ResponseRecPtr*은 널 또는 0이어야 하며 *ResponseRecOffset*은 0이어야 합니다.

MQPMO 구조의 버전 3에는 추가적으로 다음 필드가 포함되어 있습니다.

OriginalMsgHandle

이 필드를 사용할 수 있는지 여부는 *Action* 필드의 값에 달려 있습니다. 연관된 메시지 특성이 있는 새 메시지를 넣는 중인 경우, 이 필드를 이전에 작성하고 특성을 설정한 메시지 핸들로 설정하십시오. 이전에 검색한 메시지의 응답으로 보고서를 전달, 회신 또는 생성 중인 경우 이 필드에는 해당 메시지의 메시지 핸들이 포함됩니다.

NewMsgHandle

*NewMsgHandle*을 지정하는 경우, 핸들과 연관된 모든 특성이 *OriginalMsgHandle*과 연관된 특성을 대체합니다. 자세한 정보는 [조치\(MQLONG\)](#)의 내용을 참조하십시오.

Action

넣기가 수행되는 유형을 지정하려면 이 필드를 사용하십시오. 가능한 값과 해당 의미는 다음과 같습니다.

MQACTP_NEW

다른 것과 관련되지 않은 새 메시지입니다.

MQACTP_FORWARD

이 메시지는 이전에 검색되고 현재 전달되는 중입니다.

MQACTP_REPLY

이 메시지는 이전에 검색된 메시지의 회신입니다.

MQACTP_REPORT

이 메시지는 이전에 검색된 메시지의 결과로 생성된 보고서입니다.

자세한 정보는 [조치\(MQLONG\)](#)의 내용을 참조하십시오.

PubLevel

이 메시지가 발행물인 경우 메시지를 수신하는 구독을 판별하기 위해 이 필드를 설정할 수 있습니다. 이 값보다 작거나 같은 *SubLevel*이 있는 구독만이 이 발행물을 수신하게 됩니다. 기본값 9는 가장 높은 레벨이며 모든 *SubLevel*이 있는 구독이 이 발행물을 수신함을 의미합니다.

메시지의 데이터

데이터를 포함하는 버퍼의 주소를 MQPUT 호출의 **Buffer** 매개변수에 제공하십시오. 메시지의 데이터에는 어떤 것이라도 포함시킬 수 있습니다. 그러나 메시지의 데이터 양은 데이터를 처리하는 애플리케이션의 성능에 영향을 줍니다.

데이터의 최대 크기는 다음을 기준으로 판별됩니다.

- 큐 관리자의 **MaxMsgLength** 속성
- 메시지를 넣는 중인 큐의 **MaxMsgLength** 속성
- IBM MQ에서 추가된 메시지 헤더의 크기(데드 레터 헤더 MQDLH 및 분배 목록 헤더 MQDH 포함)

큐 관리자의 **MaxMsgLength** 속성은 큐 관리자가 처리할 수 있는 메시지의 크기를 보유합니다. V6 이상의 모든 IBM MQ 제품의 경우 기본값 100MB가 사용됩니다.

이 속성의 값을 판별하려면 큐 관리자 오브젝트에서 MQINQ 호출을 사용하십시오. 대용량 메시지를 위해 이 값을 변경할 수 있습니다.

큐의 **MaxMsgLength** 속성은 큐에 넣을 수 있는 메시지의 최대 크기를 판별합니다. 이 속성의 값보다 큰 크기의 메시지를 넣으려고 하면 MQPUT 호출이 실패합니다. 메시지를 리모트 큐에 넣는 중인 경우, 성공적으로 넣을 수 있는 메시지의 최대 크기는 리모트 큐, 메시지가 라우트를 따라 목적지로 놓여지는 중간 전송 큐, 사용되는 채널의 **MaxMsgLength** 속성에 의해 판별됩니다.

MQPUT 조작의 경우, 메시지의 크기는 큐 및 큐 관리자의 **MaxMsgLength** 속성보다 작거나 같아야 합니다. 이러한 속성의 값은 독립적이지만, 큐의 **MaxMsgLength**를 큐 관리자의 값보다 작거나 같게 설정하는 것이 좋습니다.

IBM MQ는 다음 상황에서 헤더 정보를 메시지에 추가합니다.

- 메시지를 리모트 큐에 넣는 경우, IBM MQ가 전송 헤더 구조(MQXQH)를 메시지에 추가합니다. 이 구조에는 목적지 큐의 이름과 고유의 큐 관리자가 포함됩니다.
- IBM MQ가 메시지를 리모트 큐로 전달할 수 없는 경우, 메시지를 데드 레터(전달되지 않은 메시지) 큐에 넣으려고 시도합니다. MQDLH 구조를 메시지에 추가합니다. 이 구조에는 목적지 큐의 이름과 메시지가 데드-레터 큐에 놓여진 이유가 포함됩니다.
- 메시지를 다중 목적지 큐로 송신하려는 경우, IBM MQ가 MQDH 헤더를 메시지에 추가합니다. 이 헤더는 메시지에 표시되고, 분배 목록에 속하고, 전송 큐에 있는 데이터를 설명합니다. 최대 메시지 크기에 대해 최적의 값을 선택할 때 이를 고려하십시오.
- 메시지가 그룹의 메시지 또는 세그먼트인 경우, IBM MQ가 MQMDE를 추가할 수 있습니다.

이러한 구조는 MQDH 및 MQMDE에 설명되어 있습니다.

메시지가 이러한 큐에 허용되는 최대 크기인 경우, 이러한 헤더를 추가하는 것은 메시지가 현재 너무 크기 때문에 넣기 조작이 실패함을 의미합니다. 넣기 조작의 실패 가능성을 줄이려면 다음을 수행하십시오.

- 메시지의 크기를 전송 및 데드-레터 큐의 **MaxMsgLength** 속성보다 작게 만드십시오. 최소한 MQ_MSG_HEADER_LENGTH 상수의 값(대형 분배 목록의 경우 그 이상의 값)을 허용하십시오.
- 데드-레터 큐의 **MaxMsgLength** 속성이 데드-레터 큐를 소유하는 큐 관리자의 **MaxMsgLength**와 동일하게 설정되었는지 확인하십시오.

큐 관리자의 속성과 메시지 큐잉 상수는 [큐 관리자에 대한 속성](#)에 설명되어 있습니다.

 분산 큐잉 환경에서 전달되지 않은 메시지가 처리되는 방식에 대한 정보는 [전달되지 않은/처리되지 않은 메시지](#)를 참조하십시오.

메시지 넣기: 메시지 핸들 사용

MQPMO 구조에서는 *OriginalMsgHandle* 및 *NewMsgHandle*의 두 가지 메시지 핸들을 사용할 수 있습니다. 이러한 메시지 핸들 간의 관계는 MQPMO Action 필드의 값에 의해 정의됩니다.

전체 세부사항은 [조치\(MQLONG\)](#)의 내용을 참조하십시오. 메시지 핸들은 메시지를 넣기 위해 반드시 필요하지는 않습니다. 메시지 핸들의 목적은 특성을 메시지와 연관시키는 것이므로 메시지 특성을 사용 중인 경우에만 필요합니다.

리모트 큐에 메시지 넣기

로컬 큐 대신에 리모트 큐(즉, 애플리케이션이 연결되는 큐가 아닌 큐 관리자가 소유하는 큐)에 메시지를 넣으려는 경우, 추가적으로 고려할 유일한 사항은 큐를 열 때 큐의 이름을 지정하는 방법입니다. 이 프로그램은 771 페이지의 [『리모트 큐 열기』](#)에서 설명합니다. 로컬 큐에 MQPUT 또는 MQPUT1 호출을 사용하는 방법에는 차이가 없습니다.

리모트 및 전송 큐 사용에 대한 자세한 정보는 [IBM MQ 분산 큐잉 기법](#)의 내용을 참조하십시오.

메시지의 특성 설정

설정하려는 각 특성에 대해 MQSETMP를 호출하십시오. 메시지를 넣을 때 MQPMO 구조의 메시지 핸들 및 조치 필드를 설정하십시오.

특성을 메시지와 연관시키려면 메시지에 메시지 핸들이 있어야 합니다. MQCRTMH 함수 호출을 사용하여 메시지 핸들을 작성하십시오. 설정하려는 각 특성에 대해 이 메시지 핸들을 지정하여 MQSETMP를 호출하십시오. MQSETMP 사용 방법을 보여주기 위해 샘플 프로그램 amqsstma.c가 제공됩니다.

새 메시지인 경우, MQPUT 또는 MQPUT1을 사용하여 메시지를 큐에 넣을 때 MQPMO의 OriginalMsgHandle 필드를 이 메시지 핸들의 값으로 설정하고 MQPMO Action 필드를 MQACTP_NEW(기본값)로 설정하십시오.

이전에 검색한 메시지도 지금 전달 또는 회신 중이거나 응답으로 보고서를 송신 중인 경우, MQPMO의 OriginalMsgHandle 필드에 원본 메시지 핸들을 입력하고 NewMsgHandle 필드에 새 메시지 핸들을 입력하십시오. Action 필드를 MQACTP_FORWARD, MQACTP_REPLY 또는 MQACTP_REPORT로 적절하게 설정하십시오.

이전에 검색한 메시지의 MQRFH2 헤더에 특성이 있는 경우, MQBUFMH 호출을 사용하여 해당 특성을 메시지 핸들 특성으로 변환할 수 있습니다.

메시지 특성을 처리할 수 없는, IBM WebSphere MQ 7.0 이전 레벨의 큐 관리자에서 메시지를 큐에 넣는 중인 경우 채널 정의에 PropertyControl 매개변수를 설정하여 특성이 처리될 방법을 지정할 수 있습니다.

메시지 컨텍스트 정보 제어

MQPUT 또는 MQPUT1 호출을 사용하여 메시지를 큐에 넣을 때 큐 관리자가 일부 기본 컨텍스트 정보를 메시지 디스크립터에 추가하도록 지정할 수 있습니다. 적절한 권한 레벨이 있는 애플리케이션이 추가 컨텍스트 정보를 추가할 수 있습니다. MQPMO 구조의 옵션 필드를 사용하여 컨텍스트 정보를 제어할 수 있습니다.

메시지 컨텍스트 정보를 통해 메시지를 검색하는 애플리케이션에서 메시지의 진원지를 찾아낼 수 있습니다. 모든 컨텍스트 정보는 메시지 디스크립터의 컨텍스트 필드에 저장됩니다. 정보 유형은 ID, 원본 및 사용자 컨텍스트 정보로 나뉘어집니다.

컨텍스트 정보를 제어하려면 MQPMO 구조의 *Options* 필드를 사용하십시오.

컨텍스트 정보에 어떤 옵션도 지정하지 않는 경우, 큐 관리자는 메시지 디스크립터에 이미 존재할 수 있는 컨텍스트 정보를 메시지에 대해 생성된 컨텍스트 정보로 덮어씁니다. 이것은 MQPMO_DEFAULT_CONTEXT 옵션을 지정하는 것과 동일합니다. 새 메시지를 작성할 때(예를 들어, 조회 화면에서 사용자 입력을 처리할 때) 이 기본 컨텍스트 정보를 원할 수 있습니다.

메시지와 연관된 컨텍스트 정보를 원하지 않는 경우 MQPMO_NO_CONTEXT 옵션을 사용하십시오. 컨텍스트가 없는 메시지를 넣을 때 IBM MQ에 의해 작성된 모든 권한 검사는 공백 사용자 ID를 사용하여 수행됩니다. 공백 사용자 ID는 IBM MQ 자원에 대해 명시적 권한이 지정될 수 없지만 특별 그룹 'nobody'의 멤버로 처리됩니다. 특별 그룹 nobody에 대한 자세한 정보는 [설치 가능 서비스 인터페이스 참조 정보](#)의 내용을 참조하십시오.

MQOPEN과 다음 절에 표시된 MQOO_ 옵션 및 MQPMO_ 옵션을 사용하는 MQPUT을 순서대로 사용하여 컨텍스트 설정을 수행할 수 있습니다. MQPUT1만을 사용하여 컨텍스트 설정을 수행할 수도 있으며, 이 경우 아래 절에 표시된 MQPMO_ 옵션을 선택해야 합니다.

이 주제의 다음 절에서는 ID 컨텍스트, 사용자 컨텍스트 및 모든 컨텍스트의 사용에 대해 설명합니다.

- [779 페이지의 『ID 컨텍스트 전달』](#)
- [779 페이지의 『사용자 컨텍스트 전달』](#)
- [780 페이지의 『모든 컨텍스트 전달』](#)
- [780 페이지의 『ID 컨텍스트 설정』](#)
- [780 페이지의 『사용자 컨텍스트 설정』](#)
- [780 페이지의 『모든 컨텍스트 설정』](#)

ID 컨텍스트 전달

일반적으로, 프로그램은 데이터가 최종 목적지에 도달할 때까지 애플리케이션 전체에서 메시지에서 메시지로 ID 컨텍스트 정보를 전달해야 합니다.

프로그램은 데이터를 변경할 때마다 원본 컨텍스트 정보를 변경해야 합니다. 그러나 컨텍스트 정보를 변경하거나 설정하려는 애플리케이션은 적절한 권한 레벨을 가지고 있어야 합니다. 큐 관리자는 애플리케이션이 큐를 열 때 이 권한을 검사합니다. 애플리케이션은 MQOPEN 호출에 적절한 컨텍스트 옵션을 사용하기 위한 권한을 가지고 있어야 합니다.

애플리케이션이 메시지를 가져와서 메시지의 데이터를 처리한 후 변경된 데이터를 다른 메시지에 넣은 경우(다른 애플리케이션에 의한 처리를 위해) 애플리케이션은 원래 메시지의 ID 컨텍스트 정보를 새 메시지로 전달해야 합니다. 큐 관리자가 원본 컨텍스트 정보를 작성할 수 있도록 허용할 수 있습니다.

원래 메시지의 컨텍스트 정보를 저장하려면 메시지를 가져오기 위한 큐를 열 때 MQOO_SAVE_ALL_CONTEXT 옵션을 사용하십시오. MQOPEN 호출에서 다른 옵션을 사용할 때도 이 옵션은 필요합니다. 그러나 메시지를 찾아 보기만 하는 경우 컨텍스트 정보를 저장할 수 없음을 참고하십시오.

두 번째 메시지를 작성할 때 다음을 수행하십시오.

- (MQOO_OUTPUT 옵션 이외에) MQOO_PASS_IDENTITY_CONTEXT 옵션을 사용하여 큐를 여십시오.
- 메시지 넣기 옵션 구조의 *Context* 필드에 컨텍스트 정보를 가져와서 저장한 큐의 핸들을 제공하십시오.
- 메시지 넣기 옵션 구조의 *Options* 필드에 MQPMO_PASS_IDENTITY_CONTEXT 옵션을 지정하십시오.

사용자 컨텍스트 전달

사용자 컨텍스트만 전달하도록 선택할 수 없습니다. 메시지를 넣을 때 사용자 컨텍스트를 전달하려면 MQPMO_PASS_ALL_CONTEXT를 지정하십시오. 사용자 컨텍스트의 특성은 원본 컨텍스트와 동일한 방법으로 전달됩니다.

MQPUT 또는 MQPUT1이 수행되고 컨텍스트가 전달되고 있는 경우, 사용자 컨텍스트에 있는 모든 특성이 검색된 메시지에서 넣어진 메시지로 전달됩니다. 넣기 애플리케이션이 대체한 모든 사용자 컨텍스트 특성은 원래 값으로 넣어집니다. 넣기 애플리케이션이 삭제한 모든 사용자 컨텍스트 특성은 넣어진 메시지에 복원됩니다. 넣기 애플리케이션이 메시지에 추가한 모든 사용자 컨텍스트 특성은 보유됩니다.

모든 컨텍스트 전달

애플리케이션이 메시지를 가져와서 메시지 데이터(변경되지 않은 데이터)를 다른 메시지에 넣은 경우, 애플리케이션은 원래 메시지의 모든 컨텍스트 정보(ID, 원본 및 사용자)를 새 메시지로 전달해야 합니다. 이 작업을 수행할 수 있는 애플리케이션의 예는 메시지 이동 프로그램이며, 한 큐에서 다른 큐로 메시지를 이동시킵니다.

MQOPEN 옵션 MQOO_PASS_ALL_CONTEXT 및 메시지 넣기 옵션 MQPMO_PASS_ALL_CONTEXT를 사용하는 경우를 제외하고는 ID 컨텍스트 전달의 경우와 동일한 프로시저를 따르십시오.

ID 컨텍스트 설정

메시지에 ID 컨텍스트 정보를 설정하려는 경우 다음을 수행하십시오.

- MQOO_SET_IDENTITY_CONTEXT 옵션을 사용하여 큐를 여십시오.
- MQPMO_SET_IDENTITY_CONTEXT 옵션을 지정하여 큐에 메시지를 넣으십시오. 메시지 디스크립터에서, 필요한 모든 ID 컨텍스트 정보를 지정하십시오.

참고: QOO_SET_IDENTITY_CONTEXT 및 MQPMO_SET_IDENTITY_CONTEXT 옵션을 사용하여 일부(전부는 아님) ID 컨텍스트 필드를 설정할 때 큐 관리자가 나머지 필드를 설정하지 않는다는 점에 유의하십시오.

임의 메시지 컨텍스트 옵션을 수정하려면 호출을 발행할 수 있는 해당 권한이 있어야 합니다. 예를 들어, MQOO_SET_IDENTITY_CONTEXT 또는 MQPMO_SET_IDENTITY_CONTEXT를 사용하려면 +setid 권한이 있어야 합니다.

사용자 컨텍스트 설정

사용자 컨텍스트의 특성을 설정하려면 MQSETMP 호출을 작성할 때 메시지 특성 디스크립터(MQPD)의 Context 필드를 MQPD_USER_CONTEXT로 설정하십시오.

사용자 컨텍스트의 특성을 설정하기 위해 특별한 권한은 필요하지 않습니다. 사용자 컨텍스트에는 MQOO_SET_* 또는 MQPMO_SET_* 컨텍스트 옵션이 없습니다.

모든 컨텍스트 설정

메시지의 ID 및 원본 컨텍스트 정보를 모두 설정하려는 경우 다음을 수행하십시오.

1. MQOO_SET_ALL_CONTEXT 옵션을 사용하여 큐를 여십시오.
2. MQPMO_SET_ALL_CONTEXT 옵션을 지정하여 큐에 메시지를 넣으십시오. 메시지 디스크립터에서, 필요한 모든 ID 및 원본 컨텍스트 정보를 지정하십시오.

각 컨텍스트 설정 유형에 대해 적절한 권한이 필요합니다.

관련 개념

[51 페이지의 『메시지 컨텍스트』](#)

메시지 컨텍스트 정보를 통해 메시지를 검색하는 애플리케이션에서 메시지의 진원지를 찾아낼 수 있습니다.

관련 정보

[770 페이지의 『메시지 컨텍스트와 관련된 MQOPEN 옵션』](#)

MQPUT1 호출을 사용하여 큐에 하나의 메시지 넣기

큐에 단일 메시지를 넣은 후 즉시 큐를 닫으려는 경우 MQPUT1 호출을 사용하십시오. 예를 들어, 서버 애플리케이션은 서로 다른 각 큐에 응답을 송신하는 중에 MQPUT1 호출을 사용할 것입니다.

MQPUT1은 기능적으로 MQOPEN, MQPUT, MQCLOSE를 순서대로 호출하는 것과 동등합니다. MQPUT 및 MQPUT1 호출의 구문에서 유일한 차이점은 MQPUT의 경우 오브젝트 핸들을 지정하는 반면, MQPUT1의 경우 MQOPEN에 지정된 대로 오브젝트 디스크립터 구조(MQOD)를 지정하는 것입니다(765 페이지의 『오브젝트 식별(MQOD 구조)』 참조). 이렇게 해야 하는 이유는 MQPUT1 호출에는 이 호출이 열어야 하는 큐에 대한 정보를 제공해야 하지만, MQPUT을 호출할 때는 큐가 이미 열려 있어야 하기 때문입니다.

MQPUT1 호출에 대한 입력으로 다음을 제공해야 합니다.

- 연결 핸들.
- 열리는 오브젝트의 설명. 오브젝트 디스크립터 구조(MQOD)의 양식으로 되어 있습니다.

- 큐에 넣으려는 메시지의 설명. 메시지 디스크립터 구조(MQMD)의 양식으로 되어 있습니다.
- 메시지 넣기 옵션 구조(MQPMO) 양식의 제어 정보.
- 메시지에 포함된 데이터의 길이(MQLONG)
- 메시지 데이터의 주소.

MQPUT1의 출력은 다음과 같습니다.

- 완료 코드
- 이유 코드

호출이 성공적으로 완료되는 경우 옵션 구조 및 메시지 디스크립터 구조도 리턴됩니다. 호출은 사용자의 옵션 구조를 수정하여 메시지가 송신된 큐 관리자 및 큐의 이름을 표시합니다. 사용자가 넣고 있는 메시지의 ID에 대해 큐 관리자가 고유한 값을 생성하도록 요청하는 경우(MQMD 구조의 *MsgId* 필드에 2진 0을 지정하여), 호출은 이 구조를 리턴하기 전에 *MsgId* 필드에 해당 값을 삽입합니다.

참고: 모델 큐 이름으로 MQPUT1을 사용할 수 없습니다. 단, 모델 큐가 열려진 후에는 동적 큐로 MQPUT1을 발행할 수 있습니다.

MQPUT1에 대한 6개의 입력 매개변수는 다음과 같습니다.

Hconn

연결 핸들입니다. CICS 애플리케이션의 경우, 상수 MQHC_DEF_HCONN(0 값을 가짐)을 지정하거나 MQCONN 또는 MQCONNx 호출에서 리턴된 연결 핸들을 사용할 수 있습니다. 다른 프로그램의 경우에는 항상 MQCONN 또는 MQCONNx 호출에서 리턴된 연결 핸들을 사용하십시오.

ObjDesc

오브젝트 디스크립터 구조(MQOD)입니다.

ObjectName 및 *ObjectQMgrName* 필드에 메시지를 넣으려는 큐의 이름과 이 큐를 소유하는 큐 관리자의 이름을 제공하십시오.

MQPUT1 호출의 경우 모델 큐를 사용할 수 없으므로 *DynamicQName* 필드가 무시됩니다.

큐를 열기 위한 권한을 테스트하는 데 사용될 대체 사용자 ID를 지정하려는 경우 *AlternateUserId* 필드를 사용하십시오.

MsgDesc

메시지 디스크립터 구조(MQMD)입니다. MQPUT 호출과 마찬가지로, 큐에 넣고 있는 메시지를 정의하는 데 이 구조를 사용하십시오.

PutMsgOpts

메시지 넣기 옵션 구조(MQPMO)입니다. MQPUT 호출에 사용하십시오(774 페이지의 『MQPMO 구조를 사용하여 옵션 지정』 참조).

Options 필드를 0으로 설정하면, 큐 관리자는 큐에 액세스하기 위한 권한 테스트를 수행할 때 사용자 고유의 사용자 ID를 사용합니다. 또한 큐 관리자는 MQOD 구조의 *AlternateUserId* 필드에 제공된 대체 사용자 ID를 무시합니다.

BufferLength

메시지의 길이입니다.

Buffer

메시지의 텍스트가 포함된 버퍼 영역입니다.

클러스터를 사용하는 경우, MQPUT1은 MQOO_BIND_NOT_FIXED가 적용되고 있는 중에도 작동합니다. 애플리케이션은 MQOD 구조 대신 MQPMO 구조의 해석된 필드를 사용하여 메시지가 송신된 위치를 판별해야 합니다. 자세한 정보는 [큐 관리자 클러스터 구성](#)의 내용을 참조하십시오.

MQPUT1 호출에 대한 설명은 [MQPUT1](#)에 있습니다.

Multi 분배 목록

배포 목록을 사용하면 IBM MQ for Multiplatforms 단일 MQPUT 또는 MQPUT1 호출로 여러 대상에 메시지를 전송할 수 있습니다. 단일 MQOPEN 호출로 다중 큐를 열고, 단일 MQPUT 호출로 이러한 각 큐에 메시지를 넣을 수

있습니다. 이 프로세스에 사용된 MQI 구조의 일부 일반 정보는 분배 목록에 포함된 개별 목적지와 관련된 특정 정보로 대체될 수 있습니다.

주의: 분배 목록에서는 토픽 오브젝트를 가리키는 알리어스 큐 사용을 지원하지 않습니다. 알리어스 큐가 분배 목록의 토픽 오브젝트를 가리키는 경우 IBM MQ에서 MQRC_ALIAS_BASE_Q_TYPE_ERROR를 리턴합니다.

MQOPEN 호출이 발행될 때 오브젝트 디스크립터(MQOD)에서 일반 정보를 가져옵니다. *Version* 필드에 MQOD_VERSION_2를 지정하고 *RecsPresent* 필드에 0보다 큰 값을 지정하면, *Hobj*를 하나의 큐가 아닌 목록(하나 이상의 큐)의 핸들로 정의할 수 있습니다. 이 경우, 특정 정보가 오브젝트 레코드(MQOR)를 통해 제공되어 목적지의 세부사항(즉, *ObjectName* 및 *ObjectQMgrName*)을 제공합니다.

오브젝트 핸들(*Hobj*)은 MQPUT 호출로 전달되어 단일 큐가 아닌 목록 큐에 넣을 수 있습니다.

메시지가 큐(MQPUT)에 넣어지면 메시지 넣기 옵션 구조(MQPMO) 및 메시지 디스크립터(MQMD)에서 일반 정보를 가져옵니다. 특정 정보는 메시지 넣기 레코드(MQPMR) 양식으로 제공됩니다.

응답 레코드(MQRR)는 각 목적지 큐에 해당하는 완료 코드 및 이유 코드를 수신할 수 있습니다.

782 페이지의 그림 86에서는 분배 목록이 작동하는 방법을 표시합니다.

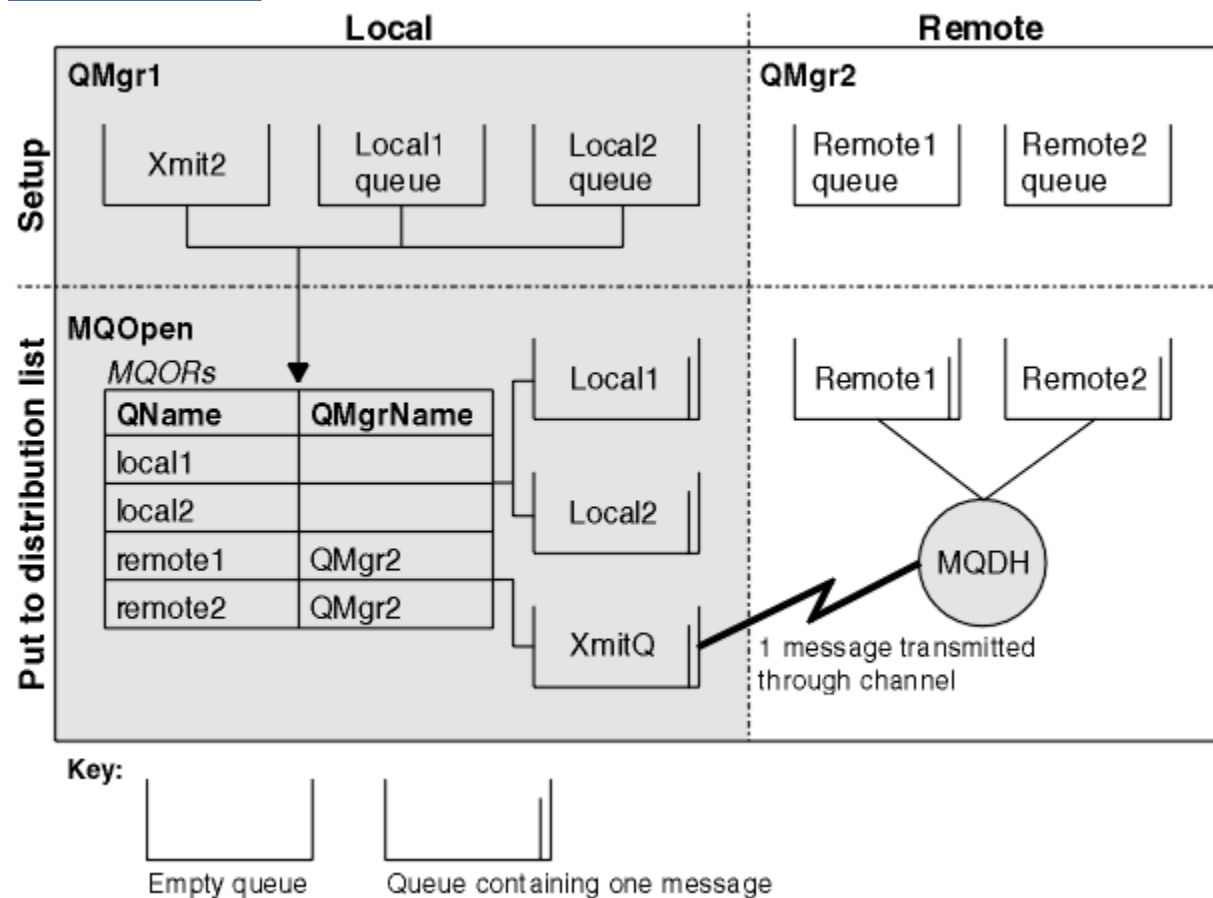


그림 86. 분배 목록의 작동 방법

분배 목록 열기

분배 목록을 열려면 MQOPEN 호출을 사용하고, 호출의 옵션을 사용하여 목록에 대해 수행하려는 작업을 지정하십시오.

MQOPEN의 입력으로 다음을 제공해야 합니다.

- 연결 핸들(설명은 772 페이지의 『큐에 메시지 넣기』 참조)
- 오브젝트 디스크립터 구조(MQOD)의 일반 정보
- 오브젝트 레코드 구조(MQOR)를 사용하여 열려는 각 큐의 이름

MQOPEN의 출력은 다음과 같습니다.

- 분배 목록에 대한 액세스 권한을 나타내는 오브젝트 핸들
- 일반 완료 코드
- 일반 이유 코드
- 응답 레코드(선택사항), 각 목적지에 대한 완료 코드 및 이유 포함

MQOD 구조 사용

열려는 큐를 식별하려면 MQOD 구조를 사용하십시오.

분배 목록을 정의하려면 *Version* 필드에 *MQOD_VERSION_2*를, *RecsPresent* 필드에 0보다 큰 값을, *ObjectType* 필드에 *MQOT_Q*를 지정해야 합니다. MQOD 구조의 모든 필드에 대한 설명은 *MQOD*의 내용을 참조하십시오.

MQOR 구조 사용

각 목적지에 대해 MQOR 구조를 제공하십시오.

구조에는 목적지 큐와 큐 관리자 이름이 포함되어 있습니다. MQOD의 *ObjectName* 및 *ObjectQMgrName* 필드는 분배 목록에 사용되지 않습니다. 하나 이상의 오브젝트 레코드가 있어야 합니다. *ObjectQMgrName*을 공백으로 두면 로컬 큐 관리자가 사용됩니다. 이러한 필드에 대한 자세한 정보는 *ObjectName* 및 *ObjectQMgrName*의 내용을 참조하십시오.

다음 두 가지 방법으로 목적지 큐를 지정할 수 있습니다.

- 오프셋 필드 *ObjectRecOffset* 사용.

이 경우, MQOD 구조를 포함하는 고유의 구조를 MQOR 레코드의 배열 뒤에 (필요한 모든 배열 요소와 함께) 선언하고, *ObjectRecOffset*을 MQOD 시작부터 배열에 있는 첫 번째 요소의 오프셋으로 설정해야 합니다. 이 오프셋이 올바른지 확인하십시오.

애플리케이션이 실행되는 모든 환경에서 내장 기능을 사용할 수 있는 경우, 프로그래밍 언어에서 제공하는 내장 기능을 사용하는 것이 좋습니다. 다음 코드는 COBOL 프로그래밍 언어에 대한 이 기법을 보여줍니다.

```
01 MY-OPEN-DATA.  
02 MY-MQOD.  
   COPY CMQODV.  
02 MY-MQOR-TABLE OCCURS 100 TIMES.  
   COPY CMQORV.  
MOVE LENGTH OF MY-MQOD TO MQOD-OBJECTRECOFFSET.
```

또는 관련된 모든 환경에서 프로그래밍 언어가 필수 내장 기능을 지원하지 않는 경우, 상수 *MQOD_CURRENT_LENGTH*를 사용하십시오. 다음 코드는 이 기법을 보여줍니다.

```
01 MY-MQ-CONSTANTS.  
   COPY CMQV.  
01 MY-OPEN-DATA.  
02 MY-MQOD.  
   COPY CMQODV.  
02 MY-MQOR-TABLE OCCURS 100 TIMES.  
   COPY CMQORV.  
MOVE MQOD-CURRENT-LENGTH TO MQOD-OBJECTRECOFFSET.
```

그러나 이 기법은 MQOD 구조와 MQOR 레코드의 배열이 연속적인 경우에만 작동합니다. 컴파일러가 MQOD 및 MQOR 어레이 사이에 건너뛰기 바이트를 삽입하는 경우, *ObjectRecOffset*에 저장된 값에 건너뛰기 바이트를 추가해야 합니다.

포인터 데이터 유형을 지원하지 않거나 서로 다른 환경으로 이동할 수 없는 방식으로 포인터 데이터 유형을 구현하는 프로그래밍 언어(예: COBOL 프로그래밍 언어)의 경우 *ObjectRecOffset*을 사용하는 것이 좋습니다.

- 포인터 필드 *ObjectRecPtr* 사용.

이 경우, 애플리케이션은 MQOD 구조와 별도로 MQOR 구조의 배열을 선언하고 *ObjectRecPtr*을 배열의 주소로 설정할 수 있습니다. 다음 코드는 C 프로그래밍 언어에 대한 이 기법을 보여줍니다.

```
MQOD MyMqod;
MQOR MyMqor[100];
MyMqod.ObjectRecPtr = MyMqor;
```

서로 다른 환경으로 이동할 수 있는 방식으로 포인터 데이터 유형을 지원하는 프로그래밍 언어(예: C 프로그래밍 언어)의 경우 *ObjectRecPtr*을 사용하는 것이 좋습니다.

어떤 기법을 선택하든지, *ObjectRecOffset* 및 *ObjectRecPtr* 중 하나를 사용해야 합니다. 둘 다 0이거나 0이 아닌 경우 호출이 이유 코드 MQRC_OBJECT_RECORDS_ERROR로 실패합니다.

MQRR 구조 사용

이러한 구조는 목적지에 특정적으로 사용하며, 각 응답 레코드에는 분배 목록의 각 큐에 대한 *CompCode* 및 *Reason* 필드가 포함되어 있습니다. 문제가 있는 위치를 구별하려면 이 구조를 사용해야 합니다.

예를 들어, 이유 코드 MQRC_MULTIPLE_REASONS가 수신되고 분배 목록에 5개의 목적지 큐가 있는 경우, 이 구조를 사용하지 않으면 어떤 큐에 문제가 있는지 알아낼 수 없습니다. 그러나 각 목적지에 대한 완료 코드 및 이유 코드가 있는 경우 오류를 훨씬 쉽게 찾을 수 있습니다.

MQRR 구조에 대한 자세한 정보는 [MQRR](#)의 내용을 참조하십시오.

784 페이지의 [그림 87](#)에서는 C로 분배 목록을 열 수 있는 방법을 표시합니다.

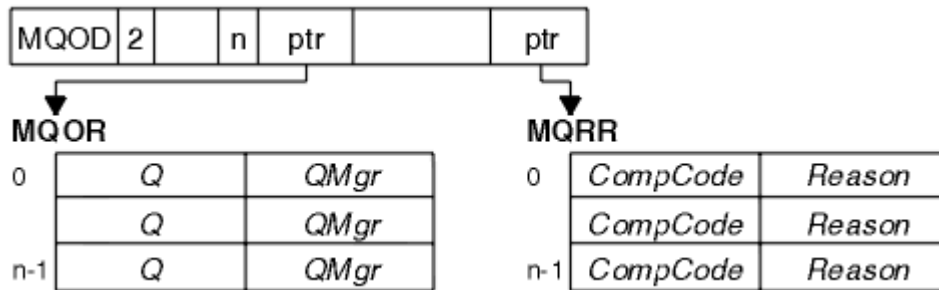


그림 87. C로 분배 목록 열기

784 페이지의 [그림 88](#)는 COBOL로 분배 목록을 열 수 있는 방법을 표시합니다.

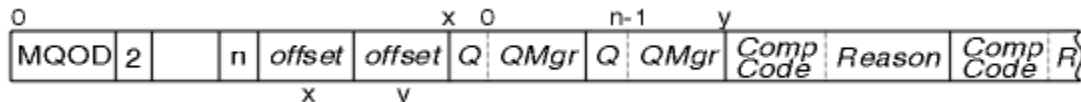


그림 88. COBOL로 분배 목록 열기

MQOPEN 옵션 사용

분배 리스트를 열 때 다음 옵션을 지정할 수 있습니다.

- MQOO_OUTPUT
- MQOO_FAIL_IF_QUIESCING(선택사항)
- MQOO_ALTERNATE_USER_AUTHORITY(선택사항)
- MQOO_*_CONTEXT(선택사항)

이러한 옵션에 대한 설명은 [763 페이지의 『오브젝트 열기 및 닫기』](#)의 내용을 참조하십시오.

분배 목록에 메시지 넣기

MQPUT 또는 MQPUT1을 사용하여 분배 목록에 메시지를 넣을 수 있습니다.

입력으로 다음을 제공해야 합니다.

- 연결 핸들(설명은 [772 페이지의 『큐에 메시지 넣기』](#) 참조)

- 오브젝트 핸들. MQOPEN을 사용하여 분배 목록을 열면, *Hobj*는 해당 목록에만 넣도록 허용합니다.
- 메시지 디스크립터 구조(MQMD). 이 구조에 대한 설명은 [MQMD](#)를 참조하십시오.
- 메시지 넣기 옵션 구조(MQPMO) 양식의 제어 정보. MQPMO 구조의 필드를 완료하는 방법에 대한 정보는 [774 페이지의 『MQPMO 구조를 사용하여 옵션 지정』](#)의 내용을 참조하십시오.
- 메시지 넣기 레코드(MQPMR) 양식의 제어 정보
- 메시지에 포함된 데이터의 길이(MQLONG)
- 메시지 데이터 자체

출력은 다음과 같습니다.

- 완료 코드
- 이유 코드
- 응답 레코드(선택사항)

MQPMR 구조 사용

이 구조는 선택사항이며, 이미 MQMD에서 식별된 필드와 다르게 식별할 수 있도록 일부 필드에 대해 목적지 특정 정보를 제공합니다.

이러한 필드에 대한 설명은 [MQPMR](#)의 내용을 참조하십시오.

각 레코드의 콘텐츠는 MQPMO의 *PutMsgRecFields* 필드에 지정된 정보에 따라 다릅니다. 예를 들어, 분배 목록 사용을 보여주는 샘플 프로그램 [AMQSPTLO.C\(1000 페이지의 『분배 목록 샘플 프로그램』](#)의 설명 참조)에서 샘플은 MQPMR의 *MsgId* 및 *CorrelId*에 대한 값을 제공하도록 선택합니다. 샘플 프로그램의 이 섹션은 다음과 같습니다.

```
typedef struct
{
    MQBYTE24 MsgId;
    MQBYTE24 CorrelId;
} PutMsgRec;
...
/*****
MQLONG PutMsgRecFields=MQPMRF_MSG_ID | MQPMRF_CORREL_ID;
```

이는 분배 목록의 각 목적지에 대해 *MsgId* 및 *CorrelId*가 제공된 것을 암시합니다. 메시지 넣기 레코드는 배열로 제공됩니다.

[785 페이지의 그림 89](#)에서는 C로 된 분배 목록에 메시지를 넣을 수 있는 방법을 보여줍니다.

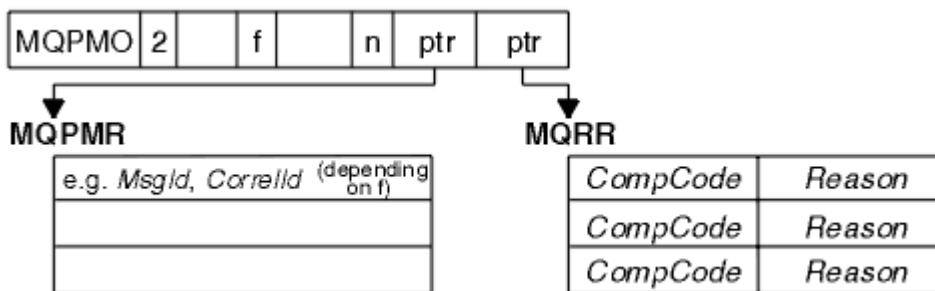


그림 89. C로 된 분배 목록에 메시지 넣기

[785 페이지의 그림 90](#)에서는 COBOL로 된 분배 목록에 메시지를 넣을 수 있는 방법을 보여줍니다.

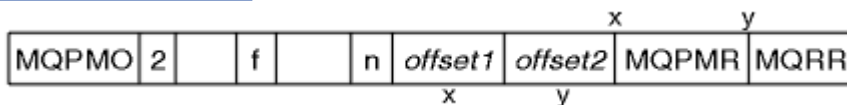


그림 90. COBOL로 된 분배 목록에 메시지 넣기

MQPUT1 사용

MQPUT1을 사용 중인 경우 다음 사항을 고려하십시오.

1. *ResponseRecOffset* 및 *ResponseRecPtr* 필드의 값은 널 또는 0이어야 합니다.
2. 필요한 경우, MQOD에서 응답 레코드의 주소를 지정해야 합니다.

Put 호출이 실패하는 몇 가지 사례

MQOPEN과 MQGET 호출을 발행하는 사이의 간격 동안 명령에 FORCE 옵션을 사용하여 큐의 특정 속성이 변경되면, MQGET 호출이 실패하고 MQRC_OBJECT_CHANGED 이유 코드를 리턴합니다.

큐 관리자는 오브젝트 핸들을 더 이상 유효하지 않는 것으로 표시합니다. 이는 MQPUT1 호출 처리 중에 변경사항이 작성되거나, 큐 이름이 확인되는 큐에 변경사항이 적용되는 경우에도 발생합니다. 이런 식으로 핸들에 영향을 주는 속성은 MQOPEN의 MQOPEN 호출에 대한 설명에 나열되어 있습니다. 호출이 MQRC_OBJECT_CHANGED 이유 코드를 리턴하면, 큐를 닫았다가 다시 연 후 메시지를 다시 넣어 보십시오.

메시지 넣기를 시도할 큐(또는 큐 이름이 확인되는 큐)에 대해 Put 조작이 금지되면, MQPUT 또는 MQPUT1 호출은 실패하고 MQRC_PUT_INHIBITED 이유 코드를 리턴합니다. 다른 프로그램이 큐의 속성을 주기적으로 변경하도록 애플리케이션이 설계된 경우에는 나중에 호출을 시도하면 메시지를 정상적으로 넣을 수 있습니다.

게다가 메시지를 넣으려는 큐가 가득 차면, MQPUT 또는 MQPUT1 호출은 실패하고 MQRC_Q_FULL을 리턴합니다.

동적 큐(임시 또는 영구)가 삭제된 경우, 이전에 확보된 오브젝트 핸들을 사용한 MQPUT 호출은 실패하고 MQRC_Q_DELETED 이유 코드를 리턴합니다. 이런 상황에서는 오브젝트 핸들이 더 이상 사용되지 않으므로 닫는 것이 좋습니다.

분배 목록의 경우, 단일 요청으로 다중 완료 코드와 이유 코드가 발생할 수 있습니다. 이러한 코드는 MQOPEN 및 MQPUT의 *CompCode* 및 *Reason* 출력 필드만 사용해서는 처리될 수 없습니다.

분배 목록을 사용하여 메시지를 여러 목적지에 넣을 때, 응답 레코드에는 각 목적지에 대한 특정 *CompCode* 및 *Reason*이 포함됩니다. MQCC_FAILED의 완료 코드를 수신하는 경우, 목적지 큐에 메시지를 넣지 못합니다. 완료 코드가 MQCC_WARNING이면 메시지가 하나 이상의 목적지 큐에 넣어집니다. MQRC_MULTIPLE_REASONS의 리턴 코드를 수신하는 경우, 각 목적지에 대한 이유 코드가 모두 동일하지는 않습니다. 따라서 오류를 야기한 큐와 각 오류의 원인을 판별할 수 있도록 MQRR 구조를 사용하는 것이 좋습니다.

큐에서 메시지 가져오기

이 정보를 사용하여 큐에서 메시지를 가져오는 방법에 대해 알아봅니다.

다음 두 가지 방법으로 큐에서 메시지를 가져올 수 있습니다.

1. 다른 프로그램에서 더 이상 볼 수 없도록 메시지를 큐에서 제거할 수 있습니다.
2. 메시지를 복사하고 원래 메시지는 큐에 남겨 둘 수 있습니다. 이 방법은 찾아보기라고 합니다. 메시지는 찾아본 후에 제거할 수 있습니다.

두 경우 모두 MQGET 호출을 사용하지만, 먼저 애플리케이션을 큐 관리자에 연결해야 하고 MQOPEN 호출을 사용해서 (입력, 찾아보기 또는 둘 모두를 위해) 큐를 열어야 합니다. 이러한 조작은 756 페이지의 『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』 및 763 페이지의 『오브젝트 열기 및 닫기』에서 설명합니다.

큐를 열었으면, MQGET 호출을 반복적으로 사용하여 동일한 큐의 메시지를 찾아보거나 제거할 수 있습니다. 큐에서 원하는 모든 메시지 가져오기가 완료되면 MQCLOSE를 호출하십시오.

큐에서 메시지를 가져오는 방법에 대해 자세히 알아보려면 다음 링크를 사용하십시오.

- 787 페이지의 『MQGET 호출을 사용하여 큐에서 메시지 가져오기』
- 791 페이지의 『큐에서 메시지를 검색하는 순서』
- 801 페이지의 『특정 메시지 가져오기』
- 802 페이지의 『비지속 메시지의 성능 개선』
-  806 페이지의 『색인 유형』
- 807 페이지의 『4MB 이상의 메시지 처리』

- [811 페이지의 『메시지 대기』](#)
- [▶ z/OS 812 페이지의 『신호보내기』](#)
- [▶ z/OS 813 페이지의 『백아웃 건너뛰기』](#)
- [816 페이지의 『애플리케이션 데이터 변환』](#)
- [817 페이지의 『큐의 메시지 찾아보기』](#)
- [822 페이지의 『MQGET 호출이 실패하는 몇 가지 사례』](#)

관련 개념

[744 페이지의 『MQI\(Message Queue Interface\) 개요』](#)

MQI(Message Queue Interface) 컴포넌트에 대해 학습합니다.

[756 페이지의 『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』](#)

IBM MQ 프로그래밍 서비스를 사용하려면 프로그램을 큐 관리자에 연결해야 합니다. 이 정보를 사용하여 큐 관리자에 연결하고 큐 관리자에서 연결을 끊는 방법에 대해 알아보십시오.

[763 페이지의 『오브젝트 열기 및 닫기』](#)

이 정보는 IBM MQ 오브젝트의 열기 및 닫기 조작에 대한 통찰력을 제공합니다.

[772 페이지의 『큐에 메시지 넣기』](#)

이 정보를 사용하여 메시지를 큐에 넣는 방법에 대해 알아보십시오.

[861 페이지의 『오브젝트 속성 조회 및 설정』](#)

속성은 IBM MQ 오브젝트의 특성을 정의하는 특성입니다.

[863 페이지의 『작업 단위 커밋 및 백아웃』](#)

이 정보는 작업 단위에서 발생한 복구 가능한 가져오기 및 넣기 조작을 커밋하고 백아웃하는 방법을 설명합니다.

[873 페이지의 『트리거를 사용한 IBM MQ 애플리케이션 시작』](#)

트리거와 트리거를 사용하여 IBM MQ 애플리케이션을 시작하는 방법에 대해 학습하십시오.

[891 페이지의 『MQI 및 클러스터에 대한 작업』](#)

클러스터와 관련된 호출 및 리턴 코드에 대한 특수 옵션이 있습니다.

[895 페이지의 『IBM MQ for z/OS의 애플리케이션 사용 및 작성』](#)

IBM MQ for z/OS 애플리케이션은 여러 다른 환경에서 실행되는 프로그램으로 구성할 수 있습니다. 이는 둘 이상의 환경에서 사용 가능한 기능을 활용할 수 있음을 의미합니다.

[71 페이지의 『IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션』](#)

이 정보는 IBM MQ을(를) 사용하여 IMS 애플리케이션을 작성하는 데 도움이 됩니다.

MQGET 호출을 사용하여 큐에서 메시지 가져오기

MQGET 호출은 열린 로컬 큐에서 메시지를 가져옵니다. 다른 시스템의 큐에서는 메시지를 가져올 수 없습니다.

MQGET 호출에 대한 입력으로 다음을 제공해야 합니다.

- 연결 핸들.
- 큐 핸들
- 큐에서 가져오려는 메시지에 대한 설명. 이는 메시지 디스크립터(MQMD) 구조 양식으로 되어 있습니다.
- 메시지 가져오기 옵션(MQGMO) 구조 양식의 제어 정보
- 메시지를 보유하기 위해 지정한 버퍼의 크기(MQLONG)
- 메시지를 넣을 스토리지의 주소

MQGET의 출력은 다음과 같습니다.

- 이유 코드
- 완료 코드
- 지정한 버퍼 영역의 메시지(호출이 완료된 경우)
- 옵션 구조(메시지가 검색된 큐의 이름을 표시하도록 수정됨)
- 메시지 디스크립터 구조(검색된 메시지에 대해 설명할 수 있도록 필드의 콘텐츠가 수정됨)

- 메시지의 길이(MQLONG)

MQGET에 MQGET 호출에 대한 설명이 있습니다.

다음 절에서는 MQGET 호출에 대한 입력으로 제공해야 할 정보에 대해 설명합니다.

- 788 페이지의 『연결 핸들 지정』
- 788 페이지의 『MQMD 구조와 MQGET 호출을 사용한 메시지 설명』
- 788 페이지의 『MQGMO 구조를 사용하여 MQGET 옵션 지정』
- 790 페이지의 『버퍼 영역의 크기 지정』

연결 핸들 지정

z/OS 응용 z/OS CICS 프로그램에서는 상수 MQHC_DEF_HCONN(값이 0인 상수)을 지정하거나, MQCONN 또는 MQCONNEX 호출이 반환한 연결 핸들을 사용할 수 있습니다. 기타 애플리케이션의 경우, MQCONN 또는 MQCONNEX 호출에 의해 리턴되는 연결 핸들을 항상 사용하십시오.

MQOPEN을 호출할 때 리턴되는 큐 핸들(Hobj)을 사용하십시오.

MQMD 구조와 MQGET 호출을 사용한 메시지 설명

큐에서 가져올 메시지를 식별하려면 메시지 디스크립터 구조(MQMD)를 사용하십시오.

이는 MQGET 호출의 입출력(I/O) 매개변수입니다. 18 페이지의 『IBM MQ 메시지』에는 MQMD가 설명하는 메시지 특성에 대한 소개가 있고, MQMD에는 구조 자체에 대한 설명이 있습니다.

큐에서 가져올 메시지를 알고 있는 경우 801 페이지의 『특정 메시지 가져오기』의 내용을 참조하십시오.

특정 메시지를 지정하지 않으면 MQGET은 큐에서 첫 번째 메시지를 검색합니다. 791 페이지의 『큐에서 메시지를 검색하는 순서』에서는 메시지의 우선순위, 큐의 **MsgDeliverySequence** 속성 및 MQGMO_LOGICAL_ORDER 옵션으로 큐에서 메시지의 순서를 판별하는 방법에 대해 설명합니다.

참고: MQGET을 두 번 이상 사용하려면(예를 들어, 큐에서 메시지를 단계별로 처리하려면) 각 호출 후에 이 구조의 *MsgId* 및 *CorrelId* 필드를 널로 설정해야 합니다. 이렇게 하면 이러한 필드에서 검색된 메시지의 ID가 지워집니다.

하지만 메시지를 그룹화하려면, 동일 그룹의 메시지에 대해 *GroupId*가 동일해야만 호출로 이전 메시지와 동일한 ID를 가진 메시지를 찾아 전체 그룹을 구성할 수 있습니다.

MQGMO 구조를 사용하여 MQGET 옵션 지정

MQGMO 구조는 옵션을 MQGET 호출로 전달하기 위한 입출력(I/O) 변수입니다. 다음 절은 이 구조의 필드 중 일부를 완료하는 데 도움을 줍니다.

MQGMO에는 MQGMO 구조에 대한 설명이 있습니다.

StrucId

*StrucId*는 구조를 메시지 가져오기 옵션 구조로 식별하는 데 사용되는 4문자 필드입니다. 항상 MQGMO_STRUC_ID를 지정하십시오.

Version

*Version*구조의 버전 번호를 설명합니다. MQGMO_VERSION_1이 기본값입니다. 버전 2 필드를 사용하거나 논리 순서대로 메시지를 검색하려면 MQGMO_VERSION_2를 지정하십시오. 버전 3 필드를 사용하거나 논리 순서대로 메시지를 검색하려면 MQGMO_VERSION_3을 지정하십시오. MQGMO_CURRENT_VERSION은 최신 레벨을 사용하도록 애플리케이션을 설정합니다.

Options

코드 내에서 원하는 순서대로 옵션을 선택할 수 있습니다. 각 옵션은 *Options* 필드에 비트로 나타냅니다.

Options 필드에서 제어하는 항목은 다음과 같습니다.

- MQGET 호출이 완료되기 전에 메시지가 큐에 도착할 때까지 대기하는지 여부(811 페이지의 『메시지 대기』 참조)

- Get 조작이 작업 단위에 포함되는지 여부
- 비지속 메시지가 메시지를 빠르게 수행하도록 동기점 외부에서 검색되는지 여부
- **z/OS** IBM MQ for z/OS에서 검색된 메시지가 백아웃 건너뛰기로 표시되는지 여부(813 페이지의 『백아웃 건너뛰기』 참조)
- 메시지가 큐에서 제거되는지 단순히 열람되는지 여부
- 메시지를 찾아보기 커서를 사용해서 선택할지 다른 선택 기준으로 선택할지 여부
- 메시지가 버퍼보다 길어도 호출이 성공하는지 여부
- **z/OS** IBM MQ for z/OS에서 호출이 완료되도록 허용할지 여부. 또한 이 옵션은 메시지가 도착할 때 알림을 받도록 표시하는 신호를 설정합니다.
- 큐 관리자가 정지 중 상태에 있는 경우 호출이 실패하는지 여부
- **z/OS** IBM MQ for z/OS에서 연결이 정지 중 상태인 경우 호출이 실패하는지 여부
- 애플리케이션 메시지 데이터 변환이 필요한지 여부(816 페이지의 『애플리케이션 데이터 변환』 참조)
- 큐에서 메시지와 세그먼트를 검색하는 순서 **z/OS** (IBM MQ for z/OS는 제외)
- 완료된 논리 메시지만 검색 가능한지 여부 **z/OS** (IBM MQ for z/OS는 제외)
- 그룹의 모든 메시지가 사용 가능한 경우에만 그룹의 메시지를 검색할 수 있는지 여부
- 논리 메시지의 모든 세그먼트가 사용 가능한 경우에만 논리 메시지의 세그먼트를 검색할 수 있는지 여부 **z/OS** (IBM MQ for z/OS는 제외)

Options 필드를 기본값(MQGMO_NO_WAIT)으로 설정해 두면 MQGET 호출이 이런 방법으로 작동합니다.

- 큐에 선택 기준과 일치하는 메시지가 없는 경우, 메시지가 도착할 때까지 대기하지 않고 호출이 즉시 완료됩니다. **z/OS** 또한 IBM MQ for z/OS에서 호출은 이러한 메시지가 도착할 때 알림을 요청하는 신호를 설정하지 않습니다.
- 호출이 동기점에 작용하는 방식은 플랫폼에 의해 판별됩니다.

플랫폼	동기점 제어 여부
IBM i	아니오
AIX and Linux 시스템	아니오
z/OS z/OS z/OS	예
Windows 시스템	아니오

- **z/OS** IBM MQ for z/OS에서는 검색된 메시지가 백아웃 건너뛰기로 표시되지 않습니다.
- 선택된 메시지가 큐에서 제거됩니다(검색되지 않음).
- 애플리케이션 메시지 데이터 변환이 필요하지 않습니다.
- 메시지가 버퍼보다 더 길면 호출은 실패합니다.

WaitInterval

WaitInterval 필드는 MQGMO_WAIT 옵션을 사용할 때 메시지가 큐에 도착하기까지 MQGET 호출이 대기하는 최대 시간(밀리초)을 지정합니다. 메시지가 *WaitInterval*에 지정된 시간 내에 도착하지 않으면, 호출은 큐에서 사용자의 선택 기준과 일치한 메시지가 없음을 표시하는 이유 코드를 완료하고 리턴합니다.

z/OS MQGMO_SET_SIGNAL 옵션을 사용하는 경우 IBM MQ for z/OS, 필드는 *WaitInterval* 신호가 설정되는 시간을 지정합니다.

이러한 옵션에 대한 자세한 정보는 811 페이지의 『메시지 대기』 **z/OS** 및 812 페이지의 『신호 보내기』의 내용을 참조하십시오.

▶ z/OS **Signal1**

Signal1. IBM MQ for z/OS에서만 지원됩니다.

적당한 메시지가 도착할 때 애플리케이션이 알림을 받도록 요청하기 위해 MQGMO_SET_SIGNAL 옵션을 사용하는 경우 *Signal1* 필드에서 신호의 유형을 지정합니다. 다른 모든 플랫폼에서는 IBM MQ 해당 *Signal1* 필드가 예약되어 있으며 그 값은 의미가 없습니다.

▶ z/OS 자세한 정보는 812 페이지의 『신호보내기』의 내용을 참조하십시오.

Signal2

Signal2 필드는 모든 플랫폼에서 예약되고 해당 값은 중요하지 않습니다.

▶ z/OS 자세한 정보는 812 페이지의 『신호보내기』의 내용을 참조하십시오.

ResolvedQName

*ResolvedQName*은 큐 관리자가 메시지를 검색한 큐의 이름(모든 알리어스 해석 이후)을 리턴하는 출력 필드입니다.

MatchOptions

*MatchOptions*는 MQGET에 대한 선택 기준을 제어합니다.

GroupStatus

*GroupStatus*는 검색한 메시지가 그룹에 있는지 여부를 표시합니다.

SegmentStatus

*SegmentStatus*는 검색한 항목이 논리 메시지의 세그먼트인지 여부를 표시합니다.

Segmentation

*Segmentation*은 검색된 메시지에 대해 세그먼트화가 허용되는지 여부를 표시합니다.

MsgToken

*MsgToken*은 메시지를 고유하게 식별합니다.

ReturnedLength

*ReturnedLength*는 큐 관리자가 리턴된 메시지 데이터의 길이(바이트)를 리턴하는 출력 필드입니다.

MsgHandle

큐에서 검색 중인 메시지의 특성으로 채워질 메시지에 대한 핸들. 핸들은 이전에 MQCRTMH 호출로 작성되었습니다. 이미 핸들과 연관된 특성은 메시지를 검색하기 전에 지워집니다.

버퍼 영역의 크기 지정

MQGET 호출의 **BufferLength** 매개변수에서 검색하는 메시지 데이터를 보유하도록 버퍼 영역의 크기를 지정하십시오. 이 크기는 세 가지 방법으로 결정합니다.

1. 이 프로그램에서 예상하는 메시지의 길이를 이미 알고 있습니다. 이 경우 이 크기의 버퍼를 지정하십시오.

그러나 메시지가 버퍼에 비해 너무 크더라도 MQGET 호출을 완료하려는 경우 MQGMO 구조에서 MQGMO_ACCEPT_TRUNCATED_MSG 옵션을 사용할 수 있습니다. 이 경우 다음과 같은 상황이 발생합니다.

- 버퍼가 보유할 수 있는 만큼의 메시지로 채워집니다.
- 호출이 경고 완료 코드를 리턴합니다.
- 메시지가 큐에서 제거되거나(나머지 메시지 제거) 또는 찾아보기 커서가 앞으로 이동합니다(큐를 찾아볼 경우).
- 메시지의 실제 길이가 *DataLength*에서 리턴됨

이 옵션이 없어도 호출은 여전히 경고와 함께 완료되지만 큐에서 메시지를 제거하지는 않습니다(또는 찾아보기 커서를 전진시킴).

2. 버퍼의 크기를 평가하거나 0바이트로 지정하고 MQGMO_ACCEPT_TRUNCATED_MSG 옵션을 사용하지 마십시오. 예를 들어, 버퍼가 너무 작아서 MQGET 호출이 실패하는 경우, 메시지의 길이가 호출의 **DataLength** 매개변수에 리턴됩니다. (버퍼는 여전히 보유할 수 있는 만큼의 메시지로 채워지지만 호출의 처리가 완료되지

않습니다.) 이 메시지의 *MsgId*를 저장한 후, MQGET 호출을 반복하여 정확한 크기의 버퍼 영역과 첫 번째 호출에서 참고한 *MsgId*를 지정하십시오.

프로그램이 다른 프로그램에서 제공하는 큐를 제공할 경우, 이러한 다른 프로그램 중 하나는 프로그램이 또 MQGET 호출을 발행하기 전에 사용자가 원하는 메시지를 제거합니다. 프로그램은 더 이상 존재하지 않는 메시지를 검색하는 데 시간을 낭비할 수 있습니다. 이 문제를 방지하려면 먼저 *BufferLength*를 0으로 지정하고 MQGMO_ACCEPT_TRUNCATED_MSG 옵션을 사용하여 원하는 메시지를 찾을 때까지 큐를 찾아보십시오. 이 경우 찾아보기 커서 위치가 원하는 메시지 아래에 지정됩니다. 그런 다음, MQGET을 다시 호출하여 MQGMO_MSG_UNDER_CURSOR 옵션을 지정하는 메시지를 검색할 수 있습니다. 다른 프로그램이 찾아보기 호출과 제거 호출 사이의 메시지를 제거하면, 찾아보기 커서 아래에 메시지가 없기 때문에 두 번째 MQGET이 (전체 큐를 검색하지 않고) 즉시 실패합니다.

3. *MaxMsgLength* 큐 속성은 해당 큐에 대해 허용된 메시지의 최대 길이를 판별합니다. *MaxMsgLength* 큐 관리자 속성은 해당 큐 관리자에 대해 허용된 메시지의 최대 길이를 판별합니다. 예상하는 메시지의 길이를 모르는 경우, MQINQ 호출을 사용하여 *MaxMsgLength* 속성에 대해 조회한 후 이 크기의 버퍼를 지정할 수 있습니다.

성능 저하를 방지하려면 버퍼 크기를 가능한 한 실제 메시지 크기에 가깝게 설정하십시오.

MaxMsgLength 속성에 대한 추가 정보는 807 페이지의 『4MB 이상의 메시지 처리』의 내용을 참조하십시오.

큐에서 메시지를 검색하는 순서

큐에서 메시지를 검색하는 순서를 제어할 수 있습니다. 이 절에서는 옵션을 살펴봅니다.

Priority

프로그램은 큐에 메시지를 넣을 때 메시지에 우선순위를 지정할 수 있습니다(32 페이지의 『메시지 우선순위』 참조). 동일 우선순위의 메시지는 커밋된 순서가 아니라 도착한 순서대로 큐에 저장됩니다.

큐 관리자는 엄격한 FIFO(First In, First Out) 순서로 또는 우선순위 내 FIFO 순서로 큐를 유지보수합니다. 이는 큐의 *MsgDeliverySequence* 속성 설정에 따라 다릅니다. 메시지가 큐에 도착하면 우선순위가 같은 마지막 메시지 바로 뒤에 삽입됩니다.

프로그램은 큐에서 첫 번째 메시지를 가져오거나, 해당 메시지의 우선순위를 무시하고 큐에서 특정 메시지를 가져올 수 있습니다. 예를 들어, 프로그램은 이미 송신한 특정 메시지에 대한 회신을 처리할 수 있습니다. 자세한 정보는 801 페이지의 『특정 메시지 가져오기』의 내용을 참조하십시오.

애플리케이션이 큐에 일련의 메시지를 넣으면, 다른 애플리케이션은 다음 조건 하에서 해당 메시지가 놓여진 순서와 동일한 순서로 해당 메시지를 검색할 수 있습니다.

- 메시지가 모두 동일한 우선순위를 가짐
- 메시지가 모두 동일한 작업 단위 내에 놓여졌거나 모두 작업 단위의 외부에 놓여졌음
- 큐가 넣기 애플리케이션에 대해 로컬임

이러한 조건이 충족되지 않고 애플리케이션이 특정 순서로 검색되는 메시지에 따라 달라지는 경우, 애플리케이션은 메시지 데이터에 순서 정보를 포함하거나 다음 메시지가 송신되기 전에 메시지의 수신을 확인하는 방법을 설정해야 합니다.

 IBM MQ for z/OS에서 큐 속성 *IndexType*을 사용하여 큐에서 MQGET 조작의 속도를 올릴 수 있습니다. 자세한 정보는 806 페이지의 『색인 유형』의 내용을 참조하십시오.

논리적 및 물리적 정렬

각 우선순위 수준 내에서, 큐에 있는 메시지는 물리적 순서 또는 논리적 순서로 발생할 수 있습니다.

물리 순서는 메시지가 큐에 도착하는 순서입니다. 논리 순서는 그룹 내의 모든 메시지 및 세그먼트가 해당 논리 시퀀스에서 그룹에 속하는 첫 번째 항목의 물리적 위치로 판별된 위치에 서로 나란히 배열된 경우를 말합니다.

그룹, 메시지 및 세그먼트에 대한 설명은 48 페이지의 『메시지 그룹』의 내용을 참조하십시오. 이러한 물리 및 논리 순서는 다음과 같은 이유로 달라질 수 있습니다.

- 다양한 애플리케이션의 그룹이 비슷한 시기에 목적지에 도착하면 이 때문에 별개의 물리 순서가 손실될 수 있습니다.

- 단일 그룹 내에서도 그룹에 있는 일부 메시지의 경로 재지정 또는 지연으로 인해 순서가 뒤바뀔 수 있습니다. 예를 들어, 논리 순서는 그림 792 페이지의 그림 91와 같습니다.

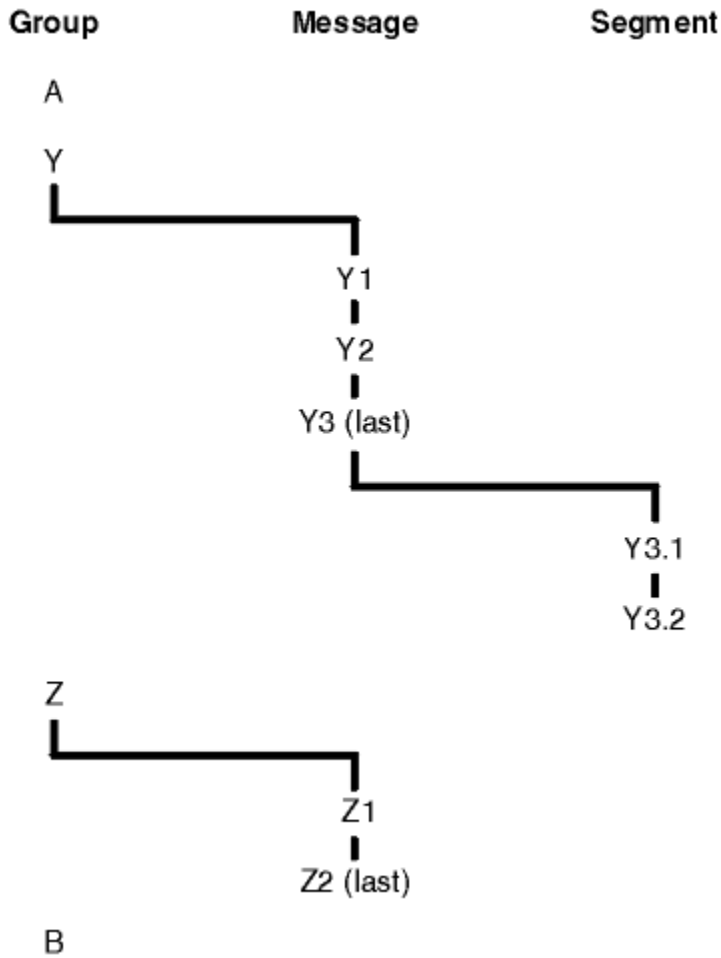
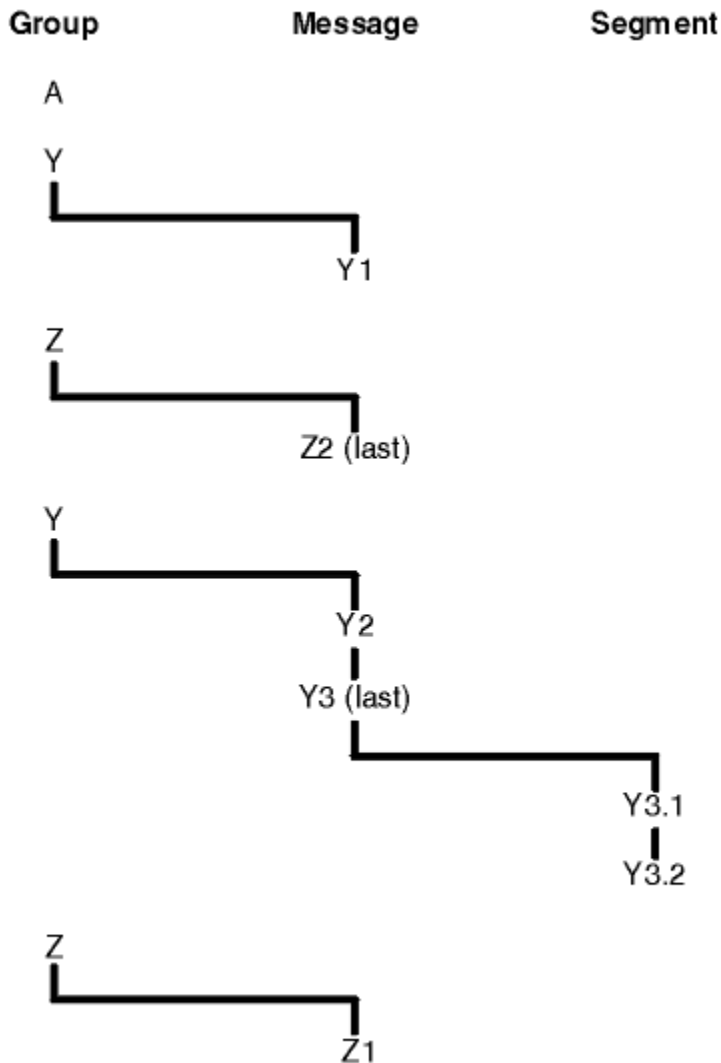


그림 91. 큐의 논리 순서

이러한 메시지는 큐에서 다음과 같은 논리 순서대로 발생합니다.

1. 메시지 A(그룹에 속하지 않음)
2. 그룹 Y의 논리 메시지 1
3. 그룹 Y의 논리 메시지 2
4. 그룹 Y의 (마지막) 논리 메시지 3의 세그먼트 1
5. 그룹 Y의 (마지막) 논리 메시지 3의 (마지막) 세그먼트 2
6. 그룹 Z의 논리 메시지 1
7. 그룹 Z의 (마지막) 논리 메시지 2
8. 메시지 B(그룹에 속하지 않음)

하지만 물리 순서는 전적으로 다릅니다. 각 그룹 안에 있는 첫 번째 항목의 물리적 위치는 전체 그룹의 논리적 위치를 판별합니다. 예를 들어, 그룹 Y와 Z가 비슷한 시간에 도착하였고 그룹 Z의 메시지 2가 동일한 그룹의 메시지 1을 추월한 경우 물리 순서는 그림 793 페이지의 그림 92와 같습니다.



B

그림 92. 큐의 물리 순서

이러한 메시지는 큐에서 다음과 같은 물리 순서대로 발생합니다.

1. 메시지 A(그룹에 속하지 않음)
2. 그룹 Y의 논리 메시지 1
3. 그룹 Z의 논리 메시지 2
4. 그룹 Y의 논리 메시지 2
5. 그룹 Y의 (마지막) 논리 메시지 3의 세그먼트 1
6. 그룹 Y의 (마지막) 논리 메시지 3의 (마지막) 세그먼트 2
7. 그룹 Z의 논리 메시지 1
8. 메시지 B(그룹에 속하지 않음)

참고:  IBM MQ for z/OS에서 큐가 GROUPID로 색인화된 경우에는 큐 메시지의 물리 순서가 보장되지 않습니다.

메시지를 가져올 때, 물리 순서가 아닌 논리 순서대로 메시지를 검색하도록 MQGMO_LOGICAL_ORDER를 지정할 수 있습니다.

MQGMO_BROWSE_FIRST 및 MQGMO_LOGICAL_ORDER를 사용하여 MQGET 호출을 발행하는 경우, MQGMO_BROWSE_NEXT를 사용한 후속 MQGET 호출 또한 MQGMO_LOGICAL_ORDER를 지정해야 합니다.

반대로, MQGMO_BROWSE_FIRST를 사용한 MQGET이 MQGMO_LOGICAL_ORDER를 지정하지 않는 경우에는 MQGMO_BROWSE_NEXT를 사용한 다음 MQGET도 마찬가지여야 합니다.

큐에서 메시지를 찾아보는 MQGET 호출에 대해 큐 관리자가 보유하는 그룹 및 세그먼트 정보는 큐에서 메시지를 제거하는 MQGET 호출에 대해 큐 관리자가 보유하는 그룹 및 세그먼트 정보와는 별개입니다.

MQGMO_BROWSE_FIRST를 지정할 때, 큐 관리자는 찾아보는 그룹 및 세그먼트 정보를 무시하고 현재 그룹도 현재 논리 메시지도 없는 것처럼 큐를 스캔합니다.

참고: MQGMO_LOGICAL_ORDER를 지정하지 않고 메시지 그룹(또는 그룹에 속하지 않는 논리 메시지)의 범위 밖에서 찾아볼 경우 MQGET 호출을 사용하지 마십시오. 예를 들어, 큐에서 그룹의 첫 번째 메시지보다 그룹의 마지막 메시지가 앞서는 경우 그룹 범위 밖에서 MQGMO_BROWSE_NEXT를 사용해 찾아보고 *MsgSeqNumber*가 1(다음 그룹의 첫 번째 메시지를 찾을)로 설정된 MQMO_MATCH_MSG_SEQ_NUMBER를 지정하면 이미 찾아본 그룹의 첫 번째 메시지가 다시 리턴됩니다. 이는 즉시 발생하거나 (중간 그룹이 있는 경우) 나중에 다수의 MQGET 호출이 발생할 수 있습니다.

찾아볼 큐를 두 번 열어서 무한 루프 발생 가능성을 배제하십시오.

- 각 그룹에서 첫 번째 메시지만 찾아보려면 첫 번째 핸들을 사용하십시오.
- 특정 그룹 내의 메시지만 찾아보려면 두 번째 핸들을 사용하십시오.
- 그룹에서 메시지를 찾아보기 전에 두 번째 찾아보기 커서를 첫 번째 찾아보기 커서의 위치로 이동하려면 MQMO_* 옵션을 사용하십시오.
- 그룹 범위 밖에서 찾아볼 경우 MQGMO_BROWSE_NEXT를 사용하지 마십시오.

이에 대한 추가 정보는 [MQGET](#), [MQMD](#) 및 [MQI](#) 옵션 유효성 검증에 대한 규칙을 참조하십시오.

대부분의 애플리케이션에서는 찾아볼 때 논리적 또는 물리적 정렬을 선택하게 됩니다. 하지만 이러한 모드 간에 전환하려면 먼저 MQGMO_LOGICAL_ORDER로 찾아보기를 실행할 때 논리 시퀀스 내에서 사용자의 위치가 설정된다는 점을 기억해 두십시오.

이 시점에서 그룹 내에 첫 번째 항목이 존재하지 않으면 사용자가 속한 그룹이 논리 시퀀스의 부분으로 간주되지 않습니다.

찾아보기 커서는 한 번 그룹에 포함되면 첫 번째 메시지가 제거된 경우에도 동일한 그룹 내에서 계속할 수 있습니다. 하지만 처음에는 MQGMO_LOGICAL_ORDER를 사용하여 첫 번째 항목이 존재하지 않는 그룹으로 이동할 수 없습니다.

MQPMO_LOGICAL_ORDER

MQPMO 옵션은 애플리케이션이 논리 메시지의 세그먼트 및 그룹에 메시지를 넣는 방법을 큐 관리자에 알려줍니다. 이 옵션은 MQPUT 호출에만 지정할 수 있으며 MQPUT1 호출에서는 유효하지 않습니다.

이 MQPMO_LOGICAL_ORDER 지정된 경우, 이는 애플리케이션이 연속적인 MQPUT 호출을 사용하여 다음을 수행함을 의미합니다:

1. 증가하는 세그먼트 오프셋(0부터 시작) 순으로 간격 없이 세그먼트를 각 논리 메시지에 넣습니다.
2. 세그먼트를 다음 논리 메시지에 넣기 전에 모든 세그먼트를 하나의 논리 메시지에 넣습니다.
3. 증가하는 세그먼트 순서 번호(1부터 시작) 순으로 간격 없이 논리 메시지를 각 메시지 그룹에 넣습니다. IBM MQ에서는 메시지 순서 번호가 자동으로 증가됩니다.
4. 논리 메시지를 다음 메시지 그룹에 넣기 전에 모든 논리 메시지를 하나의 메시지 그룹에 넣습니다.

애플리케이션이 논리 메시지의 세그먼트 및 그룹에 메시지를 넣는 방법을 큐 관리자에 알렸고 큐 관리자가 각 MQPUT 호출에 대한 그룹 및 세그먼트 정보를 유지보수하고 업데이트하기 때문에 애플리케이션은 이 정보를 유지보수하고 업데이트할 필요가 없습니다. 특히, 이는 큐 관리자가 이러한 필드를 적절한 값으로 설정하기 때문에 애플리케이션이 MQMD에서 *GroupId*, *MsgSeqNumber* 및 *Offset* 필드를 설정할 필요가 없다는 것을 의미합니다. 애플리케이션은 메시지가 그룹에 속하거나 논리 메시지의 세그먼트인 경우를 나타내고 그룹의 마지막 메시지 또는 논리 메시지의 마지막 세그먼트를 나타낼 때만 MQMD의 *MsgFlags* 필드를 설정해야 합니다.

메시지 그룹 또는 논리 메시지가 시작되면, 후속 MQPUT 호출은 MQMD의 *MsgFlags*에 적절한 MQMF_* 플래그를 지정해야 합니다. 애플리케이션에서 종료되지 않은 메시지 그룹이 있을 때 그룹에 속하지 않는 메시지를 넣으려고 하거나 종료되지 않은 논리 메시지가 있을 때 세그먼트가 아닌 메시지를 넣으려고 하면, 적절한 경우 이유 코드 MQRC_INCOMPLETE_GROUP 또는 MQRC_INCOMPLETE_MSG와 함께 호출이 실패합니다. 그러나 큐 관리자는 현재 메시지 그룹 또는 현재 논리 메시지에 관한 정보를 보유하며, 애플리케이션은

그룹에 속하지 않거나 세그먼트가 아닌 메시지를 넣도록 MQPUT 호출을 재발행하기 전에 MQMF_LAST_MSG_IN_GROUP 또는 MQMF_LAST_SEGMENT를 적절하게 지정하는 메시지(가급적 애플리케이션 메시지 데이터가 포함되지 않은)를 송신하여 해당 호출을 종료할 수 있습니다.

793 페이지의 그림 92는 올바른 플래그와 옵션의 조합 및 큐 관리자가 각 사례에서 사용하는 *GroupId*, *MsgSeqNumber* 및 *Offset* 필드의 값을 보여줍니다. 표에 표시되지 않은 옵션과 플래그의 조합은 유효하지 않습니다. 표의 열은 다음과 같은 의미를 가지고 있으며, 둘 중 하나는 예 또는 아니오를 의미합니다.

LOG ORD

MQPMO_LOGICAL_ORDER 옵션이 호출에 지정되어 있는지 여부를 나타냅니다.

MIG

MQMF_MSG_IN_GROUP 또는 MQMF_LAST_MSG_IN_GROUP 옵션이 호출에 지정되어 있는지 여부를 나타냅니다.

SEG

MQMF_SEGMENT 또는 MQMF_LAST_SEGMENT 옵션이 호출에 지정되어 있는지 여부를 나타냅니다.

SEG OK

MQMF_SEGMENTATION_ALLOWED 옵션이 호출에 지정되어 있는지 여부를 나타냅니다.

Cur grp

호출 전에 현재 메시지 그룹이 존재하는지 여부를 나타냅니다.

Cur log msg

호출 전에 현재 논리 메시지가 존재하는지 여부를 나타냅니다.

기타 열

큐 관리자가 사용하는 값을 보여줍니다. '이전'은 큐 핸들의 이전 메시지 필드에 사용된 값을 나타냅니다.

표 131. 논리 메시지의 세그먼트 및 그룹의 메시지에 관한 MQPUT 옵션								
지정하는 옵션	지정하는 옵션	지정하는 옵션	지정하는 옵션	호출 이전의 그룹 및 로그 메시지 상태	호출 이전의 그룹 및 로그 메시지 상태	큐 관리자가 사용하는 값	큐 관리자가 사용하는 값	큐 관리자가 사용하는 값
LOG ORD	MIG	SEG	SEG OK	Cur grp	Cur log msg	GroupId	MsgSeqNumber	Offset
예	아니오	아니오	아니오	아니오	아니오	MQGI_NONE	1	0
예	아니오	아니오	예	아니오	아니오	새 그룹 ID	1	0
예	아니오	예	둘 중 하나	아니오	아니오	새 그룹 ID	1	0
예	아니오	예	둘 중 하나	아니오	예	이전 그룹 ID	1	이전 오프셋 + 이전 세그먼트 길이
예	예	둘 중 하나	둘 중 하나	아니오	아니오	새 그룹 ID	1	0
예	예	둘 중 하나	둘 중 하나	예	아니오	이전 그룹 ID	이전 순서 번호 + 1	0
예	예	예	둘 중 하나	예	예	이전 그룹 ID	이전 순서 번호	이전 오프셋 + 이전 세그먼트 길이
아니오	아니오	아니오	아니오	둘 중 하나	둘 중 하나	MQGI_NONE	1	0

표 131. 논리 메시지의 세그먼트 및 그룹의 메시지에 관한 MQPUT 옵션 (계속)								
지정하는 옵션	지정하는 옵션	지정하는 옵션	지정하는 옵션	호출 이전의 그룹 및 로그 메시지 상태	호출 이전의 그룹 및 로그 메시지 상태	큐 관리자가 사용하는 값	큐 관리자가 사용하는 값	큐 관리자가 사용하는 값
아니오	아니오	아니오	예	둘 중 하나	둘 중 하나	MQGI_NONE이면 새 그룹 ID이고, 그렇지 않으면 필드 값	1	0
아니오	아니오	예	둘 중 하나	둘 중 하나	둘 중 하나	MQGI_NONE이면 새 그룹 ID이고, 그렇지 않으면 필드 값	1	필드 값
아니오	예	아니오	둘 중 하나	둘 중 하나	둘 중 하나	MQGI_NONE이면 새 그룹 ID이고, 그렇지 않으면 필드 값	필드 값	0
아니오	예	예	둘 중 하나	둘 중 하나	둘 중 하나	MQGI_NONE이면 새 그룹 ID이고, 그렇지 않으면 필드 값	필드 값	필드 값

참고:

- MQPMO_LOGICAL_ORDER가 MQPUT1 호출에 유효하지 않습니다.
- *MsgId* 필드의 경우, 큐 관리자는 MQPMO_NEW_MSG_ID 또는 MQMI_NONE이 지정되면 새 메시지 ID를 생성하고 그렇지 않으면 필드의 값을 사용합니다.
- *CorrelId* 필드의 경우, 큐 관리자는 MQPMO_NEW_CORREL_ID가 지정되면 새 상관 ID를 생성하고 그렇지 않으면 필드의 값을 사용합니다.

MQPMO_LOGICAL_ORDER를 지정하면, 큐 관리자는 논리 메시지의 세그먼트 및 그룹의 모든 메시지에서 MQMD의 *Persistence* 필드에 동일한 값(즉, 모두 지속적이거나 모두 비지속적이어야 함)을 넣도록 요구합니다. 이 조건을 충족하지 않으면 MQPUT 호출이 이유 코드 MQRC_INCONSISTENT_PERSISTENCE와 함께 실패합니다.

MQPMO_LOGICAL_ORDER 옵션은 다음과 같이 작업 단위에 영향을 줍니다.

- 논리 메시지 또는 그룹의 첫 번째 물리적 메시지를 작업 단위 안에 넣으면, 동일한 큐 핸들을 사용하는 경우 논리 메시지 또는 그룹의 기타 모든 물리적 메시지를 작업 단위 안에 넣어야 합니다. 그러나 많은 물리적 메시지로 구성되는 논리 메시지 또는 메시지 그룹을 큐 핸들에 대한 둘 이상의 연속 작업 단위로 분할할 수 있도록 하면 해당 메시지를 동일한 작업 단위 안에 넣지 않아도 됩니다.
- 그룹 또는 논리 메시지의 첫 번째 실제 메시지를 작업 단위 내에 넣지 않은 경우 동일한 큐 핸들이 사용되면 그룹 또는 논리 메시지의 다른 모든 실제 메시지를 작업 단위 내에 넣을 수 없어야 합니다.

이러한 조건을 충족하지 않으면 MQPUT 호출이 이유 코드 MQRC_INCONSISTENT_UOW와 함께 실패합니다.

MQPMO_LOGICAL_ORDER를 지정하면 MQPUT 호출에서 제공된 MQMD가 MQMD_VERSION_2 이상이어야 합니다. 이 조건을 충족하지 않으면 호출이 이유 코드 MQRC_WRONG_MD_VERSION과 함께 실패합니다.

MQPMO_LOGICAL_ORDER를 지정하지 않으면 논리 메시지의 세그먼트 및 그룹의 메시지를 원하는 순서대로 넣을 수 있으므로 완료 메시지 그룹 또는 완료 논리 메시지를 넣을 필요가 없습니다. *GroupId*, *MsgSeqNumber*, *Offset* 및 *MsgFlags* 필드에 적절한 값이 있는지 확인하는 것은 애플리케이션의 책임입니다.

시스템 실패가 발생한 후, 중간에 메시지 그룹 또는 논리 메시지를 재시작하려면 이 기술을 사용하십시오. 시스템을 다시 시작할 때 애플리케이션은 *GroupId*, *MsgSeqNumber*, *Offset*, *MsgFlags* 및

Persistence 필드를 적절한 값으로 설정할 수 있으며, 그 다음에는 MQPMO_LOGICAL_ORDER를 지정하지 않고도 필요에 따라 MQPMO_SYNCPOINT 또는 MQPMO_NO_SYNCPOINT가 설정된 MQPUT 호출을 발행할 수 있습니다. 이 호출이 성공하면 큐 관리자가 그룹 및 세그먼트 정보를 보유하며, 해당 큐 핸들을 사용하여 후속 MQPUT 호출 시 MQPMO_LOGICAL_ORDER가 정상적으로 지정될 수 있습니다.

큐 관리자가 MQPUT 호출에 대해 보유하는 그룹 및 세그먼트 정보는 MQGET 호출에 대해 보유하는 그룹 및 세그먼트 정보와 분리됩니다.

지정된 큐 핸들에 대해 애플리케이션은 MQPMO_LOGICAL_ORDER를 지정하는 MQPUT 호출과 지정하지 않는 MQPUT 호출을 혼합할 수 있습니다. 단, 다음 사항에 유의하십시오.

- MQPMO_LOGICAL_ORDER를 지정하지 않으면 MQPUT 호출이 성공할 때마다 큐 관리자가 큐 핸들에 대한 그룹 및 세그먼트 정보를 애플리케이션이 지정한 값으로 설정하므로 큐 핸들에 대해 큐 관리자가 보유한 기존 그룹 및 세그먼트 정보가 바뀝니다.
- MQPMO_LOGICAL_ORDER를 지정하지 않으면 현재 메시지 그룹 또는 논리 메시지가 있는 경우에도 호출은 실패하지 않습니다. 호출은 MQCC_WARNING 완료 코드와 함께 성공합니다. 797 페이지의 표 132에서는 발생할 수 있는 다양한 사례를 보여줍니다. 이러한 경우, 완료 코드가 MQCC_OK가 아니면 이유 코드는 다음 중 하나가 됩니다(해당되는 경우).
 - MQRC_INCOMPLETE_GROUP
 - MQRC_INCOMPLETE_MSG
 - MQRC_INCONSISTENT_PERSISTENCE
 - MQRC_INCONSISTENT_UOW

참고: 큐 관리자는 MQPUT1 호출을 위한 그룹 및 세그먼트 정보를 확인하지 않습니다.

표 132. MQPUT 또는 MQCLOSE 호출이 그룹 및 세그먼트 정보와 일치하지 않을 때의 결과		
현재 호출	이전 호출이 MQPMO_LOGICAL_ORDER가 있는 MQPUT임	이전 호출이 MQPMO_LOGICAL_ORDER가 없는 MQPUT임
MQPMO_LOGICAL_ORDER가 있는 MQPUT	MQCC_FAILED	MQCC_FAILED
MQPMO_LOGICAL_ORDER가 없는 MQPUT	MQCC_WARNING	MQCC_OK
종료되지 않은 그룹 또는 논리 메시지 의 MQCLOSE	MQCC_WARNING	MQCC_OK

메시지와 세그먼트를 논리 순서에 따라 넣는 애플리케이션의 경우 MQPMO_LOGICAL_ORDER를 지정하십시오. 이는 사용할 수 있는 가장 단순한 옵션입니다. 이 옵션은 큐 관리자가 이들 정보를 관리하기 때문에 애플리케이션이 그룹 및 세그먼트 정보를 관리할 필요가 없습니다. 하지만 특수 애플리케이션에서는 MQPMO_LOGICAL_ORDER 옵션이 제공하는 것보다 더 많은 제어가 필요하고 이런 목적은 해당 옵션을 지정하지 않아야만 달성될 수 있습니다. 이럴 경우, 각 MQPUT 또는 MQPUT1 호출 전에 MQMD의 *GroupId*, *MsgSeqNumber*, *Offset* 및 *MsgFlags* 필드가 올바르게 설정되었는지 확인해야 합니다.

예를 들어, 수신되는 물리적 메시지를 전달할 애플리케이션은 해당 메시지가 그룹에 속하는지 논리 메시지의 세그먼트인지 여부에 관계없이 다음 두 가지 이유로 MQPMO_LOGICAL_ORDER를 지정하지 않아야 합니다.

- 메시지를 검색하고 순서대로 넣으면 MQPMO_LOGICAL_ORDER 지정 시 메시지에 새 그룹 ID가 지정됩니다. 이 경우 메시지 그룹에서 생성되는 모든 응답 또는 보고 메시지를 메시지의 진원지에서 상관시키기가 어렵거나 불가능할 수 있습니다.
- 송신 및 수신 큐 관리자 사이에 다중 경로가 포함된 복잡한 네트워크에서는 실제 메시지가 순서없이 도착합니다. MQGET 호출에 MQPMO_LOGICAL_ORDER 및 MQGMO_LOGICAL_ORDER를 지정하지 않으면, 전달 애플리케이션은 논리 순서상 다음 메시지가 도착할 때까지 대기하지 않고 개별 물리적 메시지가 도착하자마자 검색하고 전달할 수 있습니다.

논리 메시지의 세그먼트 또는 그룹의 메시지에 대한 보고 메시지를 생성하는 애플리케이션은 보고 메시지를 넣을 때에도 MQPMO_LOGICAL_ORDER를 지정하지 않아야 합니다.

MQPMO_LOGICAL_ORDER는 다른 MQPMO_* 옵션과 함께 지정할 수 있습니다.

클러스터 큐(MQOO_BIND_ON_GROUP)에 논리적으로 정렬된 그룹 넣기

MQOO_BIND_ON_OPEN 옵션은 이 애플리케이션의 모든 메시지와 결과로 생기는 모든 그룹이 단일 인스턴스로 라우팅되는지 확인합니다. 이 옵션은 애플리케이션 트래픽이 클러스터 큐의 여러 인스턴스에 걸쳐 로드 밸런싱 되지 않는 단점이 있습니다. 메시지 그룹을 그대로 유지하면서 워크로드 밸런싱을 사용하도록 하려면 다음 옵션을 설정해야 합니다.

- MQPUT 호출은 MQPMO_LOGICAL_ORDER를 지정해야 합니다.
- MQOPEN 호출은 다음 두 가지 옵션 중 하나를 지정해야 합니다.
 - MQOO_BIND_ON_GROUP
 - MQOO_BIND_AS_Q_DEF. 그리고 큐 정의는 DEFBIND(GROUP)를 지정해야 합니다.

그러면 큐의 MQCLOSE 및 MQOPEN을 실행하지 않아도 메시지 그룹 간 워크로드 밸런싱이 구동됩니다. 그룹 간은 MQMD(v2) 또는 MQMDE에 MQMF_MSG_IN_GROUP이 설정되어 있고 부분적으로 진행 중인 완료 그룹이 없음을 의미합니다. 그룹이 진행 중일 때, 오브젝트 핸들에서 해석된 큐 관리자 및 큐 이름이 재사용됩니다.

이전 메시지가 MQPMO_LOGICAL_ORDER이고/거나 MQMF_MSG_IN_GROUP이 설정되었지만 현재 메시지가 그룹의 일부가 아니면 PUT 호출이 MQRC_INCOMPLETE_GROUP으로 실패합니다.

개별 MQPUT이 MQPMO_LOGICAL_ORDER를 지정하지 않으면 현재 그룹이 활성화되지 않은 경우 (MQOPEN 호출이 MQOO_BIND_NOT_FIXED를 지정한 것처럼) 해당 메시지에 대한 워크로드 밸런싱이 구동됩니다.

MQOO_BIND_ON_GROUP을 사용하여 목적지에 바인드된 메시지에 대해서는 재할당이 일어나지 않습니다. 재할당에 대한 자세한 정보는 48 페이지의 『메시지 그룹』의 내용을 참조하십시오.

논리 메시지 그룹화

그룹의 논리 메시지를 사용하는 두 가지 주된 이유가 있습니다.

- 특정 순서로 메시지를 처리해야 합니다.
- 그룹의 각 메시지를 관련 방법으로 처리해야 합니다.

각각의 경우에 동일한 가져오기 애플리케이션 인스턴스로 전체 그룹을 검색하십시오.

예를 들어, 그룹이 4가지 논리 메시지로 구성된다고 가정하십시오. 넣기 애플리케이션은 다음과 같습니다.

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP

MQCMIT
```

가져오기 애플리케이션은 그룹의 첫 번째 메시지에 대해 MQGMO_ALL_MSGS_AVAILABLE 옵션을 지정합니다. 이렇게 하면 그룹 내 모든 메시지가 도착할 때까지 처리가 시작되지 않습니다. 그룹 내 후속 메시지에 대해서는 MQGMO_ALL_MSGS_AVAILABLE 옵션이 무시됩니다.

그룹의 첫 번째 논리 메시지를 검색할 때, MQGMO_LOGICAL_ORDER를 사용하여 그룹의 나머지 논리 메시지가 순서대로 검색되는지 확인할 수 있습니다.

따라서 가져오기 애플리케이션은 다음과 같습니다.

```
/* Wait for the first message in a group, or a message not in a group */
GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
              | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
  MQGET
  /* Process each remaining message in the group */
  ...
MQCMIT
```

메시지 그룹화에 대한 추가 예제는 809 페이지의 『논리 메시지의 애플리케이션 세그먼트화』 및 799 페이지의 『작업 단위에 걸친 그룹 넣기 및 가져오기』의 내용을 참조하십시오.



주의: 발행/구독을 사용하여 토픽에 메시지를 송신하거나 토픽 알리어스에 메시지를 넣을 때 메시지 그룹화 및 세그먼트화가 허용되지 않습니다.

구독은 발행 활동과 독립적으로 작성되고 제거될 수 있으므로 구독자가 전체 메시지 그룹 또는 메시지의 모든 세그먼트를 수신할 것이라고 확신할 수 없습니다. **RC2417:**

MQRC_MSG_NOT_ALLOWED_IN_GROUP의 내용을 참조하십시오.

애플리케이션을 통해 그룹의 메시지를 모두 클러스터 큐의 동일한 목적지 인스턴스에 할당하도록 요청하는 방법에 대한 정보는 **DefBind**를 참조하십시오.

작업 단위에 걸친 그룹 넣기 및 가져오기

이전 사례에서는 전체 그룹 넣기가 완료되고 작업 단위가 커밋될 때까지 메시지 또는 세그먼트를 노드에서 종료하거나(해당 목적지가 리모트인 경우) 검색하는 작업을 시작할 수 없습니다. 전체 그룹을 넣는 데 시간이 오래 걸리거나 노드에서 큐 공간이 제한된 경우 이 작업이 필요하지 않을 수 있습니다. 이를 극복하려면 그룹을 여러 작업 단위에 넣으십시오.

그룹을 여러 작업 단위 안에 넣으면 넣기 애플리케이션이 실패할 때에도 그룹의 일부가 커밋될 수 있습니다. 따라서 애플리케이션은 개별 작업 단위로 커밋된 상태 정보를 저장해야 합니다. 이 정보는 재시작 후 완료되지 않은 그룹을 재개하기 위해 사용할 수 있습니다. 이 정보를 기록하는 가장 단순한 위치는 **STATUS** 큐입니다. 완료 그룹 넣기가 완료되면 **STATUS** 큐는 비어 있습니다.

세그먼트가 포함된 경우에도 논리는 같습니다. 이 경우 **StatusInfo**에는 *Offset*이 포함되어야 합니다.

여러 작업 단위에 그룹을 넣는 예는 다음과 같습니다.

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

/* First UOW */

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Next and subsequent UOWs */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Last UOW */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
MQCMIT
```

모든 작업 단위가 커밋되면, 전체 그룹 넣기가 완료되고 **STATUS** 큐는 비어 있습니다. 그렇지 않으면, 상태 정보에 표시된 지점에서 그룹을 재개해야 합니다. **MQPMO_LOGICAL_ORDER**는 첫 번째 넣기에서 사용할 수 없지만 그 후에는 사용할 수 있습니다.

재시작 처리는 다음과 같습니다.

```
MQGET (StatusInfo from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
    /* Proceed to normal processing */
    ...
else
    /* Group was terminated prematurely */
    Set GroupId, MsgSeqNumber in MQMD to values from Status message
    PMO.Options = MQPMO_SYNCPOINT
    MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
```

```

/* Now normal processing is resumed.
   Assume this is not the last message */
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

```

가져오기 애플리케이션에서 전체 그룹이 도착하기 전에 그룹의 메시지 처리를 시작할 수도 있습니다. 이렇게 하면 그룹 내 메시지에 대한 응답 시간이 개선되고 전체 그룹에 대한 스토리지가 필요하지 않습니다. 이점을 실현하려면 각 메시지 그룹에 대해 여러 작업 단위를 사용하십시오. 복구하려면 작업 단위 내에서 각 메시지를 검색해야 합니다.

이 경우, 해당 넣기 애플리케이션과 마찬가지로 각 작업 단위가 커밋될 때 상태 정보가 자동으로 임의의 위치에 기록되어야 합니다. 게다가 이 정보를 기록하는 가장 단순한 위치는 STATUS 큐입니다. 완료 그룹 처리가 완료되면 STATUS 큐는 비어 있습니다.

참고: 중간 작업 단위의 경우, 각 작업 단위에 대해 새 완료 메시지를 넣지 않고 상태 큐에 대한 개별 MQPUT이 메시지의 세그먼트가 되도록 지정(즉, MQMF_SEGMENT 플래그를 설정)하여 STATUS 큐에서 MQGET 호출을 방지할 수 있습니다. 마지막 작업 단위에서는 MQMF_LAST_SEGMENT를 지정하는 상태 큐에 최종 세그먼트를 넣은 후, MQGMO_COMPLETE_MSG를 지정하는 MQGET으로 상태 정보를 지웁니다.

재시작 처리 동안은 단일 MQGET을 사용하여 가능한 상태 메시지를 가져오지 말고 마지막 세그먼트에 도달할 때까지(즉, 추가 세그먼트가 리턴되지 않을 때까지) MQGMO_LOGICAL_ORDER로 상태 큐를 찾아보십시오. 재시작 후 첫 번째 작업 단위에서 상태 세그먼트를 넣을 때에도 오프셋을 명시적으로 지정하십시오.

다음 예제에서는 애플리케이션의 버퍼가 항상 전체 메시지를 보유할 만큼 크다고 가정하고 메시지가 세그먼트되었는지 여부에 관계없이 그룹 내의 메시지만 고려합니다. 따라서 MQGMO_COMPLETE_MSG가 각 MQGET에 지정됩니다. 세그먼트가 포함된 경우에도 동일한 원리가 적용됩니다. (이 경우 StatusInfo에 *Offset*이 포함되어야 합니다.)

단순성 보장을 위해 단일 UOW 내에서 최대 4개의 메시지가 검색된다고 가정합니다.

```

msgs = 0      /* Counts messages retrieved within UOW */
/* Should be no status message at this point */

/* Retrieve remaining messages in the group */
do while ( GroupStatus == MQGS_MSG_IN_GROUP )

    /* Process up to 4 messages in the group */
    GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
                | MQGMO_LOGICAL_ORDER
    do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
        MQGET
        msgs = msgs + 1
        /* Process this message */
        ...
    /* end while

    /* Have retrieved last message or 4 messages */
    /* Update status message if not last in group */
    MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
    if ( GroupStatus == MQGS_MSG_IN_GROUP )
        StatusInfo = GroupId,MsgSeqNumber from MQMD
        MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
    MQCMIT
    msgs = 0
/* end while

if ( msgs > 0 )
    /* Come here if there was only 1 message in the group */
    MQCMIT

```

모든 작업 단위가 커밋되면, 전체 그룹 검색이 완료되고 STATUS 큐는 비어 있습니다. 그렇지 않으면, 상태 정보에 표시된 지점에서 그룹을 재개해야 합니다. MQGMO_LOGICAL_ORDER는 첫 번째 검색에서 사용할 수 없지만 그 이후에는 사용할 수 있습니다.

재시작 처리는 다음과 같습니다.

```

MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)

```

```

/* Proceed to normal processing */
...

else
/* Group was terminated prematurely */
/* The next message on the group must be retrieved by matching
the sequence number and group ID with those retrieved from the
status information. */
GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID | MQMO_MATCH_MSG_SEQ_NUMBER,
      MQMD.GroupId      = value from Status message,
      MQMD.MsgSeqNumber = value from Status message plus 1
msgs = 1
/* Process this message */
...

/* Now normal processing is resumed */
/* Retrieve remaining messages in the group */
do while ( GroupStatus == MQGS_MSG_IN_GROUP )

    /* Process up to 4 messages in the group */
    GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
                | MQGMO_LOGICAL_ORDER
    do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
        MQGET
        msgs = msgs + 1
        /* Process this message */
        ...

    /* Have retrieved last message or 4 messages */
    /* Update status message if not last in group */
    MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
    if ( GroupStatus == MQGS_MSG_IN_GROUP )
        StatusInfo = GroupId,MsgSeqNumber from MQMD
        MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
    MQCMIT
    msgs = 0

```

특정 메시지 가져오기

큐에서 특정 메시지를 가져오는 다양한 방법이 있습니다. 즉, `MsgId`와 `CorrelId`에서 선택, `GroupId`, `MsgSeqNumber` 및 `Offset`에서 선택 및 `MsgToken`에서 선택하는 방법입니다. 또한 큐를 열 때 선택 문자열을 사용할 수 있습니다.

큐에서 특정 메시지를 가져오려면, MQMD 구조의 `MsgId` 및 `CorrelId` 필드를 사용하십시오. 그러나 애플리케이션은 사용자가 지정한 값이 고유 메시지를 식별하지 못하도록 이러한 필드를 명시적으로 설정할 수 있습니다. 801 페이지의 표 133은 이러한 필드의 가능한 설정에 대해 검색되는 메시지를 보여줍니다. MQGET 호출의 **GetMsgOpts** 매개변수에 MQGMO_MSG_UNDER_CURSOR를 지정하는 경우 이러한 필드가 입력에서 무시됩니다.

표 133. 메시지 및 상관 ID 사용		
검색 대상 ...	MsgId	CorrelId
큐의 첫 번째 메시지	MQMI_NONE	MQCI_NONE
MsgId와 일치하는 첫 번째 메시지	0이 아님	MQCI_NONE
CorrelId와 일치하는 첫 번째 메시지	MQMI_NONE	0이 아님
MsgId 및 CorrelId 둘 모두와 일치하는 첫 번째 메시지	0이 아님	0이 아님

각 사례에서 첫 번째는 선택 기준을 충족하는 첫 번째 메시지를 의미합니다(단, 선택 기준을 충족하는 순서에서 다음 메시지를 의미하는 MQGMO_BROWSE_NEXT가 지정된 경우는 제외).

리턴 시, MQGET 호출은 `MsgId` 및 `CorrelId` 필드를 리턴된 메시지의 메시지 및 상관 ID(있는 경우)로 설정합니다.

MQMD 구조의 `Version` 필드를 2로 설정하면, `GroupId`, `MsgSeqNumber` 및 `Offset` 필드를 사용할 수 있습니다. 802 페이지의 표 134은 이러한 필드의 가능한 설정에 대해 검색되는 메시지를 보여줍니다.

표 134. 그룹 ID 사용	
검색 대상 ...	일치 옵션
큐의 첫 번째 메시지	MQMO_NONE
MsgId와 일치하는 첫 번째 메시지	MQMO_MATCH_MSG_ID
CorrelId와 일치하는 첫 번째 메시지	MQMO_MATCH_CORREL_ID
GroupId와 일치하는 첫 번째 메시지	MQMO_MATCH_GROUP_ID
MsgSeqNumber와 일치하는 첫 번째 메시지	MQMO_MATCH_MSG_SEQ_NUMBER
MsgToken와 일치하는 첫 번째 메시지	MQMO_MATCH_MSG_TOKEN
Offset와 일치하는 첫 번째 메시지	MQMO_MATCH_OFFSET

참고:

1. MQMO_MATCH_XXX는 MQMD 구조의 XXX 필드가 일치되는 값으로 설정된 것을 암시합니다.
2. MQMO 플래그는 결합에서 사용될 수 있습니다. 예를 들어, MQMO_MATCH_GROUP_ID, MQMO_MATCH_MSG_SEQ_NUMBER 및 MQMO_MATCH_OFFSET을 함께 사용하여 GroupId, MsgSeqNumber 및 Offset 필드로 식별된 세그먼트를 제공할 수 있습니다.
3. MQGMO_LOGICAL_ORDER를 지정하면, 큐 핸들에 대해 제어되는 상태 정보에 따라 옵션이 달라지기 때문에 검색하려는 메시지가 영향을 받습니다. 이에 대한 정보는 [791 페이지의 『논리적 및 물리적 정렬』](#) 및 [옵션](#)을 참조하십시오.

MQGET 호출은 대개 큐에서 첫번째 메시지를 검색합니다. MQGET 호출을 사용할 때 특정 메시지를 지정하는 경우, 큐 관리자는 해당 메시지를 찾을 때까지 큐를 검색해야 합니다. 이는 애플리케이션의 성능에 영향을 미칠 수 있습니다.

버전 2 이상의 MQGMO 구조를 사용 중이고 MQMO_MATCH_MSG_ID 또는 MQMO_MATCH_CORREL_ID 플래그를 지정하지 않은 경우, MQGET 사이에 MsgId 또는 CorrelId 필드를 재설정할 필요가 없습니다.

 IBM MQ for z/OS에서 큐 속성 IndexType을 사용하여 큐에서 MQGET 조작의 속도를 올릴 수 있습니다. 자세한 정보는 [806 페이지의 『색인 유형』](#)의 내용을 참조하십시오.

MQGMO 구조에서 해당 MsgToken 및 MatchOption MQMO_MATCH_MSG_TOKEN을 지정하여 큐로부터 특정 메시지를 가져올 수 있습니다. MsgToken은 원래 해당 메시지를 큐에 넣은 MQPUT 호출 또는 이전 MQGET 조작에 의해 리턴되고, 큐 관리자가 재시작되지 않으면 상수로 유지됩니다.

큐에 있는 메시지의 서브세트에만 관심이 있는 경우, MQOPEN 또는 MQSUB 호출에 선택 문자열을 사용하여 처리할 메시지를 지정할 수 있습니다. 그러면 MQGET은 해당 선택 문자열을 충족하는 다음 메시지를 검색합니다. 선택 문자열에 대한 자세한 정보는 [36 페이지의 『선택자』](#)의 내용을 참조하십시오.

비지속 메시지의 성능 개선

클라이언트는 서버의 메시지가 필요할 때 서버에 요청을 송신합니다. 이용하는 각 메시지에 대해 별도의 요청을 송신합니다. 이러한 요청 메시지 송신을 방지하여 클라이언트의 비지속 메시지 이용 성능을 향상시키려는 경우 클라이언트가 미리 읽기를 사용하도록 구성할 수 있습니다. 미리 읽기를 사용하면 애플리케이션이 요청하지 않아도 메시지를 클라이언트에 송신할 수 있습니다.

미리 읽기가 설정되어 있으면, 메시지는 미리 읽기 버퍼라고 하는 클라이언트의 메모리 버퍼로 송신됩니다. 클라이언트에는 미리 읽기가 사용 가능한 상태로 열려 있는 각 큐에 대한 미리 읽기 버퍼가 있습니다. 미리 읽기 버퍼의 메시지는 지속되지 않습니다. 클라이언트는 이용한 데이터 용량에 대한 정보로 서버를 주기적으로 업데이트합니다.

MQOO_READ_AHEAD와 함께 MQOPEN을 호출할 경우 특정 조건이 충족되면 IBM MQ 클라이언트에서는 미리 읽기만 가능합니다. 이러한 조건은 다음과 같습니다.

- 클라이언트 애플리케이션이 컴파일되고 스레드된 IBM MQ MQI 클라이언트 라이브러리에 대해 링크되어야 합니다.
- 클라이언트 채널이 TCP/IP 프로토콜을 사용해야 합니다.

- 채널이 클라이언트 및 서버 채널 정의 모두에서 0이 아닌 SharingConversations(SHARECNV) 설정을 사용해야 합니다.

미리 읽기를 사용하면 클라이언트 애플리케이션에서 비지속 메시지를 이용할 때 성능을 향상시킬 수 있습니다. 이러한 성능 개선은 MQI 및 JMS 애플리케이션 모두에서 가능합니다. MQGET을 사용하거나 비동기로 이용하는 클라이언트 애플리케이션은 비지속 메시지 이용 시 성능 개선의 이점이 있습니다.

모든 옵션이 미리 읽기와 함께 사용하도록 지원되는 것이 아니고 일부 옵션은 미리 읽기가 사용 가능할 때 MQGET 호출 간에 일치해야 하므로 모든 클라이언트 애플리케이션 디자인이 미리 읽기 사용에 적합한 것은 아닙니다. 클라이언트가 MQGET 호출 간에 해당 선택 기준을 대체하는 경우, 미리 읽기 버퍼에 저장될 메시지가 클라이언트 미리 읽기 버퍼에 스트랜드 상태로 유지됩니다.

이전 선택 기준에 따라 스트랜드 메시지의 백로그가 더 이상 필요하지 않은 경우, 클라이언트에서 이러한 메시지를 자동으로 영구 제거하도록 구성 가능한 영구 제거 간격을 클라이언트에 설정할 수 있습니다. 영구 제거 간격은 클라이언트에 의해 판별된 한 그룹의 미리 읽기 성능 조정 옵션 중 하나입니다. 이러한 옵션은 사용자 요구사항에 맞게 조정할 수 있습니다.

클라이언트 애플리케이션이 재시작되면, 미리 읽기 버퍼의 메시지가 손실될 수 있습니다. 반대로, 미리 읽기 버퍼로 이동된 메시지는 기본 큐에서 삭제할 수 있지만 이 결과로 버퍼에서 해당 메시지가 제거되지는 않습니다. 따라서 미리 읽기를 사용하여 MQGET 호출을 수행하면 더 이상 존재하지 않는 메시지가 리턴될 수 있습니다.

미리 읽기는 클라이언트 바인딩에 대해서만 수행됩니다. 다른 모든 바인딩에 대한 속성은 무시됩니다.

미리 읽기는 트리거에 영향을 미치지 않습니다. 클라이언트가 메시지를 미리 읽을 때에는 트리거 메시지가 생성되지 않습니다. 미리 읽기가 사용으로 설정되면 회계 및 통계 정보를 생성하지 않습니다.

발행/구독 메시징에서 미리 읽기 사용

구독 애플리케이션에서 발행물이 송신되는 목적지 큐를 지정하면 지정된 큐의 DEFREADA 값이 기본 미리 읽기 값으로 사용됩니다.

IBM MQ가 발행물이 송신되는 목적지를 관리하도록 구독 애플리케이션에서 요청하면, 사전정의된 모델 큐를 기반으로 관리되는 큐가 동적 큐로 작성됩니다. 기본 미리 읽기 값으로 사용되는 모델 큐의 DEFREADA 값입니다. 이 토픽 또는 상위 토픽에 대해 모델 큐가 정의되지 않으면 기본 모델 큐 SYSTEM.DURABLE.PUBLICATIONS.MODEL 또는 SYSTEM.NONDURABLE.PUBLICATIONS.MODEL이 사용됩니다.

관련 개념

805 페이지의 『AIX의 비지속 메시지에 대한 성능 조정』

AIX V5.3 이상을 사용 중인 경우 비지속 메시지에 대한 전체 성능을 사용하려면 성능 조정 매개변수 설정을 고려하십시오.

관련 태스크

804 페이지의 『미리 읽기 사용 및 사용 안함』

기본적으로 미리 읽기는 사용 불가능합니다. 큐 또는 애플리케이션 레벨에서 미리 읽기를 사용할 수 있습니다.

관련 참조

803 페이지의 『MQGET 옵션 및 미리 읽기』

미리 읽기를 사용할 때 모든 MQGET 옵션이 지원되는 것은 아닙니다. 일부 옵션은 MQGET 호출 간에 일치해야 합니다.

MQGET 옵션 및 미리 읽기

미리 읽기를 사용할 때 모든 MQGET 옵션이 지원되는 것은 아닙니다. 일부 옵션은 MQGET 호출 간에 일치해야 합니다.

MQOO_READ_AHEAD와 함께 MQOPEN을 호출할 경우 특정 조건이 충족되면 IBM MQ 클라이언트에서는 미리 읽기만 가능합니다. 이러한 조건은 다음과 같습니다.

- 클라이언트 애플리케이션이 컴파일되고 스투드된 IBM MQ MQI 클라이언트 라이브러리에 대해 링크되어야 합니다.
- 클라이언트 채널이 TCP/IP 프로토콜을 사용해야 합니다.
- 채널이 클라이언트 및 서버 채널 정의 모두에서 0이 아닌 SharingConversations(SHARECNV) 설정을 사용해야 합니다.

다음 표에서는 미리 읽기와 함께 사용하도록 지원되는 옵션과, 해당 옵션이 MQGET 호출 간에 대체될 수 있는지를 표시합니다.

표 135. MQGET 옵션 및 미리 읽기			
MQGET 값과 옵션	미리 읽기를 사용하는 경우 허용되며 MQGET 호출 간에 대체될 수 있음 ⁵	미리 읽기를 사용하는 경우 허용되지만 MQGET 호출 간에 대체될 수 없음 ¹	미리 읽기를 사용하는 경우 허용되지 않는 MQGET 옵션 ²
MQGET MQMD 값	MsgId ³ CorrelId ³	Encoding CodedCharSetId	
MQGET MQGMO 옵션	- MQGMO_NO_WAIT - MQGMO_BROWSE_MESSAGE_UNDER_CURSOR - MQGMO_BROWSE_FIRST - MQGMO_BROWSE_NEXT - MQGMO_FAIL_IF QUIESCING	- MQGMO_SYNCPOINT_IF_PERSISTENT - MQGMO_NO_SYNCPOINT - MQGMO_ACCEPT_TRUNCATED_MSG - MQGMO_CONVERT	- MQGMO_SET_SIGNAL - MQGMO_SYNCPOINT - MQGMO_MARK_SKIP_BACKOUT - MQGMO_MSG_UNDER_CURSOR⁴ - MQGMO_LOCK - MQGMO_UNLOCK - MQGMO_LOGICAL_ORDER - MQGMO_COMPLETE_MSG - MQGMO_ALL_MSGS_AVAILABLE - MQGMO_ALL_SEGMENTS_AVAILABLE

참고:

1. MQGET 호출 간에 이러한 옵션이 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 애플리케이션이 MQGET 호출 간에 MsgId 및 CorrelId 값을 대체하는 경우, 이전 값을 가진 메시지는 이미 클라이언트에 송신되었을 수 있으며 이용되거나 자동으로 영구 제거될 때까지 클라이언트 미리 읽기 버퍼에 남게 됩니다.
4. MQGMO_MSG_UNDER_CURSOR는 미리 읽기에 사용할 수 없습니다. 미리 읽기는 큐를 열 때 MQOO_BROWSE와 MQOO_INPUT_SHARED 또는 MQOO_INPUT_EXCLUSIVE 옵션 중 하나가 모두 지정되면 사용 안함으로 설정됩니다.
5. 미리 읽기를 사용으로 설정하면, 첫 번째 MQGET이 메시지를 찾아볼지 큐에서 가져올지 여부를 판별합니다. 그런 다음 클라이언트 애플리케이션이 변경된 옵션(예: 초기 가져오기 후 찾아보기 시도 또는 초기 찾아보기 후 가져오기 시도)과 함께 MQGET을 사용하는 경우 MQRC_OPTIONS_CHANGED 이유 코드가 리턴됩니다.

클라이언트가 MQGET 호출 간에 해당 선택 기준을 대체하는 경우, 미리 읽기 버퍼에 저장되고 초기 선택 기준과 일치하는 메시지가 클라이언트 애플리케이션에서 이용되지 않고 클라이언트 미리 읽기 버퍼에 스트랜드 상태로 남습니다. 클라이언트 미리 읽기 버퍼에 여러 스트랜드 메시지가 포함되어 있는 경우에는 미리 읽기와 연관된 이점이 손실되므로, 이용된 각 메시지에 대해 서버에 별도 요청이 필요합니다. 미리 읽기가 효율적으로 사용되는지 여부를 판별하기 위해 연결 상태 매개변수 READA를 사용할 수 있습니다.

첫 번째 MQGET 호출에 지정된 호환되지 않는 옵션으로 인해 애플리케이션의 요청이 있을 때 미리 읽기가 금지될 수 있습니다. 이런 상황에서 연결 상태는 미리 읽기가 금지된 것으로 표시됩니다.

이러한 MQGET 제한사항 때문에 클라이언트 애플리케이션 디자인이 미리 읽기에 적합하지 않다고 결정한 경우 MQOPEN 옵션 MQOO_READ_AHEAD_NO를 지정하십시오. 또는 열려는 큐의 기본 미리 읽기 값이 NO 또는 DISABLED로 대체되도록 설정하십시오.

미리 읽기 사용 및 사용 안함

기본적으로 미리 읽기는 사용 불가능합니다. 큐 또는 애플리케이션 레벨에서 미리 읽기를 사용할 수 있습니다.

이 태스크 정보

MQOO_READ_AHEAD와 함께 MQOPEN을 호출할 경우 특정 조건이 충족되면 IBM MQ 클라이언트에서는 미리 읽기만 가능합니다. 이러한 조건은 다음과 같습니다.

- 클라이언트 애플리케이션이 컴파일되고 스레드된 IBM MQ MQI 클라이언트 라이브러리에 대해 링크되어야 합니다.
- 클라이언트 채널이 TCP/IP 프로토콜을 사용해야 합니다.
- 채널이 클라이언트 및 서버 채널 정의 모두에서 0이 아닌 SharingConversations(SHARECNV) 설정을 사용해야 합니다.

미리 읽기를 사용하려면 다음을 수행하십시오.

- 큐 레벨에서 미리 읽기를 구성하려면 큐 속성 DEFREADA를 YES로 설정하십시오.
- 애플리케이션 레벨에서 미리 읽기를 구성하려면 다음을 수행하십시오.
 - 가능한 경우 미리 읽기를 사용하려면 MQOPEN 함수 호출에 MQOO_READ_AHEAD 옵션을 사용하십시오. DEFREADA 큐 속성이 DISABLED로 설정된 경우에 클라이언트 애플리케이션이 미리 읽기를 사용하는 것은 불가능합니다.
 - 큐에서 미리 읽기가 사용으로 설정된 경우에만 미리 읽기를 사용하려면 MQOPEN 함수 호출에 MQOO_READ_AHEAD_AS_Q_DEF 옵션을 사용하십시오.

클라이언트 애플리케이션 디자인이 미리 읽기에 적합하지 않으면 다음 방법으로 해당 디자인을 사용 안함으로 설정할 수 있습니다.

- 큐 레벨에서 큐 속성 DEFREADA를 NO(클라이언트 애플리케이션에서 요청하지 않는 한 미리 읽기를 사용하지 않으려는 경우) 또는 DISABLED(클라이언트 애플리케이션에 미리 읽기가 필요한지 여부에 관계없이 미리 읽기를 사용하지 않으려는 경우)로 설정하는 방법
- 애플리케이션 레벨에서 MQOPEN 함수 호출의 MQOO_NO_READ_AHEAD 옵션을 사용하는 방법

두 가지 MQCLOSE 옵션을 사용하여 큐가 처리완료된 경우 미리 읽기 버퍼에 저장되는 모든 메시지에서 발생한 내용을 구성할 수 있습니다.

- 미리 읽기 버퍼의 메시지를 제거하려면 MQCO_IMMEDIATE를 사용하십시오.
- 큐가 처리완료되기 전에 미리 읽기 버퍼의 메시지가 애플리케이션에서 이용되었는지 확인하려면 MQCO_QUIESCE를 사용하십시오. MQCLOSE가 MQCO_QUIESCE와 함께 발행되고 미리 읽기 버퍼에 메시지가 남아 있는 경우 MQRC_READ_AHEAD_MSGS가 MQCC_WARNING으로 리턴됩니다.

AIX의 비지속 메시지에 대한 성능 조정

AIX V5.3 이상을 사용 중인 경우 비지속 메시지에 대한 전체 성능을 사용하려면 성능 조정 매개변수 설정을 고려하십시오.

즉시 적용되도록 성능 조정 매개변수를 설정하려면 다음 명령을 루트 사용자로 실행하십시오.

```
/usr/sbin/ios -o j2_nPagesPerWriteBehindCluster=0
```

즉시 적용되고 시스템 다시 시작을 통해 지속되도록 성능 조정 매개변수를 설정하려면 루트 사용자로 다음 명령을 실행하십시오.

```
/usr/sbin/ios -p -o j2_nPagesPerWriteBehindCluster=0
```

보통은 비지속 메시지가 메모리에만 보관되지만, AIX에서는 비지속 메시지가 디스크에 기록되도록 스케줄할 수 있는 경우도 있습니다. 디스크에 기록되도록 스케줄된 메시지는 디스크 쓰기가 완료될 때까지 MQGET에서 사용 불가능합니다. 제안된 성능 조정 명령은 이 임계값을 변경합니다. 16킬로바이트의 데이터가 큐에 대기될 때 메시지를 디스크에 쓰도록 스케줄링하지 않고, 시스템의 실제 스토리지가 거의 찰 때만 디스크에 쓰기가 발생합니다. 이는 글로벌 대체이며 다른 소프트웨어 컴포넌트에 영향을 미칠 수 있습니다.

AIX에서 멀티스레드 애플리케이션을 사용할 때 및 특히 다중 프로세서가 있는 시스템에서 실행 중인 경우에는, 성능 개선과 보다 확실한 스케줄 설정을 위해 애플리케이션을 시작하기 전 환경에 AIXTHREAD_SCOPE=S를 설정하거나 mqm ID .profile에 AIXTHREAD_SCOPE=S를 설정할 것을 강력히 권장합니다. 예를 들면, 다음과 같습니다.

```
export AIXTHREAD_SCOPE=S
```

AIXTHREAD_SCOPE=S를 설정하면 기본 속성으로 작성된 사용자 스레드가 시스템 전체 경합 범위에 배치됩니다. 시스템 전체 경합 범위를 사용하여 사용자 스레드가 작성되면 해당 스레드는 커널 스레드에 바인드되고 커널에 의해 스케줄됩니다. 기본 커널 스레드는 다른 사용자 스레드와 공유되지 않습니다.

파일 디스크립터

에이전트 프로세스 같은 멀티스레드 프로세스를 실행할 때는 파일 디스크립터의 소프트 한계에 이를 수 있습니다. 이 한계로 인해 IBM MQ 이유 코드 MQRC_UNEXPECTED_ERROR (2195)가 수신되고 충분한 파일 디스크립터가 있으면 IBM MQ FFST™ 파일이 제공됩니다.

이 문제점이 발생하지 않도록 파일 디스크립터 수에 대한 프로세스 한계를 늘릴 수 있습니다. 이를 수행하려면, mqm 사용자 ID에 대해서나 기본 스탠자에서 /etc/security/limits의 nofiles 속성을 10,000으로 대체하십시오.

시스템 자원 한계

명령 프롬프트에서 다음 명령을 사용하여 데이터 세그먼트 및 스택 세그먼트에 대한 시스템 자원 한계를 무제한으로 설정하십시오.

```
ulimit -d unlimited
ulimit -s unlimited
```

색인 유형

큐 속성 *IndexType*은 큐에서 MQGET 조작의 속도를 올리기 위해 큐 관리자가 유지보수하는 색인의 유형을 지정합니다.

참고: IBM MQ for z/OS에서만 지원됩니다.

5가지 옵션이 있습니다.

값	설명
NONE	색인이 유지보수되지 않습니다. 순차적으로 메시지를 검색할 때 이 옵션을 사용하십시오(791 페이지의 『Priority』 참조).
GROUPID	그룹 ID의 색인이 유지보수됩니다. 메시지 그룹의 논리 순서를 지정할 경우 이 색인 유형을 사용해야 합니다(791 페이지의 『논리적 및 물리적 정렬』 참조).
MSGID	메시지 ID의 색인이 유지보수됩니다. MQGET 호출에 대한 선택 기준으로 <i>MsgId</i> 필드를 사용하여 메시지를 검색할 때 이 옵션을 사용하십시오(801 페이지의 『특정 메시지 가져오기』 참조).
MSGTOKEN	메시지 토큰의 색인이 유지보수됩니다.
CORRELID	상관 ID의 색인이 유지보수됩니다. MQGET 호출에 대한 선택 기준으로 <i>CorrelId</i> 필드를 사용하여 메시지를 검색할 때 이 옵션을 사용하십시오(801 페이지의 『특정 메시지 가져오기』 참조).

참고:

1. MSGID 옵션 또는 CORRELID 옵션을 사용하여 색인화할 경우, MQMD에서 상대 **MsgId** 또는 **CorrelId** 매개변수를 설정하십시오. 두 옵션을 모두 설정하는 것은 유익하지 않습니다.
2. 큐가 다음의 모든 조건과 일치하면 색인 메커니즘을 사용하여 메시지를 찾습니다.
 - 색인 유형이 MSGID, CORRELID 또는 GROUPID임
 - 동일 유형의 ID를 사용하여 찾아봅니다.
 - 메시지의 우선순위가 하나뿐입니다.
3. 수천 개의 메시지를 포함하는 (*MsgId* 또는 *CorrelId*로 색인화된) 큐는 재시작 시간에 영향을 주므로 이를 방지합니다. (재시작 시 삭제되므로 비지속 메시지는 적용되지 않습니다.)
4. MSGTOKEN은 z/OS 워크로드 관리자에서 관리되는 큐를 정의하는 데 사용됩니다.

IndexType 속성에 대한 전체 설명은 **IndexType**을 참조하십시오. **IndexType** 속성에 대한 추가 정보는 67 페이지의 『z/OS 애플리케이션에 대한 설계 및 성능 고려사항』의 내용을 참조하십시오.

4MB 이상의 메시지 처리

메시지가 애플리케이션, 큐 또는 큐 관리자에 비해 너무 클 수 있습니다. 환경에 따라 IBM MQ은 4MB보다 큰 메시지를 처리하는 여러 가지 방법을 제공합니다. 최대 메시지 크기는 100MB입니다.

이 태스크 정보

MaxMsgLength 대기열 및 대기열 관리자 속성을 늘려 대기열을 사용하는 메시지의 크기를 반영할 수 있으며, 최대 메시지 크기는 100MB까지입니다. 대용량 메시지 핸들링 방법은 쉽고 편리합니다. 그러나 사용하기 전에 다음 요인을 고려하십시오.

- 큐 관리자 사이에 균일성이 감소합니다. 메시지 데이터의 최대 크기는 메시지가 놓일 각 대기열(전송 대기열 포함)의 **MaxMsgLength**에 따라 결정됩니다. 이 값은 특히 전송 대기열의 경우 대기열 관리자의 **MaxMsgLength**로 기본 설정되는 경우가 많습니다. 이렇게 되면 리모트 큐 관리자로 이동할 때 메시지가 너무 큰지 여부를 예측하기가 어렵습니다.
- 시스템 자원 사용량이 증가합니다. 예를 들어 애플리케이션에는 더 큰 버퍼가 필요하고 일부 플랫폼에서는 공유 스토리지의 사용량이 증가할 수 있습니다. 대기열 저장소는 실제로 대용량 메시지에 필요한 경우에만 영향을 받습니다.
- 채널 배치 작업이 영향을 받습니다. 대용량 메시지는 배치 수로는 여전히 한 개의 메시지로 계수되지만 전송하는 데 시간이 더 필요하기 때문에 다른 메시지에 대한 응답 시간을 증가시킵니다.

Multi IBM MQ for Multiplatforms에서는 최대 메시지 길이 100MB까지 대용량 메시지를 처리할 수 있는 두 가지 추가 옵션이 있습니다:

- 메시지 세분화: 각각의 큰 메시지를 여러 개의 작은 청크로 나누어 전송하는 방식입니다. 애플리케이션 또는 대기열 관리자가 메시지를 세그먼트화할 수 있습니다.
- 참조 메시지는 메시지에 대한 포인터가 실제 메시지를 받는 방법을 알려주는 지침과 함께 수신자에게 전송되는 메시지입니다.

자세한 내용은 이 주제 끝에 있는 링크를 참조하세요.

프로시저

1. **MaxMsgLength** 속성 중 하나 또는 둘 모두를 변경합니다.

Multi IBM MQ for Multiplatforms에서 대기열 속성을 0~100MB 범위로 설정하고 대기열 관리자 속성을 32768바이트~100MB 범위로 설정합니다.

z/OS IBM MQ for z/OS에서 대기열 속성을 0~100MB 범위로 설정합니다. **MaxMsgLength** 대기열 관리자 속성은 100MB로 하드 코딩되어 있습니다.

2. 애플리케이션과 채널을 다시 시작하여 변경 사항이 적용되도록 합니다.

이렇게 변경한 후에는 메시지 길이가 대기열 및 대기열 관리자 **MaxMsgLength** 속성 모두보다 작거나 같아야 합니다. 그러나 기존 메시지는 더 길어질 수 있습니다.

메시지가 대기열에 비해 너무 크면 MQRC_MSG_TOO_BIG_FOR_Q 메시지가 반환됩니다. 메시지가 큐 매니저에 비해 너무 크면 MQRC_MSG_TOO_BIG_FOR_Q_MGR 메시지가 반환됩니다.

Multi 메시지 세그먼트화

특정 상황에서는 대용량 메시지(최대 100MB)를 여러 개의 작은 조각으로 분할하여 전송할 경우 성공률이 높아질 수 있습니다. 메시지는 애플리케이션 또는 큐 관리자에 의해 세분화될 수 있습니다. 이 기능은 IBM MQ classes for JMS 사용하는 애플리케이션이나 IBM MQmessaging REST API에서 ~에 IBM MQ for z/OS, 또는 지원되지 않습니다.

설명된 대로 최대 메시지 길이를 늘리는 807 페이지의 『4MB 이상의 메시지 처리』 것은 몇 가지 부정적인 영향을 미칩니다. 또한 메시지가 여전히 큐 또는 큐 관리자에 비해 너무 클 수 있습니다. 이러한 경우 메시지를 분할할 수 있습니다. 세그먼트에 대한 정보는 48 페이지의 『메시지 그룹』의 내용을 참조하십시오.

다음 절에서는 메시지를 세그먼트화하는 일반적인 용도에 대해 살펴봅니다. 넣기 및 가져오기 후 삭제의 경우, MQPUT 또는 MQGET 호출이 항상 작업 단위 내에서 작동한다고 가정합니다. 항상 이 기술을 사용하여 네트워크에 존재하는 미완료 그룹의 가능성을 줄이는 방안을 고려하십시오. 큐 관리자에서 1단계 커미트를 가정하지만, 다른 조정 기술도 똑같이 유효합니다.

또한 가져오기 애플리케이션에서는 여러 서버가 동일한 큐를 처리할 경우, (MQGMO_ALL_MSGS_AVAILABLE 또는 MQGMO_ALL_SEGMENTS_AVAILABLE이 이미 지정되었기 때문에) 하나의 서버에서 이용할 수 있을 것으로 예상되는 메시지 또는 세그먼트를 찾는 데 실패하지 않도록 각 서버가 유사한 코드를 실행한다고 가정합니다.



주의: 토픽에 메시지를 전송할 때(또는 토픽 별칭에 메시지를 배치할 때) 퍼블리시/서브스크라이브 방식을 사용할 경우 메시지 그룹화 및 분할이 허용되지 않습니다.

구독은 게시 활동과 독립적으로 생성 및 제거될 수 있으므로, 구독자가 메시지 그룹 전체 또는 메시지의 모든 세그먼트를 수신할 것이라고 보장할 수 없습니다. 자세한 내용은 [RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP](#)을 참조하십시오.

작업 단위에 걸친 세그먼트된 메시지 넣기 및 가져오기

799 페이지의 『작업 단위에 걸친 그룹 넣기 및 가져오기』와 유사한 방식으로 작업 단위에 걸친 세그먼트된 메시지를 넣고 가져올 수 있습니다.

하지만 글로벌 작업 단위에 세그먼트된 메시지를 넣거나 가져올 수는 없습니다.

Multi

큐 관리자의 세그먼트화 및 리어셈블리

이는 한 애플리케이션이 다른 애플리케이션에 의해 검색될 메시지를 넣는 가장 단순한 시나리오입니다. 메시지는 대용량일 수 있습니다. 즉, 메시지가 넣기 또는 가져오기 애플리케이션이 단일 버퍼로 핸들링할 수 없을 정도로 크다는 의미가 아니라, 메시지를 넣을 큐 또는 큐 관리자에 비해 너무 크다는 의미입니다.

이러한 애플리케이션에 필요한 유일한 변경은 넣기 애플리케이션이 필요한 경우 세그먼트화를 수행할 수 있도록 큐 관리자에 권한을 부여하고

```
PMO.Options = (existing options)
MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED
MD.Version = MQMD_VERSION_2
memcpy(MD.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH)
MQPUT
```

가져오기 애플리케이션이 세그먼트된 메시지를 리어셈블하도록 큐 관리자에 요청하는 것입니다.

```
GMO.Options = MQGMO_COMPLETE_MSG | (existing options)
MQGET
```

이 단순한 시나리오에서는 큐 관리자가 각 메시지에 대해 고유 그룹 ID를 생성할 수 있도록 애플리케이션이 MQPUT 호출 전에 GroupId 필드를 MQGI_NONE으로 재설정해야 합니다. 이렇게 하지 않으면, 관련 없는 메시지가 동일한 그룹 ID를 가질 수 있으므로 후속 처리가 올바르게 작동할 수 없습니다.

애플리케이션 버퍼는 리어셈블된 메시지를 포함할 수 있을 만큼 커야 합니다 (MQGMO_ACCEPT_TRUNCATED_MSG 옵션을 포함하지 않는 경우).

메시지 세그먼트를 수용하도록 큐의 MAXMSGLEN 속성을 수정해야 하는 경우 다음을 고려하십시오.

- 로컬 큐에 대해 지원되는 최소 메시지 세그먼트는 16바이트입니다.
- 전송 큐의 경우 MAXMSGLEN에 헤더에 필요한 공간도 포함되어야 합니다. 전송 큐에 넣을 수 있는 메시지 세그먼트에서 예상되는 사용자 데이터의 최대 길이보다 최소 4000바이트 더 큰 값을 사용하는 방안을 고려하십시오.

데이터 변환이 필요한 경우, 가져오기 애플리케이션은 MQGMO_CONVERT를 지정하여 해당 작업을 수행해야 합니다. 이렇게 하면 데이터 변환 엑시트에 완료 메시지가 표시되기 때문에 간단합니다. 메시지가 세그먼트되어 있고 데이터의 형식이 데이터 변환 엑시트가 미완료 데이터의 변환을 수행할 수 없는 형식인 경우 송신자 채널의 데이터를 변환하려고 시도하지 마십시오.

Multi

애플리케이션 세그먼트화

큐 관리자 세그먼트화가 적절하지 않을 때나 애플리케이션의 특정 세그먼트 경계에서 데이터 변환이 필요할 때 애플리케이션 세그먼트화가 사용됩니다.

애플리케이션 세그먼트화는 두 가지 주된 이유에서 사용됩니다.

1. 메시지가 너무 커서 애플리케이션에서 단일 버퍼로 핸들링할 수 없기 때문에 큐 관리자 세그먼트화만으로는 적절하지 않습니다.
2. 송신자 채널에서 데이터 변환을 수행해야 하며, 개별 세그먼트 변환이 가능하도록 하기 위해 세그먼트 경계가 있어야 할 위치를 넣기 애플리케이션이 규정해야 하는 형식입니다.

그러나 데이터 변환 문제가 아니거나 가져오기 애플리케이션이 항상 MQGMO_COMPLETE_MSG를 사용하는 경우에는 MQMF_SEGMENTATION_ALLOWED를 지정하여 큐 관리자 세그먼트화도 허용할 수 있습니다. 예제에서 애플리케이션은 메시지를 4개의 세그먼트로 분할합니다.

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT
```

```
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_SEGMENT
```

```
MQCMIT
```

MQPMO_LOGICAL_ORDER를 사용하지 않는 경우, 애플리케이션은 *Offset* 및 각 세그먼트의 길이를 설정해야 합니다. 이 경우에는 논리 상태가 자동으로 유지보수되지 않습니다.

가져오기 애플리케이션은 리어셈블된 메시지를 보유할 만큼 큰 버퍼를 보장할 수 없습니다. 그러므로 세그먼트를 개별적으로 처리할 준비가 되어 있어야 합니다.

세그먼트된 메시지의 경우, 이 애플리케이션은 논리 메시지를 구성하는 모든 세그먼트가 존재할 때까지 세그먼트를 하나도 처리하지 않습니다. 따라서 첫 번째 세그먼트에 대해 MQGMO_ALL_SEGMENTS_AVAILABLE이 지정됩니다. MQGMO_LOGICAL_ORDER를 지정하고 현재 논리 메시지가 있는 경우에는 MQGMO_ALL_SEGMENTS_AVAILABLE이 무시됩니다.

논리 메시지의 첫 번째 세그먼트가 검색되면, MQGMO_LOGICAL_ORDER를 사용하여 논리 메시지의 나머지 세그먼트가 순서대로 검색되는지 확인하십시오.

다른 그룹에 있는 메시지는 고려되지 않습니다. 이러한 메시지가 발생하면 큐에서 각 메시지의 첫 번째 세그먼트가 발생한 순서대로 처리됩니다.

```
GMOptions = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_SEGMENTS_AVAILABLE | MQGMO_WAIT
do while ( SegmentStatus == MQSS_SEGMENT )
  MQGET
  /* Process each remaining segment of the logical message */
  ...
MQCMIT
```

Multi 논리 메시지의 애플리케이션 세그먼트화

메시지는 그룹의 논리적 순서로 유지보수해야 하며, 그 중 일부 또는 전체가 너무 커서 애플리케이션 세그먼트화가 필요할 수 있습니다.

예제에서는 4개의 논리 메시지 그룹을 넣습니다. 세 번째 메시지를 제외한 나머지는 대용량이므로 세그먼트화해야 합니다. 이는 넣기 애플리케이션에서 수행됩니다.

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT
```

```
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP | MQMF_LAST_SEGMENT
```

```
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP | MQMF_LAST_SEGMENT
```

```
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
```

```
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_LAST_SEGMENT
```

```
MQCMIT
```

가져오기 애플리케이션에서 MQGMO_ALL_MSGS_AVAILABLE은 첫 번째 MQGET에 지정됩니다. 이는 전체 그룹이 사용 가능할 때까지 그룹의 메시지나 세그먼트가 검색되지 않음을 의미합니다. 그룹의 첫 번째 물리적 메시지가 검색되면, MQGMO_LOGICAL_ORDER를 사용하여 그룹의 세그먼트 및 메시지가 순서대로 검색되는지 확인합니다.

```

GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
              | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT

do while ( (GroupStatus != MQGS_LAST_MSG_IN_GROUP) ||
           (SegmentStatus != MQGS_LAST_SEGMENT) )
    MQGET
    /* Process a segment or complete logical message. Use the GroupStatus
       and SegmentStatus information to see what has been returned */
    ...
MQCMIT

```

참고: MQGMO_LOGICAL_ORDER를 지정하고 현재 그룹이 있는 경우에는 MQGMO_ALL_MSGS_AVAILABLE이 무시됩니다.

Multi 참조 메시지 및 대용량 객체 전송

참조 메시지를 사용하면 소스 노드나 대상 노드의 큐에 IBM MQ 객체를 저장하지 않고도 대용량 객체(최대 100MB)를 한 노드에서 다른 노드로 전송할 수 있습니다. 이는 데이터가 다른 양식으로 존재할 때(예를 들면, 메일 애플리케이션) 특히 유용합니다.

이 전송 방식을 활성화하려면 채널 양 끝에 메시지 출구를 지정합니다. 이를 수행하는 방법에 대한 정보는 [725 페이지](#)의 『채널 메시지 엑시트 프로그램』의 내용을 참조하십시오.

IBM MQ는 참조 메시지 헤더(MQRMH)의 형식을 정의합니다. 이에 대한 설명은 [MQRMH](#)를 참조하십시오. 이는 정의된 형식 이름으로 인식되고 실제 데이터가 이어질 수 있습니다.

대형 오브젝트의 전송을 시작하기 위해 애플리케이션은 이어지는 데이터 없이 참조 메시지 헤더로만 구성된 메시지를 넣을 수 있습니다. 이 메시지가 노드에서 나가면, 메시지 엑시트는 적절한 방법으로 오브젝트를 검색하여 참조 메시지에 추가합니다. 그런 다음, 수신 MCA로의 전송을 위해 송신 메시지 채널 에이전트로 메시지(이전보다 지금 더 큼)를 리턴합니다.

다른 메시지 엑시트가 수신 MCA에서 구성됩니다. 이 메시지 엑시트가 이러한 메시지 중 하나를 수신하면, 추가된 오브젝트 데이터를 사용하여 오브젝트를 작성하고 해당 데이터 없이 참조 메시지를 전달합니다. 이제 참조 메시지를 애플리케이션에서 수신할 수 있고, 이 애플리케이션은 오브젝트(또는 최소한 이 참조 메시지로 표시된 오브젝트의 일부)가 이 노드에서 작성된 것을 알고 있습니다.

송신 메시지 엑시트가 참조 메시지에 추가할 수 있는 오브젝트 데이터의 최대 크기는 채널에 대해 조정된 최대 메시지 길이로 제한됩니다. 이 엑시트는 전달되는 각 메시지의 MCA에 하나의 메시지만 리턴할 수 있습니다. 따라서 넣기 애플리케이션이 여러 메시지를 넣어도 하나의 오브젝트가 전송됩니다. 각 메시지는 추가되는 오브젝트의 논리적 길이 및 오프셋을 식별해야 합니다. 그러나 오브젝트의 전체 크기 또는 채널에서 허용하는 최대 크기를 알 수 없는 경우에는 넣기 애플리케이션이 하나의 메시지만 넣고 엑시트 자체가 전달된 메시지에 최대한 많은 데이터를 추가했을 때 다음 메시지를 전송 큐에 넣도록 송신 메시지 엑시트를 설계하십시오.

이러한 대규모 메시지 처리 방법을 사용하기 전에 다음 사항을 고려하십시오.

- MCA 및 메시지 엑시트가 IBM MQ 사용자 ID로 실행됩니다. 메시지 엑시트(따라서 사용자 ID)는 오브젝트에 액세스하여 오브젝트를 송신 끝에서 검색하거나 수신 끝에서 작성해야 합니다. 이는 오브젝트가 보편적으로 액세스 가능한 경우에만 실행 가능합니다. 이는 보안 문제를 야기합니다.
- 대용량 데이터가 추가된 참조 메시지가 여러 큐 관리자를 통과해야만 해당 목적지에 도달할 수 있다면 대용량 데이터는 중간 노드의 IBM MQ 큐에 있는 것입니다. 하지만 이러한 경우에는 특수 지원이나 엑시트를 제공할 필요가 없습니다.
- 경로 재지정 또는 데드 레터 큐잉을 허용하면 메시지 엑시트를 설계하기가 어렵습니다. 이러한 경우에는 오브젝트의 부분이 순서 상관없이 도착할 수 있습니다.
- 참조 메시지가 해당 목적지에 도착하면, 수신 메시지 엑시트는 오브젝트를 작성합니다. 그러나 이것이 MCA의 작업 단위와 동기화되지 않아 배치가 백아웃되면, 오브젝트의 이와 동일한 부분을 포함하는 다른 참조 메시지가 차후 배치에 도착하고 메시지 엑시트가 오브젝트의 동일 부분을 다시 작성하려고 시도합니다. 오브젝트가 예를 들어 일련의 데이터베이스 업데이트이면 이것은 허용되지 않을 수 있습니다. 이러한 경우, 메시지 엑시트는 적용된 업데이트의 로그를 보관해야 하고 IBM MQ 큐를 사용해야 합니다.

- 오브젝트 유형의 특성에 따라 오브젝트가 더 이상 필요하지 않으면 삭제할 수 있도록 메시지 엑시트 및 애플리케이션이 사용 계수 유지보수에 협업해야 합니다. 인스턴스 ID도 필요하기 때문에 이에 대한 필드가 참조 메시지 헤더에 제공됩니다(MQRMH 참조).
- 참조 메시지를 분배 목록으로 넣은 경우 해당 노드에서 각 결과 분배 목록 또는 개별 목적지에 대해 오브젝트를 검색할 수 있어야 합니다. 사용 계수를 유지보수해야 할 수 있습니다. 또한 노드가 목록의 일부 목적지에 대해서는 최종 노드이지만 다른 목적지에 대해서는 중간 노드일 가능성도 고려하십시오.
- 대용량 데이터는 일반적으로 변환되지 않습니다. 변환은 메시지 엑시트가 호출되기 전에 일어나기 때문입니다. 이러한 이유로, 원래 송신자 채널에 대해서는 변환을 요청하지 않아야 합니다. 참조 메시지가 중간 노드를 통과하면, 요청 시 중간 노드에서 전송될 때 대용량 데이터가 변환됩니다.
- 참조 메시지는 세그먼트화할 수 없습니다.

MQRMH 및 MQMD 구조 사용

참조 메시지 헤더 및 메시지 디스크립터의 필드에 대한 설명은 [MQRMH](#) 및 [MQMD](#)를 참조하십시오.

MQMD 구조에서 *Format* 필드를 MQFMT_REF_MSG_HEADER로 설정하십시오. MQHREF 형식은 MQGET에서 요청 시 다음 대용량 데이터와 함께 IBM MQ에서 자동으로 변환됩니다.

MQRMH의 *DataLogicalOffset* 및 *DataLogicalLength* 필드 사용의 예는 다음과 같습니다.

넣기 애플리케이션은 다음과 같은 참조 메시지를 넣을 수 있습니다.

- 물리적 데이터가 없음
- *DataLogicalLength* = 0(이 메시지는 전체 오브젝트를 나타냄)
- *DataLogicalOffset* = 0

오브젝트 길이가 70,000바이트라고 가정할 때, 송신 메시지 엑시트는 처음 40,000바이트를 다음과 같은 참조 메시지로 채널을 따라 송신합니다.

- MQRMH에 따르는 40,000바이트의 물리적 데이터
- *DataLogicalLength* = 40000
- *DataLogicalOffset* = 0(오브젝트의 시작부터)

그리고 다음과 같은 또 하나의 메시지를 전송 큐에 배치합니다.

- 물리적 데이터가 없음
- *DataLogicalLength* = 0(오브젝트의 끝까지). 여기에 값을 30,000으로 지정할 수 있습니다.
- *DataLogicalOffset* = 40000(이 지점부터 시작)

이 메시지 엑시트가 송신 메시지 엑시트에서 확인되면, 나머지 30,000바이트의 데이터가 추가되고 필드가 다음으로 설정됩니다.

- MQRMH에 따르는 30,000바이트의 물리적 데이터
- *DataLogicalLength* = 30000
- *DataLogicalOffset* = 40000(이 지점부터 시작)

MQRMHF_LAST 플래그도 설정됩니다.

참조 메시지 사용을 위해 제공되는 샘플 프로그램에 대한 설명은 971 페이지의 [『멀티플랫폼용 샘플 절차적 프로그램 사용』](#)의 내용을 참조하십시오.

메시지 대기

메시지가 큐에 도착할 때까지 프로그램을 대기시키려면 MQGMO 구조의 *Options* 필드에 MQGMO_WAIT 옵션을 지정하십시오.

MQGMO 구조의 *WaitInterval* 필드를 사용하여 메시지가 큐에 도착할 때까지 MQGET 호출을 대기시킬 최대 시간(밀리초)을 지정하십시오.

메시지가 이 시간 내에 도착하지 않으면, MQGET 호출은 MQRC_NO_MSG_AVAILABLE 이유 코드와 함께 완료됩니다.

WaitInterval 필드에서 상수 `MQWI_UNLIMITED`를 사용하여 무제한 대기 간격을 지정할 수 있습니다. 그러나 이벤트가 제어 범위를 벗어나면 프로그램이 오랫동안 대기할 수 있으므로, 이 상수는 신중하게 사용하십시오. 무제한 대기 간격을 지정하면 IMS 시스템 종료에 방지되므로, IMS 애플리케이션은 이를 지정하지 않아야 합니다. (IMS 종료 시 모든 종속 영역이 종료되어야 합니다.) 대신 IMS 애플리케이션은 유한 대기 간격을 지정할 수 있습니다. 그런 다음, 해당 간격 이후에 메시지를 검색하지 않고 호출이 완료되면 대기 옵션을 사용하여 다른 `MQGET` 호출을 발행하십시오.

참고: 둘 이상의 프로그램이 동일한 공유 큐에서 메시지를 제거하려고 대기 중이면, 메시지 도착으로 한 프로그램만 활성화됩니다. 하지만 둘 이상의 프로그램이 메시지를 찾아보기 위해 대기 중인 경우에는 모든 프로그램이 활성화될 수 있습니다. 자세한 정보는 [MQGMO](#)에서 `MQGMO` 구조의 *Options* 필드에 대한 설명을 참조하십시오.

대기 간격이 만료되기 전에 큐 또는 큐 관리자의 상태가 변경되면 다음 조치가 발생합니다.

- 큐 관리자가 정지 중 상태로 전환되고 `MQGMO_FAIL_IF QUIESCING` 옵션을 사용한 경우, 대기가 취소되고 `MQGET` 호출이 `MQRC_Q_MGR QUIESCING` 이유 코드와 함께 완료됩니다. 이 옵션을 사용하지 않으면 호출이 대기 중 상태를 유지합니다.
- **z/OS** z/OS에서 (CICS 또는 IMS 애플리케이션의) 연결이 정지 중 상태로 전환되고 `MQGMO_FAIL_IF QUIESCING` 옵션을 사용한 경우, 대기가 취소되고 `MQGET` 호출이 `MQRC_CONN QUIESCING` 이유 코드와 함께 완료됩니다. 이 옵션을 사용하지 않으면 호출이 대기 중 상태를 유지합니다.
- 큐 관리자가 강제 중지되거나 취소되면, `MQGET` 호출이 `MQRC_Q_MGR STOPPING` 또는 `MQRC_CONNECTION_BROKEN` 이유 코드와 함께 완료됩니다.
- Get 요청이 즉시 금지되도록 큐(또는 큐 이름이 확인되는 큐)의 속성이 변경되면 대기가 취소되고 `MQGET` 호출이 `MQRC_GET_INHIBITED` 이유 코드와 함께 완료됩니다.
- 큐(또는 큐 이름이 확인되는 큐)의 속성이 `FORCE` 옵션이 필요한 방식으로 변경되면, 대기가 취소되고 `MQGET` 호출이 `MQRC_OBJECT_CHANGED` 이유 코드와 함께 완료됩니다.

z/OS 애플리케이션을 둘 이상의 큐에서 대기시키려면 IBM MQ for z/OS의 신호 기능을 사용하십시오 (812 페이지의 『신호보내기』 참조). 이러한 조치가 발생하는 상황에 관한 자세한 정보는 [MQGMO](#)를 참조하십시오.

신호보내기

신호보내는 IBM MQ for z/OS에서만 지원됩니다.

신호보내는 운영 체제에서 예상 메시지가 큐에 도착할 때 프로그램에 알리거나 신호를 보내게 하는 `MQGET` 호출의 옵션입니다. 이 옵션은 프로그램이 신호를 대기하는 동안 다른 작업을 계속할 수 있도록 하기 때문에 811 페이지의 『메시지 대기』 주제에 설명된 *get with wait* 함수와 유사합니다. 하지만 신호보내기를 사용하는 경우 애플리케이션 스레드를 비우고 운영 체제를 통해 메시지가 도착할 때 프로그램에 알릴 수 있습니다.

신호 설정

신호를 설정하려면 `MQGET` 호출에 사용되는 `MQGMO` 구조에서 다음을 수행하십시오.

1. *Options* 필드에서 `MQGMO_SET_SIGNAL` 옵션을 설정하십시오.
2. *WaitInterval* 필드에서 신호의 최대 수명을 설정하십시오. 이는 IBM MQ에서 큐를 모니터할 시간의 길이(밀리초)를 설정합니다. `MQWI_UNLIMITED` 값을 사용하여 무제한 수명을 지정하십시오.
참고: 무제한 대기 간격을 지정하면 IMS 시스템 종료에 방지되므로, IMS 애플리케이션은 이를 지정하지 않아야 합니다. (IMS 종료 시 모든 종속 영역이 종료되어야 합니다.) 대신에 IMS 애플리케이션은 일정한 간격으로 ECB의 상태를 조사할 수 있습니다(3단계 참조). 프로그램은 동시에 여러 큐 핸들에 신호를 설정할 수 있습니다.
3. *Signal1* 필드에서 이벤트 제어 블록(ECB)의 주소를 지정하십시오. 이렇게 하여 신호의 결과를 알립니다. ECB 스토리지는 큐가 처리완료될 때까지 사용 가능한 상태로 유지되어야 합니다.

참고: `MQGMO_SET_SIGNAL`은 `MQGMO_WAIT` 옵션과 함께 사용할 수 없습니다.

메시지가 도착할 때

적당한 메시지가 도착할 때 완료 코드는 ECB로 리턴됩니다.

완료 코드는 다음 중 하나를 설명합니다.

- 신호가 설정된 메시지가 큐에 도착했습니다. 메시지가 신호를 요청한 프로그램에 대해 예약되지 않았으므로, 프로그램은 MQGET 호출을 다시 발행하여 메시지를 가져와야 합니다.

참고: 다른 애플리케이션이 신호 수신과 다른 MQGET 호출 발행 사이의 시간에 메시지를 가져올 수 있습니다.

- 설정한 대기 간격이 만료되었으며 신호가 설정된 메시지가 큐에 도착하지 않았습니다. IBM MQ가 신호를 취소했습니다.
- 신호가 취소되었습니다. 이는 예를 들어 신호 MQGET 호출이 더 이상 허용되지 않도록 큐 관리자가 중지되거나 큐의 속성이 변경된 경우에 발생합니다.

적당한 메시지가 이미 큐에 있을 때, MQGET 호출은 신호를 보내지 않고 MQGET 호출과 같은 방식으로 완료됩니다. 또한 오류가 바로 감지되면, 호출은 완료되고 리턴 코드가 설정됩니다.

호출이 허용되고 메시지가 바로 사용 가능하지 않을 때, 다른 작업을 계속할 수 있도록 제어가 프로그램으로 리턴됩니다. 메시지 디스크립터의 출력 필드가 설정되지 않지만, **CompCode** 매개변수는 MQCC_WARNING으로 설정되고 **Reason** 매개변수는 MQRC_SIGNAL_REQUEST_ACCEPTED로 설정됩니다.

신호보내기를 통해 MQGET 호출 수행 시 IBM MQ가 애플리케이션으로 리턴할 수 있는 내용에 대한 정보는 [MQGET](#)을 참조하십시오.

ECB가 게시될 때까지 대기하는 동안 수행할 다른 작업이 프로그램에 없는 경우 프로그램은 다음을 사용하여 ECB를 기다릴 수 있습니다.

- CICS Transaction Server for z/OS 프로그램의 경우, EXEC CICS WAIT EXTERNAL 명령
- 배치 및 IMS 프로그램의 경우, z/OS WAIT 매크로

신호가 설정되는 동안 큐 또는 큐 관리자의 상태가 변경되는 경우(즉, ECB가 아직 게시되지 않았음) 다음 조치가 발생합니다.

- 큐 관리자가 정지 중 상태로 전환되고 MQGMO_FAIL_IF_QUIESCING 옵션을 사용한 경우 신호가 취소됩니다. ECB에는 MQEC_Q_MGR_QUIESCING 완료 코드가 게시됩니다. 이 옵션을 사용하지 않으면 신호가 계속 설정되어 있습니다.
- 큐 관리자가 강제 중지되거나 취소되면 신호가 취소됩니다. 신호는 MQEC_WAIT_CANCELED 완료 코드와 함께 전달됩니다.
- 가져오기 요청이 즉시 금지되도록 큐(또는 큐 이름이 확인되는 큐)의 속성이 변경되면 신호가 취소됩니다. 신호는 MQEC_WAIT_CANCELED 완료 코드와 함께 전달됩니다.

참고:

1. 둘 이상의 프로그램이 메시지를 제거하기 위한 신호를 동일한 공유 큐에 설정한 경우, 메시지 도착으로 한 프로그램만 활성화됩니다. 하지만 둘 이상의 프로그램이 메시지를 찾아보기 위해 대기 중인 경우에는 모든 프로그램이 활성화될 수 있습니다. 큐 관리자가 활성화할 애플리케이션을 결정할 때 따르는 규칙은 대기 중인 애플리케이션에 대한 규칙과 동일합니다. 자세한 내용은 [MQGMO - 메시지 수신 옵션의 MQGMO 구조체 내 필드 Options](#) 설명을 참조하십시오.
2. 대기 및 신호 옵션을 혼합 사용하여 동일한 메시지를 대기하는 MQGET 호출이 두 개 이상 있는 경우, 각 대기 호출은 동일하게 간주됩니다. 자세한 내용은 [MQGMO - Get-message options](#) 문서의 MQGMO 구조체 필드 Options 설명을 참조하십시오.
3. 일부 조건 하에서 MQGET 호출로 메시지를 검색하고 신호(동일한 메시지 도착의 결과로 발생하는)를 전달할 수 있습니다. 이는 (신호가 전달되었기 때문에) 프로그램이 다른 MQGET 호출을 발행할 때 사용 가능한 메시지가 없을 수 있음을 의미합니다. 이 상황을 테스트하도록 프로그램을 설계하십시오.

신호 설정 방법에 대한 정보는 MQGMO_SET_SIGNAL 옵션 설명과 의 필드를 [Signal1](#) [Signal1](#) 참조하십시오.

z/OS 백아웃 건너뛰기

MQGET 호출에 MQGMO_MARK_SKIP_BACKOUT 옵션을 지정하여 애플리케이션 프로그램이 MQGET-error-backout 루프를 입력하지 못하게 할 수 있습니다.

애플리케이션 프로그램은 작업 단위의 일부로 하나 이상의 MQGET 호출을 발행하여 큐에서 메시지를 가져올 수 있습니다. 애플리케이션 프로그램은 오류를 감지하면 작업 단위를 백아웃할 수 있습니다. 이 경우 해당 작업 단위에서 업데이트된 모든 자원이 작업 단위가 시작되기 전의 상태로 복원되고 MQGET 호출로 검색된 메시지가 다시 인스턴스화됩니다.

이러한 메시지는 복원되면 애플리케이션 프로그램이 발행한 후속 MQGET 호출에서 사용할 수 있습니다. 대부분의 경우, 이로 인해 애플리케이션 프로그램에서 문제점이 발생하지 않습니다. 하지만 백아웃을 초래하는 오류를 회피할 수 없는 경우, 큐에서 메시지를 복원하면 애플리케이션 프로그램이 *MQGET-error-backout* 루프를 입력할 수 있습니다.

이 문제점을 방지하려면 MQGET 호출에 MQGMO_MARK_SKIP_BACKOUT 옵션을 지정하십시오. 이 옵션은 MQGET 요청을 애플리케이션 시작 백아웃에 관련 없는 것으로 표시하므로 이 요청이 백아웃되지 않습니다. 이 옵션을 사용하면 백아웃이 발생할 때 필요에 따라 다른 자원에 대한 업데이트가 백아웃되지만, 표시된 메시지는 새 작업 단위 아래에서 검색된 것처럼 처리됩니다.

애플리케이션 프로그램은 IBM MQ 호출을 발행하여 새 작업 단위를 커밋하거나 새 작업 단위를 백아웃해야 합니다. 예를 들어, 프로그램은 예외 핸들링(예: 메시지가 제거된 것을 진원지에 알림)을 수행할 수 있고 큐에서 메시지를 제거하도록 작업 단위를 커밋할 수 있습니다. 어떤 이유로든 새 작업 단위가 백아웃되면 큐에서 메시지가 복원됩니다.

작업 단위 내에 백아웃 건너뛰기로 표시된 MQGET 요청은 하나만 있을 수 있습니다. 하지만 백아웃 건너뛰기로 표시되지 않은 다른 메시지는 여럿 있을 수 있습니다. 메시지가 백아웃 건너뛰기로 표시된 경우, 작업 단위 내에서 MQGMO_MARK_SKIP_BACKOUT을 지정하는 추가 MQGET 호출이 이유 코드 MQRC_SECOND_MARK_NOT_ALLOWED와 함께 실패합니다.

참고:

1. 표시된 메시지는 이를 포함하는 작업 단위가 이를 백아웃하라는 애플리케이션 요청에 의해 종료된 경우에만 백아웃을 건너뛰니다. 작업 단위가 다른 어떤 이유로 백아웃되면, 메시지는 백아웃을 건너뛰도록 표시되지 않은 경우와 같은 방식으로 큐에 백아웃됩니다.
2. RRS로 제어되는 작업 단위에 참여 중인 Db2 스토어드 프로시저 내에서는 백아웃 건너뛰기가 지원되지 않습니다. 예를 들어, MQGMO_MARK_SKIP_BACKOUT 옵션을 포함하는 MQGET 호출은 이유 코드 MQRC_OPTION_ENVIRONMENT_ERROR와 함께 실패합니다.

815 페이지의 그림 93에서는 백아웃을 건너뛰기 위해 MQGET 요청이 필요한 경우 애플리케이션 프로그램에 포함되는 일반적인 일련의 단계를 설명합니다.

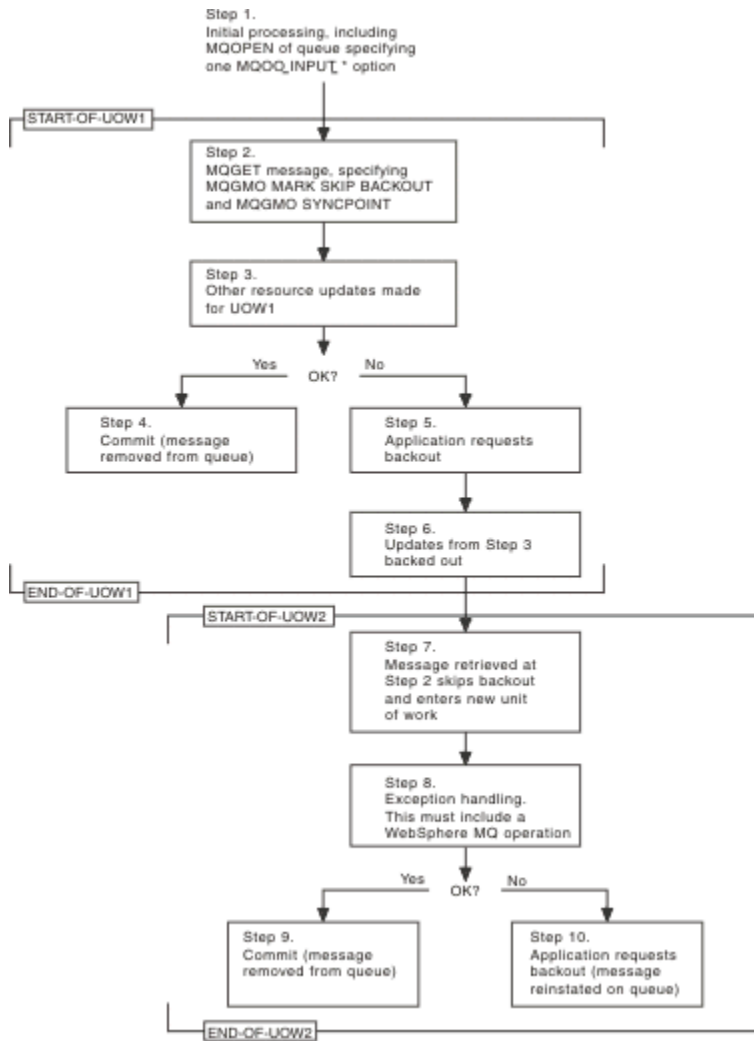


그림 93. MQGMO_MARK_SKIP_BACKOUT을 사용한 백아웃 건너뛰기

815 페이지의 그림 93의 단계는 다음과 같습니다.

1단계

(2단계의 큐에서 메시지를 가져오기 위해 MQOO_INPUT_* 옵션 중 하나를 지정하는) 큐를 여는 MQOPEN 호출을 포함하여 트랜잭션 내에서 초기 처리가 발생합니다.

2단계

MQGET이 MQGMO_SYNCPOINT 및 MQGMO_MARK_SKIP_BACKOUT과 함께 호출됩니다. MQGMO_MARK_SKIP_BACKOUT을 적용하려면 작업 단위 내에 MQGET이 있어야 하므로 MQGMO_SYNCPOINT가 필요합니다. 815 페이지의 그림 93에서 이 작업 단위는 UOW1이라고 합니다.

3단계

UOW1의 일부로 기타 자원 업데이트가 작성됩니다. 이 업데이트에는 (MQGMO_MARK_SKIP_BACKOUT 없이 발행된) 추가 MQGET 호출이 포함될 수 있습니다.

4단계

2단계 및 3단계의 모든 업데이트가 필요에 따라 완료됩니다. 애플리케이션 프로그램이 업데이트를 커밋하고 UOW1이 종료됩니다. 2단계에서 검색된 메시지는 큐에서 제거됩니다.

5단계

2단계 및 3단계의 일부 업데이트가 필요에 따라 완료됩니다. 애플리케이션 프로그램은 이러한 단계에서 작성된 업데이트를 백아웃하도록 요청합니다.

6단계

3단계에서 작성된 업데이트가 백아웃됩니다.

7단계

2단계에서 작성된 MQGET 요청이 백아웃을 건너뛰고 새 작업 단위 UOW2의 일부가 됩니다.

8단계

UOW2는 백아웃되는 UOW1에 대한 응답으로 예외 핸들링을 수행합니다 (예를 들어, 다른 큐에 대한 MQPUT 호출로 UOW1을 백아웃시키는 문제점이 발생했음을 나타냄).

9단계

필요에 따라 8단계가 완료됩니다. 애플리케이션 프로그램이 활동을 커밋하고 UOW2가 종료됩니다.

MQGET 요청이 UOW2의 일부이므로(7단계 참조), 이 커밋에 의해 메시지가 큐에서 제거됩니다.

10단계

필요에 따라 8단계가 완료되지 않고 애플리케이션 프로그램이 UOW2를 백아웃합니다. 메시지 가져오기 요청이 UOW2의 일부이므로(7단계 참조) 이 또한 백아웃되고 큐에서 복원됩니다. 이제는 (큐의 다른 메시지와 같은 방식으로) 이 애플리케이션 프로그램 또는 다른 애플리케이션 프로그램에 의해 발행된 추가 MQGET 호출에서 사용 가능합니다.

애플리케이션 데이터 변환

MCA는 필요하면 메시지 디스크립터 및 헤더 데이터를 필수 문자 세트 및 인코딩으로 변환합니다. 링크의 양 끝 (즉, 로컬 MCA 또는 리모트 MCA)에서 변환을 수행할 수 있습니다.

애플리케이션이 큐에 메시지를 넣으면 로컬 큐 관리자가 제어 정보를 메시지 디스크립터에 추가하여 메시지가 큐 관리자 및 MCA에 의해 처리될 때 메시지 제어를 용이하게 합니다. 환경에 따라 메시지 헤더 데이터 필드가 로컬 시스템의 문자 세트 및 인코딩으로 작성됩니다.

시스템 간에 메시지를 이동할 때 애플리케이션 데이터를 수신 시스템에 필요한 문자 세트 및 인코딩으로 변환해야 하는 경우가 있습니다. 이는 수신 시스템의 애플리케이션 프로그램 내에서 수행되거나 송신 시스템의 MCA에 의해 수행될 수 있습니다. 수신 시스템에서 데이터 변환이 지원되는 경우, 송신 시스템에서 이미 발생한 변환에 의존하기보다는 애플리케이션 프로그램을 사용하여 애플리케이션 데이터를 변환하십시오.

MQGET 호출에 전달된 MQGMO 구조의 *Options* 필드에 MQGMO_CONVERT 옵션을 지정하고 다음 명령문이 모두 해당될 때 애플리케이션 프로그램 내에서 애플리케이션 데이터가 변환됩니다.

- 큐의 메시지와 연관된 MQMD 구조에서 설정된 *CodedCharSetId* 또는 *Encoding* 필드가 MQGET 호출에 지정된 MQMD 구조에서 설정된 *CodedCharSetId* 또는 *Encoding* 필드와 다릅니다.
- 메시지와 연관된 MQMD 구조의 *Format* 필드가 MQFMT_NONE이 아닙니다.
- MQGET 호출에 지정된 *BufferLength*가 0이 아닙니다.
- 메시지 데이터 길이가 0이 아닙니다.
- 큐 관리자가 MQGET 호출 및 메시지와 연관된 MQMD 구조에 지정된 *CodedCharSetId* 및 *Encoding* 필드 간의 변환을 지원합니다. 지원되는 코드화 문자 세트 ID 및 시스템 인코딩의 세부사항은 [CodedCharSetId](#) 및 [인코딩](#)을 참조하십시오.
- 큐 관리자가 메시지 형식의 변환을 지원합니다. 메시지와 연관된 MQMD 구조의 *Format* 필드가 내장 형식 중 하나인 경우 큐 관리자가 메시지를 변환할 수 있습니다. *Format*이 내장 형식 중 하나가 아닌 경우, 메시지를 변환하기 위한 데이터 변환 엑시트를 작성해야 합니다.

송신 MCA가 데이터를 변환하는 경우 변환이 필요한 각 송신자 또는 서버 채널의 정의에 CONVERT(YES) 키워드를 지정하십시오. 데이터 변환에 실패하면 메시지는 송신 큐 관리자의 DLQ로 송신되고 MQDLH 구조의 *Feedback* 필드에 이유가 표시됩니다. 메시지를 DLQ에 넣을 수 없는 경우, 채널이 닫히며 변환되지 않는 메시지는 전송 큐에 남게 됩니다. 송신 MCA 대신 애플리케이션 내에서 데이터를 변환하면 이러한 상황이 발생하지 않습니다.

일반적으로 내장 형식 또는 데이터 변환 엑시트에 의해 문자 데이터로 설명된 메시지의 데이터는 메시지에 사용되는 코드화 문자 세트에서 요청된 문자 세트로 변환되고 숫자 필드는 요청된 인코딩으로 변환됩니다.

내장 형식 변환 시 사용되는 변환 처리 규약에 대한 추가 세부사항 및 고유 데이터 변환 엑시트 작성에 대한 정보는 729 페이지의 『데이터 변환 엑시트 작성』의 내용을 참조하십시오. 지원되는 시스템 인코딩 및 언어 지원 테이블에 대한 정보는 [자국어\(NL\)](#) 및 [시스템 인코딩](#)도 참조하십시오.

EBCDIC 개행 문자 변환

EBCDIC 플랫폼에서 ASCII 플랫폼으로 보내는 데이터가 다시 돌려받는 데이터와 동일한지 확인해야 하는 경우 EBCDIC 개행 문자의 변환을 제어해야 합니다.

IBM MQ가 수정되지 않은 변환 테이블을 사용하도록 강제 실행하는 플랫폼 종속 스위치를 사용하여 이 작업을 수행할 수 있지만, 결과로 초래되는 일치하지 않는 작동을 인식해야 합니다.

이 문제점은 EBCDIC 개행 문자가 플랫폼 또는 변환 테이블 전체에서 일관되게 변환되지 않기 때문에 발생합니다. 결과적으로 데이터가 ASCII 플랫폼에 표시된 경우 형식이 올바르지 않을 수 있습니다. 따라서 예를 들어 RUNMQSC를 사용하여 ASCII 플랫폼에서 원격으로 IBM i 시스템을 관리하기가 어려울 수 있습니다.

EBCDIC 형식 데이터를 ASCII 형식으로 변환하는 방법에 대한 추가 정보는 [데이터 변환](#)을 참조하십시오.

큐의 메시지 찾아보기

이 정보를 사용하여 MQGET 호출을 통해 큐의 메시지를 찾아보는 방법에 대해 알아봅니다.

MQGET 호출을 사용하여 큐의 메시지를 찾아보려면 다음을 수행하십시오.

1. MQOO_BROWSE 옵션을 지정해서 MQOPEN을 호출하여 찾아볼 큐를 여십시오.
2. 큐의 첫 번째 메시지를 찾아보려면 MQGMO_BROWSE_FIRST 옵션을 사용하여 MQGET을 호출하십시오. 원하는 메시지를 찾으려면, MQGMO_BROWSE_NEXT 옵션으로 반복해서 MQGET을 호출하여 여러 메시지를 처리하십시오.

모든 메시지를 확인하려면, 각 MQGET 호출 후 MQMD 구조의 *MsgId* 및 *CorrelId* 필드를 널로 설정해야 합니다.

3. MQCLOSE를 호출하여 큐를 닫으십시오.

찾아보기 커서

찾아보기 위해 큐를 열 때(MQOPEN), 해당 호출은 찾아보기 옵션 중 하나를 사용하는 MQGET 호출용 찾아보기 커서를 설정합니다. 찾아보기 커서는 큐의 첫 번째 메시지 앞에 위치 지정되는 논리 포인터로 생각할 수 있습니다.

동일 큐에 대한 여러 MQOPEN 요청을 발행하여 (단일 프로그램에서) 둘 이상의 찾아보기 커서를 활성화할 수 있습니다.

찾아보기 위해 MQGET을 호출할 때, MQGMO 구조에서 다음 옵션 중 하나를 사용하십시오.

MQGMO_BROWSE_FIRST

MQMD 구조에 지정된 조건을 충족하는 첫 번째 메시지의 사본을 가져옵니다.

MQGMO_BROWSE_NEXT

MQMD 구조에 지정된 조건을 충족하는 다음 메시지의 사본을 가져옵니다.

MQGMO_BROWSE_MSG_UNDER_CURSOR

현재 커서가 가리키는 메시지 즉, MQGMO_BROWSE_FIRST 또는 MQGMO_BROWSE_NEXT 옵션을 사용하여 마지막으로 검색된 메시지의 사본을 가져옵니다.

모든 경우에 메시지가 큐에 남아 있습니다.

큐를 열면, 찾아보기 커서는 큐의 첫 번째 메시지 바로 앞에 논리적으로 위치 지정됩니다. 이는 MQOPEN 호출 직후에 MQGET 호출을 수행하는 경우 MQGMO_BROWSE_NEXT 옵션을 사용하여 첫 번째 메시지를 찾아볼 수 있으므로 MQGMO_BROWSE_FIRST 옵션을 사용할 필요가 없다는 것을 의미합니다.

큐에서 메시지가 복사되는 순서는 큐의 **MsgDeliverySequence** 속성에 의해 판별됩니다. (자세한 정보는 [791 페이지](#)의 『큐에서 메시지를 검색하는 순서』의 내용을 참조하십시오.)

- [818 페이지](#)의 『FIFO(First In, First Out) 순서의 큐』
- [818 페이지](#)의 『우선순위 순서의 큐』
- [818 페이지](#)의 『커밋되지 않는 메시지』
- [818 페이지](#)의 『큐 순서 변경』
-  [818 페이지](#)의 『큐 인덱스 사용』

FIFO(First In, First Out) 순서의 큐

이 순서의 큐에서 첫 번째 메시지는 큐에서 가장 긴 메시지입니다.

큐에서 순차적으로 메시지를 읽으려면 MQGMO_BROWSE_NEXT를 사용하십시오. 이 순서의 큐는 메시지가 끝에 배치되어 있으므로, 찾아보는 동안 큐에 들어진 메시지를 볼 수 있습니다. 커서가 큐의 끝에 도달한 것을 인식하면, 찾아보기 커서는 현재 위치를 유지하며 MQRC_NO_MSG_AVAILABLE과 함께 리턴합니다. 이 커서는 추가 메시지를 기다리도록 그대로 놔두거나, MQGMO_BROWSE_FIRST 호출을 사용하여 큐의 시작 부분으로 재설정할 수 있습니다.

우선순위 순서의 큐

이 순서의 큐에서 첫 번째 메시지는 큐에서 가장 길고 MQOPEN 호출이 발행될 때 가장 높은 우선순위를 가진 메시지만입니다.

큐에서 메시지를 읽으려면 MQGMO_BROWSE_NEXT를 사용하십시오.

찾아보기 커서는 다음 메시지를 가리키고, 첫 번째 메시지의 우선순위부터 작동하기 시작해서 가장 낮은 우선순위의 메시지부터 끝냅니다. 현재 찾아보기 커서에 의해 식별된 메시지보다 우선순위가 더 낮거나 같지만 하면 이 시간에 큐에 들어진 모든 메시지를 찾아봅니다.

더 높은 우선순위의 큐에 들어진 모든 메시지를 찾아보려면 다음을 수행해야 합니다.

- 찾아볼 큐를 다시 엽니다. 이 시점에서 새 찾아보기 큐가 설정됩니다.
- MQGMO_BROWSE_FIRST 옵션 사용

커밋되지 않는 메시지

커밋되지 않는 메시지는 찾아보기 커서가 건너뛰기 때문에 찾아보기에 표시되지 않습니다.

작업 단위가 커밋될 때까지 작업 단위 내의 메시지를 찾아볼 수 없습니다. 메시지는 커밋될 때 큐에서 해당 위치를 변경하지 않습니다. 따라서 MQGMO_BROWSE_FIRST 옵션을 사용하여 큐 작업을 다시 수행하지 않는 한, 건너뛴 커밋되지 않는 메시지는 커밋된 경우에도 표시되지 않습니다.

큐 순서 변경

큐에 메시지가 있는 동안에 메시지 전달 순서가 우선순위에서 FIFO로 변경되더라도 이미 큐에 넣은 메시지의 순서는 변경되지 않습니다. 나중에 큐에 추가된 메시지는 큐의 기본 우선순위를 가집니다.

큐 인덱스 사용



특정 형태의 브라우저가 사용될 때 IBM MQ for z/OS, 쿼트 매니저는 인덱싱된 큐를 탐색합니다. 이때 해당 큐에는 단일 우선순위(지속적 또는 비지속적 또는 둘 다)의 메시지만 포함됩니다.

색인화된 큐에 단일 우선순위의 메시지만 포함된 경우 다음 찾아보기 양식이 사용됩니다.

1. 큐가 MSGID로 색인화된 경우, MQMD 구조에서 MSGID를 전달하는 찾아보기 요청을 색인을 사용해 처리하여 대상 메시지를 찾습니다.
2. 큐가 CORRELID로 색인화된 경우, MQMD 구조에서 CORRELID를 전달하는 찾아보기 요청을 색인을 사용해 처리하여 대상 메시지를 찾습니다.
3. 큐가 GROUPID로 색인화된 경우, MQMD 구조에서 GROUPID를 전달하는 찾아보기 요청을 색인을 사용해 처리하여 대상 메시지를 찾습니다.

찾아보기 요청이 MQMD 구조에서 MSGID, CORRELID 또는 GROUPID를 전달하지 않고 큐가 색인화되고 메시지가 리턴되면, 메시지의 색인 항목을 찾아 그 안에 있는 정보를 사용하여 찾아보기 커서를 업데이트해야 합니다. 선택 가능한 다양한 색인 값을 사용하는 경우 상당한 추가 처리가 찾아보기 요청에 추가되지 않습니다.

메시지 길이를 알 수 없는 경우 메시지 찾아보기

메시지 크기를 모를 때 메시지를 찾아보고 *MsgId*, *CorrelId* 또는 *GroupId* 필드를 사용하지 않고 메시지를 찾으려는 경우, MQGMO_BROWSE_MSG_UNDER_CURSOR 옵션을 사용할 수 있습니다.

1. MQGET을 발행합니다(다음 옵션 사용).

- MQGMO_BROWSE_FIRST 또는 MQGMO_BROWSE_NEXT 옵션
- MQGMO_ACCEPT_TRUNCATED_MSG 옵션
- 버퍼 길이 0

참고: 다른 프로그램이 동일한 메시지를 가져올 가능성이 있으면 MQGMO_LOCK 옵션 사용도 고려하십시오. MQRC_TRUNCATED_MSG_ACCEPTED가 리턴되어야 합니다.

2. 리턴된 *DataLength*를 사용하여 필요한 스토리지를 할당하십시오.

3. MQGMO_BROWSE_MSG_UNDER_CURSOR를 사용하여 MQGET을 발행합니다.

지정된 메시지가 마지막으로 검색된 메시지이면 찾아보기 커서가 이동하지 않습니다. MQGMO_LOCK 옵션을 사용하여 메시지를 잠그거나 MQGMO_UNLOCK 옵션을 사용하여 잠긴 메시지를 잠금 해제하도록 선택할 수 있습니다.

큐가 열린 이후로 MQGMO_BROWSE_FIRST 또는 MQGMO_BROWSE_NEXT 옵션으로 MQGET이 발행되지 않으면 호출은 실패합니다.

찾아본 메시지 제거

찾아보기 및 메시지 제거를 위해 큐를 연 경우 이미 찾아본 메시지를 큐에서 제거할 수 있습니다. (MQOPEN 호출에 MQOO_BROWSE 옵션 중 하나와 MQOO_INPUT_* 옵션을 지정해야 합니다.)

메시지를 제거하려면 MQGET을 다시 호출하지만 MQGMO 구조의 *Options* 필드에서 MQGMO_MSG_UNDER_CURSOR를 지정하십시오. 이 경우, MQGET 호출은 MQMD 구조의 *MsgId*, *CorrelId* 및 *GroupId* 필드를 무시합니다.

찾아보기 및 제거 단계 사이의 시간에 다른 프로그램이 찾아보기 커서 아래에 있는 메시지를 포함하여 큐에서 메시지를 제거했을 수 있습니다. 이 경우, MQGET 호출은 메시지가 사용 불가능함을 나타내는 이유 코드를 리턴합니다.

논리 순서대로 메시지 찾아보기

791 페이지의 『논리적 및 물리적 정렬』에서는 큐에 있는 메시지의 논리 순서와 물리 순서의 차이점에 대해 설명합니다. 이 차이는 큐를 찾아볼 때 특히 중요합니다. 일반적으로 메시지는 삭제되지 않으며 찾아보기 조작은 큐의 처음부터 시작할 필요가 없기 때문입니다.

애플리케이션이 (논리 순서를 사용하여) 한 그룹의 다양한 메시지를 찾아보는 경우 논리 순서에 따라서 다음 그룹의 시작 부분에 도달하는 것이 중요합니다. 이는 한 그룹의 마지막 메시지가 실제로 다음 그룹의 첫 번째 메시지 이후에 발생할 수 있기 때문입니다. MQGMO_LOGICAL_ORDER 옵션은 큐를 스캐닝할 때 논리 순서에 따르는지 확인합니다.

찾아보기 조작에서 MQGMO_ALL_MSGS_AVAILABLE(또는 MQGMO_ALL_SEGMENTS_AVAILABLE)은 신중하게 사용하십시오. MQGMO_ALL_MSGS_AVAILABLE을 포함한 논리 메시지의 경우를 고려하십시오. 이 결과로 그룹에 남아 있는 모든 메시지가 존재하는 경우에만 논리 메시지를 사용할 수 있습니다. 그렇지 않은 경우, 메시지가 무시됩니다. 이는 누락된 메시지가 나중에 도착할 때 다음 찾아보기 조작에서 알림을 받지 못한다는 것을 의미합니다.

예를 들어, 다음과 같은 논리 메시지가 존재하고

```
Logical message 1 (not last) of group 123
Logical message 1 (not last) of group 456
Logical message 2 (last)      of group 456
```

찾아보기 함수가 MQGMO_ALL_MSGS_AVAILABLE과 함께 발행된 경우, 그룹 456의 첫 번째 논리 메시지가 리턴되고 이 논리 메시지에 찾아보기 커서가 남습니다. 그룹 123의 두 번째(마지막) 메시지가 지금 도착하고

```
Logical message 1 (not last) of group 123
Logical message 2 (last)     of group 123
Logical message 1 (not last) of group 456 <=== browse cursor
Logical message 2 (last)     of group 456
```

동일한 다음 찾아보기 함수가 발행된 경우, 이 그룹의 첫 번째 메시지가 찾아보기 커서 앞에 있기 때문에 그룹 123이 지금 완료되었다는 알림을 받지 못합니다.

일부 경우(예를 들어, 그룹 전체가 존재할 때 메시지가 검색 후 삭제되는 경우)에는 MQGMO_BROWSE_FIRST와 함께 MQGMO_ALL_MSGS_AVAILABLE을 사용할 수 있습니다. 그렇지 않으면, 찾아보기 스캔을 반복하여 새로 도착한 메시지 중 누락된 메시지를 알아내야 합니다. MQGMO_BROWSE_NEXT 및 MQGMO_ALL_MSGS_AVAILABLE과 함께 MQGMO_WAIT를 발행하는 것은 바로 이 점을 간과하는 것입니다. (이 문제는 메시지 스캐닝이 완료된 후에 도착하는 상위 우선순위 메시지에서도 발생합니다.)

다음 절에서는 세그먼트되지 않은 메시지를 처리하는 찾아보기 예제를 검토합니다. 세그먼트된 메시지도 이와 유사한 원리를 따릅니다.

그룹의 메시지 찾아보기

이 예제에서 애플리케이션은 논리 순서대로 큐의 각 메시지를 찾아봅니다.

큐의 메시지는 그룹화할 수 있습니다. 그룹화된 메시지의 경우, 애플리케이션은 해당 그룹의 모든 메시지가 도착할 때까지 그룹 처리를 시작하지 않습니다. 따라서 그룹의 첫 번째 메시지에 대해 MQGMO_ALL_MSGS_AVAILABLE이 지정되지만, 그룹의 후속 메시지에 대해서는 이 옵션이 불필요합니다.

이 예제에서는 MQGMO_WAIT이 사용됩니다. 그러나 819 페이지의 『논리 순서대로 메시지 찾아보기』의 이유 때문에 새 그룹이 도착하면 대기가 충족될 수 있지만, 찾아보기 커서가 이미 그룹의 첫 번째 논리 메시지를 전달한 경우에는 대기가 충족되지 않고 나머지 메시지가 그 즉시 도착합니다. 그럼에도 불구하고, 새 메시지 또는 세그먼트를 기다리는 동안 애플리케이션이 끊임없이 반복되지 않도록 하려면 적당한 간격 동안 대기해야 합니다.

MQGMO_LOGICAL_ORDER는 스캔이 논리 순서대로 수행되도록 하는 데 사용됩니다. 이는 각 그룹이 제거되기 때문에 그룹의 첫 번째 메시지(만)을 찾을 때는 MQGMO_LOGICAL_ORDER가 사용되지 않는 MQGET 후 삭제 예제와 대조를 보입니다.

메시지가 세그먼트되었는지 여부에 관계없이 애플리케이션 버퍼가 항상 전체 메시지를 보유할 만큼 크다고 가정합니다. 따라서 MQGMO_COMPLETE_MSG가 각 MQGET에 지정됩니다.

다음은 그룹에서 논리 메시지를 찾아보는 예를 보여줍니다.

```
/* Browse the first message in a group, or a message not in a group */
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
| MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT
MQGET GMO.MatchOptions = MQMO_MATCH_MSG_SEQ_NUMBER, MD.MsgSeqNumber = 1
/* Examine first or only message */
...

GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
    MQGET
    /* Examine each remaining message in the group */
    ...
```

그룹은 MQRC_NO_MSG_AVAILABLE이 리턴될 때까지 반복됩니다.

찾아보기 및 검색 후 삭제

이 예에서, 애플리케이션은 그룹을 파괴적으로 검색할지 여부를 결정하기 전에 그룹 내의 각 논리 메시지를 찾아봅니다.

이 예제의 첫 번째 부분은 이전 예제와 비슷합니다. 하지만 이 경우에는 전체 그룹을 찾아보고 돌아가서 해당 그룹을 검색한 후 삭제하기로 결정합니다.

이 예제에서 각 그룹이 제거된 것처럼, 그룹의 첫 번째 메시지(만)을 찾을 때는 MQGMO_LOGICAL_ORDER가 사용되지 않습니다.

다음은 찾아보고 검색 후 삭제하는 예를 보여줍니다.

```
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
| MQGMO_ALL_MESSAGES_AVAILABLE | MQGMO_WAIT
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
    MQGET
    /* Examine each remaining message in the group (or as many as
       necessary to decide whether to get it destructively) */
    ...

    if ( we want to retrieve the group destructively )

        if ( GroupStatus == ' ' )
            /* We retrieved an ungrouped message */
            GMO.Options = MQGMO_MSG_UNDER_CURSOR | MQGMO_SYNCPOINT
```



```

MQGET GMO.MatchOptions = 0
/* Process the message */
...

else
/* We retrieved one or more messages in a group. The browse cursor */
/* will not normally be still on the first in the group, so we have */
/* to match on the GroupId and MsgSeqNumber = 1. */
/* Another way, which works for both grouped and ungrouped messages, */
/* would be to remember the MsgId of the first message when it was */
/* browsed, and match on that. */
GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID
                    | MQMO_MATCH_MSG_SEQ_NUMBER,
                    (MQMD.GroupId = value already in the MD)
                    MQMD.MsgSeqNumber = 1
/* Process first or only message */
...

GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
                    | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
MQGET
/* Process each remaining message in the group */
...

```

찾아본 메시지의 반복 전달 방지

특정 열기 옵션 및 메시지 가져오기 옵션을 사용하면 메시지를 현재 또는 기타 협업 애플리케이션에서 다시 검색하지 않도록 찾아본 것으로 표시할 수 있습니다. 메시지는 다시 찾아볼 수 있도록 명시적으로 또는 자동으로 표시 해제할 수 있습니다.

큐에서 메시지를 찾아보는 경우, 해당 메시지를 가져온 후 삭제하는 경우에 검색하는 순서와는 다른 순서로 검색할 수 있습니다. 특히, 같은 메시지를 여러 번 찾아볼 수 있지만 큐에서 제거되면 이 작업을 수행할 수 없습니다. 이 문제를 해결하려면, 메시지를 찾아본 것으로 표시하고 표시된 메시지 검색을 방지할 수 있습니다. 이를 표시로 찾아보기라고도 합니다. 찾아본 메시지를 표시하려면 메시지 가져오기 옵션

MQGMO_MARK_BROWSE_HANDLE를 사용하고, 표시되지 않는 메시지만 검색하려면 MQGMO_UNMARKED_BROWSE_MSG를 사용하십시오. MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG 및 MQGMO_MARK_BROWSE_HANDLE 옵션의 조합을 사용하여 반복 MQGET을 발행하면 큐에서 각 메시지가 차례대로 검색됩니다. 이 경우 MQGMO_BROWSE_FIRST를 사용하여 메시지를 건너뛰지 못하게 하더라도 메시지가 반복해서 전달되지 않습니다. 이 옵션 조합은 단일 상수 MQGMO_BROWSE_HANDLE로 나타낼 수 있습니다. 큐에서 찾아보지 않은 메시지가 없을 때는 MQRC_NO_MSG_AVAILABLE이 리턴됩니다.

여러 애플리케이션에서 같은 큐를 찾아볼 경우 MQOO_CO_OP 및 MQOO_BROWSE 옵션으로 큐를 열 수 있습니다. 각 MQOPEN에 의해 리턴된 오브젝트 핸들은 협업 그룹의 일부인 것으로 간주됩니다. MQGMO_MARK_BROWSE_CO_OP 옵션을 지정하는 MQGET 호출에 의해 리턴된 모든 메시지는 이 협업 핸들 세트에 대해 표시된 것으로 간주됩니다.

메시지는 일정 시간 동안 표시된 후 큐 관리자에 의해 자동으로 표시 해제되어 다시 찾아볼 수 있게 됩니다. 큐 관리자 속성 MsgMarkBrowseInterval은 메시지가 협업 핸들 세트에 대해 표시 상태를 유지하는 시간(밀리초)을 제공합니다. MsgMarkBrowseInterval이 -1이면 메시지가 자동으로 표시 해제되지 않습니다.

메시지를 표시하는 단일 프로세스 또는 협업 프로세스 세트가 중지되면 모든 표시된 메시지가 표시 해제됩니다.

협업 찾아보기의 예

디스패처 애플리케이션의 여러 사본을 실행하여 큐에서 메시지를 찾아보고 각 메시지의 콘텐츠를 기반으로 이용자를 시작할 수 있습니다. 각 디스패처에서 MQOO_CO_OP로 큐를 여십시오. 그러면 디스패처가 협업하며 서로 표시된 메시지를 인식하게 됩니다. 그런 다음, 각 디스패처는 MQGET 호출을 반복하여

MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG 및 MQGMO_MARK_BROWSE_CO_OP 옵션을 지정합니다. (단일 상수 MQGMO_BROWSE_CO_OP를 사용하여 이러한 옵션 조합을 나타낼 수 있습니다.) 그런 다음, 각 디스패처 애플리케이션은 아직 다른 협업 디스패처에 의해 표시되지 않은 메시지만 검색합니다. 디스패처는 이용자를 초기화하고 MQGET에 의해 리턴된 MsgToken을 이용자에게 전달하며, 이용자는 큐에서 메시지를 가져온 후 삭제합니다. 이용자가 메시지의 MQGET을 백아웃하면 메시지가 더 이상 표시되지 않기 때문에 브라우저 중 하나에서 메시지를 다시 디스패치할 수 있습니다. 이용자가 메시지에 대해 MQGET을 수행하지 않으면, MsgMarkBrowseInterval이 지난 후 큐 관리자는 협업 핸들 세트에 대해 메시지를 표시 해제하고 다시 디스패치할 수 있습니다.

동일한 디스패처 애플리케이션의 다중 사본 대신에, 각각 큐의 메시지 서브세트를 처리하기에 적당하고 큐 찾아 보기에 사용되는 여러 다양한 디스패처 애플리케이션이 있을 수 있습니다. 각 디스패처에서 MQOO_CO_OP로 큐를 여십시오. 그러면 디스패처가 협업하며 서로 표시된 메시지를 인식하게 됩니다.

- 단일 디스패처의 메시지 처리 순서가 중요한 경우, 각 디스패처는 MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG 및 MQGMO_MARK_BROWSE_HANDLE(또는 MQGMO_BROWSE_HANDLE)을 지정하여 MQGET 호출을 반복합니다. 찾아본 메시지가 이 디스패처에서 처리하기에 적당하면 이전 MQGET 호출에 의해 리턴된 MQMO_MATCH_MSG_TOKEN, MQGMO_MARK_BROWSE_CO_OP 및 MsgToken을 지정하는 MQGET 호출을 수행합니다. 호출이 성공하면, 디스패처는 이용자를 초기화하고 MsgToken을 해당 이용자에게 전달합니다.
- 메시지 처리의 순서가 중요하지 않고 디스패처가 발견된 대부분의 메시지를 처리할 것으로 예상되면, MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG 및 MQGMO_MARK_BROWSE_CO_OP(또는 MQGMO_BROWSE_CO_OP)를 사용하십시오. 디스패처는 처리할 수 없는 메시지를 찾아볼 경우 이전에 리턴된 MQMO_MATCH_MSG_TOKEN, MQGMO_UNMARK_BROWSE_CO_OP 및 MsgToken 옵션으로 MQGET을 호출하여 메시지를 표시 해제합니다.

MQGET 호출이 실패하는 몇 가지 사례

MQOPEN 및 MQGET 호출 발행 사이에 FORCE 옵션을 명령에 사용하여 큐의 특정 속성이 변경되면, MQGET 호출이 실패하고 MQRC_OBJECT_CHANGED 이유 코드를 리턴합니다.

큐 관리자는 오브젝트 핸들을 더 이상 유효하지 않는 것으로 표시합니다. 이 문제는 큐 이름이 확인되는 큐에 변경사항이 적용되는 경우에도 발생합니다. 이런 식으로 핸들에 영향을 주는 속성은 MQOPEN의 MQOPEN 호출에 대한 설명에 나열되어 있습니다. 호출이 MQRC_OBJECT_CHANGED 이유 코드를 리턴하면, 큐를 닫았다가 다시 연 후 메시지를 다시 가져오십시오.

메시지 가져오기를 시도할 큐(또는 큐 이름이 확인되는 큐)에 대해 Get 조작이 금지되면, MQGET 호출은 실패하고 MQRC_GET_INHIBITED 이유 코드를 리턴합니다. 이 문제는 찾아보기에 MQGET 호출을 사용 중인 경우에도 발생합니다. 다른 프로그램이 큐의 속성을 주기적으로 변경하도록 애플리케이션이 설계된 경우에는 나중에 MQGET 호출을 시도하면 메시지를 정상적으로 가져올 수 있습니다.

동적 큐(임시 또는 영구)가 삭제된 경우, 이전에 확보된 오브젝트 핸들을 사용한 MQGET 호출은 실패하고 MQRC_Q_DELETED 이유 코드를 리턴합니다.

발행/구독 애플리케이션 작성

발행/구독 IBM MQ 애플리케이션 작성을 시작합니다.

발행/구독 개념에 대한 개요는 [발행/구독 메시징](#)을 참조하십시오.

서로 다른 유형의 발행/구독 애플리케이션 작성에 대한 정보는 다음 주제를 참조하십시오.

- [823 페이지의 『발행자 애플리케이션 작성』](#)
- [829 페이지의 『구독자 애플리케이션 작성』](#)
- [845 페이지의 『발행/구독 라이프사이클』](#)
- [849 페이지의 『발행/구독 메시지 특성』](#)
- [851 페이지의 『메시지 정렬』](#)
- [851 페이지의 『발행물 인터셉트』](#)
- [859 페이지의 『발행 옵션』](#)
- [859 페이지의 『구독 옵션』](#)

관련 개념

[6 페이지의 『애플리케이션 개발 개념』](#)

사용자는 원하는 절차적 또는 객체 지향 언어를 사용하여 IBM MQ 애플리케이션을 작성할 수 있습니다. IBM MQ 애플리케이션을 디자인하고 작성하기 전에 기본 IBM MQ 개념을 숙지하십시오.

[5 페이지의 『IBM MQ용 애플리케이션 개발』](#)

메시지를 송신하고 수신하며, 큐 관리자와 관련 자원을 관리하기 위한 애플리케이션을 개발할 수 있습니다. IBM MQ는 많은 다양한 언어와 프레임워크로 작성된 애플리케이션을 지원합니다.

[53 페이지의 『IBM MQ 애플리케이션에 대한 설계 고려사항』](#)

애플리케이션에서 사용 가능한 플랫폼과 환경을 이용할 수 있는 방법을 결정한 경우 IBM MQ에서 제공한 기능의 사용 방법을 결정해야 합니다.

743 페이지의 『큐잉을 위한 프로시저 애플리케이션 작성』

이 정보를 사용하여 큐잉 애플리케이션 작성, 큐 관리자에 연결 및 연결 끊기, 발행/구독 및 오브젝트 열기 및 닫기에 대해 알아보십시오.

917 페이지의 『클라이언트 프로시저 애플리케이션 작성』

프로시저 언어를 사용하여 IBM MQ에서 클라이언트 애플리케이션을 작성할 때 알아야 할 사항입니다.

937 페이지의 『프로시저 애플리케이션 빌드』

여러 프로시저 언어 중 하나로 IBM MQ 애플리케이션을 작성하고 여러 다른 플랫폼에서 애플리케이션을 실행할 수 있습니다.

1133 페이지의 『절차에 따른 프로그램 오류 핸들링』

이 정보는 호출할 때 또는 메시지를 최종 목적지에 전달할 때 애플리케이션 MQI 호출과 관련된 오류를 설명합니다.

관련 태스크

971 페이지의 『멀티플랫폼용 샘플 절차적 프로그램 사용』

이 샘플 프로시저 프로그램은 제품과 함께 전달됩니다. 샘플은 C 및 COBOL로 작성되며 MQI(Message Queue Interface)의 일반 사용을 보여줍니다.

1079 페이지의 『다음의 샘플 절차적 프로그램을 사용하여 z/OS』

IBM MQ for z/OS와 함께 전달된 샘플 프로시저 애플리케이션은 MQI(Message Queue Interface)의 일반적인 사용에 대해 설명합니다.

발행자 애플리케이션 작성

두 가지 예제를 검토하여 발행자 애플리케이션을 작성하는 작업을 시작합니다. 첫 번째 예제는 큐에 메시지를 넣는 포인트-투-포인트 애플리케이션에 가능한 한 가깝게 모델화되었고, 두 번째 예제는 동적으로 토픽을 작성하는 방법(즉, 발행자 애플리케이션에 대한 보다 일반적인 패턴)에 대해 설명합니다.

단순 IBM MQ 발행자 애플리케이션을 작성하는 것은 메시지를 큐(823 페이지의 표 136)에 넣는 IBM MQ 포인트-투-포인트 애플리케이션을 작성하는 것과 같습니다. 차이점은 큐가 아닌 토픽에 메시지를 MQPUT한다는 것입니다.

표 136. 포인트-투-포인트 대 발행/구독 IBM MQ 프로그램 패턴		
단계	포인트-투-포인트 MQ 호출	MQ 호출 발행
큐 관리자에 연결	MQCONN	MQCONN
큐 열기	MQOPEN	
토픽 열기		MQOPEN
메시지 넣기	MQPUT	MQPUT
토픽 닫기		MQCLOSE
큐 닫기	MQCLOSE	
큐 관리자와의 연결 끊기	MQDISC	MQDISC

이를 구체적으로 설명하기 위해 판매 품목 가격을 발행하는 애플리케이션의 두 가지 예제가 있습니다. 큐에 메시지를 넣는 작업과 근접하게 모델화된 첫 번째 예제(824 페이지의 『예제 1: 고정 토픽에 대한 발행자』)에서 관리자는 큐 작성과 유사한 방법으로 토픽 정의를 작성합니다. 프로그래머는 메시지를 큐에 쓰지 않고 토픽에 쓰도록 MQPUT을 코딩합니다. 두 번째 예제(826 페이지의 『예제 2: 가변 토픽에 대한 발행자』)에서 IBM MQ와 프로그램의 상호작용 패턴은 비슷합니다. 차이는 관리자 대신 프로그래머가 메시지가 기록된 토픽을 제공하는 데 있습니다. 실제로 이는 일반적으로 토픽 문자열이 콘텐츠 정의되거나 다른 소스(예: 브라우저를 통한 사용자 입력)에 의해 제공된다는 것을 의미합니다.

관련 개념

829 페이지의 『구독자 애플리케이션 작성』

세 가지 예제 즉, 큐의 메시지를 이용하는 IBM MQ 애플리케이션, 큐잉에 대한 지식 없이 구독을 작성하는 애플리케이션, 그리고 마지막으로 큐잉과 구독을 모두 사용하는 예제를 검토하여 구독자 애플리케이션 작성을 시작합니다.

관련 참조

DEFINE TOPIC

[디스플레이토픽](#)

[디스플레이 상태 표시](#)

예제 1: 고정 토픽에 대한 발행자

관리적으로 정의된 토픽에 대한 발행을 설명하는 IBM MQ 프로그램입니다.

참고: 압축 코딩 스타일은 프로덕션용이 아닌 읽기용입니다.

[825 페이지의 그림 95](#)의 출력을 참조하십시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char    topicNameDefault[]    = "IBMSTOCKPRICE";
    char    publicationDefault[]  = "129";
    MQCHAR48 qmName = "";

    MQHCONN Hconn = MQHC_UNUSABLE_HCONN; /* connection handle */
    MQHOBJ  Hobj  = MQHO_NONE;           /* object handle sub queue */
    MQLONG  CompCode = MQCC_OK;          /* completion code */
    MQLONG  Reason = MQRC_NONE;          /* reason code */
    MQOD    td = {MQOD_DEFAULT};        /* Object descriptor */
    MQMD    md = {MQMD_DEFAULT};        /* Message Descriptor */
    MQPMO    pmo = {MQPMO_DEFAULT};     /* put message options */
    MQCHAR    resTopicStr[151];          /* Returned vale of topic string */
    char *    topicName = topicNameDefault;
    char *    publication = publicationDefault;
    memset    (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){
        /* replace defaults with args if provided */
        default:
            publication = argv[2];
        case(2):
            topicName = argv[1];
        case(1):
            printf("Optional parameters:  TopicObject Publication\n");
    }
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;      /* Object is a topic */
        td.Version = MQOD_VERSION_4;     /* Descriptor needs to be V4 */
        strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode,
        &Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" using topic \"%s\" to topic string \"%s\"\n",
            publication, td.ObjectName, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

그림 94. 고정 토픽에 대한 단순 IBM MQ 발행자

```
X:\Publish1\Debug>PublishStock
Optional parameters: TopicObject Publication
Published "129" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish1\Debug>PublishStock IBMSTOCKPRICE 155
Optional parameters: TopicObject Publication
Published "155" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

그림 95. 첫 번째 발행자의 샘플 출력의 예

다음의 선택된 코드 행은 IBM MQ에 대한 발행자 애플리케이션을 작성하는 측면을 보여줍니다.

```
char topicNameDefault[] = "IBMSTOCKPRICE";
```

기본 토픽 이름은 프로그램에 정의되어 있습니다. 다른 토픽 오브젝트의 이름을 프로그램에 대한 첫 번째 인수로 제공하여 대체할 수 있습니다.

```
MQCHAR resTopicStr[151];
```

resTopicStr은 td.ResObjectString.VSPtr에 의해 지정되고 MQOPEN에서 해석된 토픽 문자열을 리턴하는 데 사용됩니다. 널 종료용 공간을 제공하기 위해 resTopicStr의 길이를 td.ResObjectString.VSBufSize에 전달된 길이보다 1만큼 크게 설정하십시오.

```
memset (resTopicStr, 0, sizeof(resTopicStr));
```

resTopicStr을 널로 초기화하여 MQCHARV에 리턴되는 해석된 토픽 문자열이 널(Null) 종료되었는지 확인합니다.

```
td.ObjectType = MQOT_TOPIC
```

새 유형의 발행/구독 오브젝트 즉, 토픽 오브젝트가 있습니다.

```
td.Version = MQOD_VERSION_4;
```

새 유형의 오브젝트를 사용하려면 최소한 버전 4의 오브젝트 디스크립터를 사용해야 합니다.

```
strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
```

topicName은 토픽 오브젝트의 이름이며 관리 토픽 오브젝트라고도 합니다. 예제에서 토픽 오브젝트는 IBM MQ Explorer 또는 이 MQSC 명령을 사용하여 미리 작성해야 합니다.

```
DEFINE TOPIC(IBMSTOCKPRICE) TOPICSTR(NYSE/IBM/PRICE) REPLACE;
```

```
td.ResObjectString.VSPtr = resTopicStr;
```

해석된 토픽 문자열은 프로그램의 최종 printf에서 반환됩니다. 'IBM MQ'에 대한 'MQCHARV ResObjectString 구조를 설정하여 해석된 문자열을 프로그램에 다시 반환합니다.

```
MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
```

출력할 큐를 여는 것처럼 출력할 토픽을 엽니다.

```
pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
```

발행자에 MQPMO_RETAIN을 지정하여 새 구독자가 발행물을 수신할 수 있도록 합니다. 구독자를 시작하면, 구독자는 구독자가 시작되기 전에 발행된 최신 발행물을 첫 번째 일치 발행물로 수신합니다. 대안은 구독자가 시작된 후에 발행된 발행물만 구독자에게 제공하는 것입니다. 또한 구독자는 해당 구독에 MQSO_NEW_PUBLICATIONS_ONLY를 지정하여 보유된 발행물을 수신하는 것을 거절할 수 있습니다.

```
MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
```

MQPUT에 전달된 문자열의 길이에 1을 더하여 널 종료 문자를 IBM MQ에 메시지 버퍼의 일부로 전달합니다.

두 번째 예제에서 설명하는 내용은 무엇입니까? 이 예제는 포인트-투-포인트 IBM MQ 프로그램 작성에 사용되는 검증된 일반 패턴을 최대한 비슷하게 모방한 것입니다. IBM MQ 프로그래밍 패턴의 중요한 특징은 프로그래머가 메시지 송신 위치와 관련이 없는 점입니다. 프로그래머의 태스크는 큐 관리자에 연결하고 수신인에게 분배할 메시지를 큐 관리자에 전달하는 것입니다. 포인트-투-포인트 패러다임에서 프로그래머는 관리자가 구성한 큐(대개는 알리어스 큐)를 엽니다. 알리어스 큐는 대상 큐의 로컬 큐 관리자 또는 리모트 큐 관리자로 메시지를 라우팅합니다. 메시지가 전달 대기 중인 동안에는 소스와 목적지 사이에 있는 큐에 저장됩니다.

발행/구독 패턴에서 프로그래머는 큐를 열지 않고 토픽을 엽니다. 이 예제에서 관리자는 토픽을 토픽 문자열과 연관시킵니다. 큐 관리자는 큐를 사용해서 발행물의 토픽 문자열과 일치하는 구독을 가진 로컬 또는 리모트 구독

자에게 발행물을 전달합니다. 발행물이 보유된 경우, 큐 관리자는 지금 구독자가 없어도 발행물의 최신 사본을 보관합니다. 보유된 발행물은 미래의 구독자에게 전달할 수 있습니다. 발행자 애플리케이션은 발행물을 선택하거나 목적지로 라우팅함에 있어 아무런 역할도 하지 않지만, 해당 태스크는 발행물을 작성하여 관리자가 정의한 토픽에 넣는 것입니다.

이런 고정 토픽 예제는 여러 발행/구독 애플리케이션의 비전형적 사례입니다. 따라서 관리자는 토픽 문자열을 정의하고 발행된 토픽을 변경해야 합니다. 일반적으로 발행/구독 애플리케이션은 일부 또는 전체 토픽 트리를 알아야 합니다. 어쩌면 토픽이 자주 변경되거나, 토픽이 크게 변경되지 않더라도 토픽 조합 수가 크면 발행해야 할 모든 토픽 문자열에 대해 관리자가 토픽 노드를 정의하기가 너무 부담됩니다. 어쩌면 발행 전에는 토픽 문자열을 모를 수도 있으므로, 발행자 애플리케이션은 발행 콘텐츠의 정보를 사용하여 토픽 문자열을 지정할 수도 있고 다른 소스(예: 브라우저에서 사용자 입력)에서 발행하는 토픽 문자열에 대한 정보가 있을 수 있습니다. 보다 동적인 발행 스타일의 요구를 충족시키기 위해 다음 예제에서는 토픽을 발행자 애플리케이션의 일부로서 동적으로 작성하는 방법을 보여 줍니다.

토픽은 발행자와 구독자를 함께 연결합니다. 토픽 이름을 지정하고 토픽 트리에서 토픽을 구성하기 위한 아키텍처 또는 규칙 설계는 발행/구독 솔루션을 개발함에 있어 중요한 단계입니다. 토픽 트리 조직에서 발행자 및 구독자 프로그램을 함께 바인딩하는 범위를 주의깊게 살펴보고 토픽 트리의 콘텐츠에 해당 프로그램을 바인딩합니다. 토픽 트리의 변경사항이 발행자 및 구독자 애플리케이션에 영향을 미치는지 여부와 영향을 최소화할 수 있는 방법에 대해 스스로 자문해 보십시오. 토픽의 루트 부분이나 루트 서브트리를 제공하는 관리 토픽 오브젝트의 개념이 IBM MQ 발행/구독 모델의 아키텍처에 빌드됩니다. 토픽 오브젝트는 애플리케이션 프로그래밍 및 조작을 단순화해서 유지보수성을 향상시키는 토픽 트리의 루트 부분을 관리 면에서 정의하는 옵션을 제공합니다. 예를 들어, 토픽 트리가 격리되어 있는 여러 발행/구독 애플리케이션을 배치하는 경우 토픽 트리의 루트 부분을 관리 면에서 정의하면 다른 애플리케이션이 채택한 토픽 이름 지정 규칙에 일관성이 없어도 토픽 트리의 격리를 보장할 수 있습니다.

실제로 발행자 애플리케이션의 범위에는 이 예제에서의 고정 토픽 단독 사용 및 다음 예제에서의 가변 토픽 단독 사용이 포함됩니다. [826 페이지의 『예제 2: 가변 토픽에 대한 발행자』](#)에서는 토픽과 토픽 문자열의 사용을 결합하는 방법에 대해서도 설명합니다.

관련 개념

[826 페이지의 『예제 2: 가변 토픽에 대한 발행자』](#)

프로그래밍 방식으로 정의된 토픽 발행에 대해 설명하는 WebSphere MQ 프로그램입니다.

[829 페이지의 『구독자 애플리케이션 작성』](#)

세 가지 예제 즉, 큐의 메시지를 이용하는 IBM MQ 애플리케이션, 큐잉에 대한 지식 없이 구독을 작성하는 애플리케이션, 그리고 마지막으로 큐잉과 구독을 모두 사용하는 예제를 검토하여 구독자 애플리케이션 작성을 시작합니다.

예제 2: 가변 토픽에 대한 발행자

프로그래밍 방식으로 정의된 토픽 발행에 대해 설명하는 WebSphere MQ 프로그램입니다.

참고: 압축 코딩 스타일은 프로덕션용이 아닌 읽기용입니다.

827 페이지의 그림 97에서 출력을 확인하십시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char    topicNameDefault[] = "STOCKS";
    char    topicStringDefault[] = "IBM/PRICE";
    char    publicationDefault[] = "130";
    MQCHAR48 qmName = "";

    MQHCONN Hconn = MQHC_UNUSABLE_HCONN; /* connection handle */
    MQHOBJ  Hobj  = MQHO_NONE;           /* object handle sub queue */
    MQLONG  CompCode = MQCC_OK;          /* completion code */
    MQLONG  Reason = MQRC_NONE;          /* reason code */
    MQOD    td = {MQOD_DEFAULT};        /* Object descriptor */
    MQMD    md = {MQMD_DEFAULT};        /* Message Descriptor */
    MQPMO    pmo = {MQPMO_DEFAULT};     /* put message options */
    MQCHAR    resTopicStr[151];          /* Returned value of topic string */
    char *    topicName = topicNameDefault;
    char *    topicString = topicStringDefault;
    char *    publication = publicationDefault;
    memset (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){
        /* Replace defaults with args if provided */
        default:
            publication = argv[3];
        case(3):
            topicString = argv[2];
        case(2):
            if (strcmp(argv[1],"/")) /* "/" invalid = No topic object */
                topicName = argv[1];
            else
                *topicName = '\0';
        case(1):
            printf("Provide parameters: TopicObject TopicString Publication\n");
    }

    printf("Publish \"%s\" to topic \"%-48s\" and topic string \"%s\"\\n", publication, topicName,
topicString);
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC; /* Object is a topic */
        td.Version = MQOD_VERSION_4; /* Descriptor needs to be V4 */
        strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        td.ObjectString.VSPtr = topicString;
        td.ObjectString.VSLength = (MQLONG)strlen(topicString);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" to topic string \"%s\"\\n", publication, resTopicStr);
    printf("Completion code %d and Return code %d\\n", CompCode, Reason);
}
```

그림 96. 가변 토픽에 대한 단순 IBM MQ 발행자

```
X:\Publish2\Debug>PublishStock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish2\Debug>PublishStock / NYSE/IBM/PRICE 131
Provide parameters: TopicObject TopicString Publication
Publish "131" to topic "" and topic string "NYSE/IBM/PRICE"
Published "131" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

그림 97. 두 번째 발행자의 샘플 출력의 예

이 예제에 대한 몇 가지 참고사항이 있습니다.

char topicNameDefault[] = "STOCKS";

기본 토픽 이름 STOCKS는 토픽 문자열의 일부를 정의합니다. 이 토픽 이름을 프로그램에 대한 첫 번째 인수로 제공하여 대체하거나, /를 첫 번째 매개변수로 제공하여 토픽 이름을 사용하지 않을 수 있습니다.

char topicString[101] = "IBM/PRICE";

IBM/PRICE가 기본 토픽 문자열입니다. 이 토픽 문자열은 두 번째 인수로 프로그램에 제공하여 대체할 수 있습니다.

큐 관리자는 STOCKS 토픽 오브젝트, "NYSE"가 제공하는 토픽 문자열을 "IBM/PRICE" 프로그램이 제공하는 토픽 문자열과 결합하고 두 토픽 문자열 사이에 "/"를 삽입합니다. 결과는 해석된 토픽 문자열 "NYSE/IBM/PRICE"입니다. 결과로 얻어지는 토픽 문자열은 IBMSTOCKPRICE 토픽 오브젝트에 정의된 토픽 문자열과 동일하며 똑같은 효과가 있습니다.

해석된 토픽 문자열과 연관된 관리 토픽 오브젝트는 발행자가 MQOPEN에 전달한 토픽 오브젝트와 다를 수도 있습니다. IBM MQ는 해석된 토픽 문자열에 암시된 트리를 사용하여 발행과 연관된 속성을 정의하는 관리 토픽 오브젝트를 확인합니다.

두 개의 토픽 객체 'A'와 'B'가 있고 'A' 토픽 'a'를 정의하고 'B' 토픽 'a/b' ('828 페이지의 그림 98)을 정의한다고 가정해 보겠습니다. 퍼블리셔 프로그램이 토픽 객체 'A'를 참조하고 토픽 문자열 'b'를 제공하여 토픽을 토픽 문자열 'a/b'으로 해석하는 경우 토픽이 'B'에 대해 정의된 토픽 문자열 'a/b'와 일치하므로 발행물은 토픽 객체 'B'에서 그 속성을 상속받습니다.

if (strcmp(argv[1],"/"))

argv[1]은 선택적으로 제공된 topicName입니다. "/"는 토픽 이름으로 올바르지 않으며 여기에서는 토픽 이름이 없음을 의미하고, 토픽 문자열은 전적으로 프로그램에 의해 제공됩니다. 827 페이지의 그림 97의 출력은 프로그램에서 동적으로 제공되는 전체 토픽 문자열을 표시합니다.

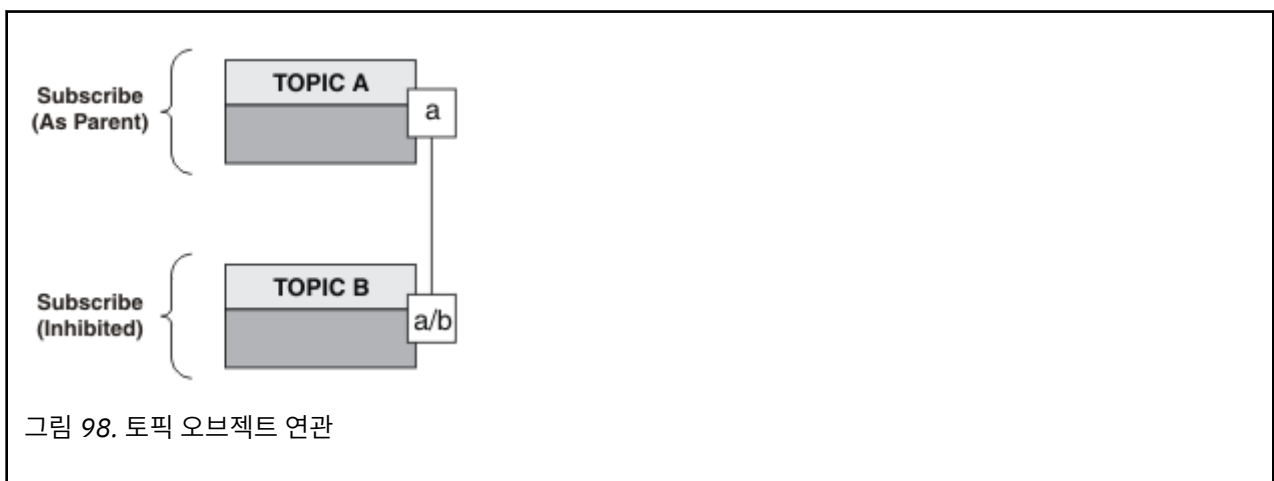
strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);

기본값의 경우, 'IBM MQ 탐색기 또는 이 MQSC 명령을 사용하여 선택적 'topicName'을 미리 만들어야 합니다:

```
DEFINE TOPIC(STOCKS) TOPICSTR(NYSE) REPLACE;
```

td.ObjectString.VSPtr = topicString;

토픽 문자열은 토픽 디스크립터의 MQCHARV 필드입니다.



두 번째 예제에서 설명하는 내용은 무엇입니까? 코드는 첫 번째 예제와 매우 유사하지만(사실상 두 행의 차이만 있음) 결과는 첫 번째 예제와는 많이 다른 프로그램입니다. 프로그래머는 보낸 발행물의 목적지를 제어합니다. 구독자 애플리케이션 설계에 있어 관리자가 최소한으로 입력해야 하며, 발행자로부터 구독자에게로 발행물을 라우팅하도록 토픽 또는 큐를 사전정의하지 않아도 됩니다.

포인트-투-포인트 메시징 패러다임에서 메시지 플로우를 실행하려면 먼저 큐를 정의해야 합니다. 발행/구독의 경우에는 큐를 정의하지 않아도 IBM MQ가 기본 큐잉 시스템을 사용하여 발행/구독을 구현하고, 메시징 및 큐잉과 연관된 전달, 트랜잭션성 및 느슨한 결합이 보장되는 이점이 발행/구독 애플리케이션에 의해 상속됩니다.

디자이너는 발행자 및 구독자 프로그램이 기본 토픽 트리를 인식해야 하는지 여부와 구독자 프로그램이 큐잉을 인식해야 하는지 여부를 결정해야 합니다. 다음에는 구독자 예제 애플리케이션을 학습하십시오. 이 애플리케이션은 발행자 예제(일반적으로 NYSE/IBM/PRICE 발행 및 구독)와 함께 사용하도록 설계되어 있습니다.

관련 개념

824 페이지의 『예제 1: 고정 토픽에 대한 발행자』

관리적으로 정의된 토픽에 대한 발행을 설명하는 IBM MQ 프로그램입니다.

829 페이지의 『구독자 애플리케이션 작성』

세 가지 예제 즉, 큐의 메시지를 이용하는 IBM MQ 애플리케이션, 큐잉에 대한 지식 없이 구독을 작성하는 애플리케이션, 그리고 마지막으로 큐잉과 구독을 모두 사용하는 예제를 검토하여 구독자 애플리케이션 작성을 시작합니다.

구독자 애플리케이션 작성

세 가지 예제 즉, 큐의 메시지를 이용하는 IBM MQ 애플리케이션, 큐잉에 대한 지식 없이 구독을 작성하는 애플리케이션, 그리고 마지막으로 큐잉과 구독을 모두 사용하는 예제를 검토하여 구독자 애플리케이션 작성을 시작합니다.

829 페이지의 표 137에는 이용자 또는 구독자의 세 가지 스타일과 해당 특성을 나타내는 IBM MQ 함수 호출의 순서가 나열되어 있습니다.

1. 첫 번째 스타일로 MQ 발행 이용자는 MQGET만 수행하는 포인트-투-포인트 MQ 프로그램과 동일합니다. 애플리케이션은 발행물을 이용 중이라는 사실 즉, 단순히 큐에서 메시지를 읽고 있는 것을 알지 못합니다. 관리면에서 IBM MQ Explorer 또는 명령을 사용하여 발행물이 큐로 라우팅되도록 하는 구독을 작성합니다.
2. 두 번째 스타일은 대부분의 구독자 애플리케이션에서 선호되는 패턴입니다. 구독자 애플리케이션은 구독을 작성한 후 발행물을 가져옵니다. 큐 관리는 모두 큐 관리자에 의해 수행됩니다. 이를 관리되는 구독자라고 합니다.
3. 세 번째 스타일에서 구독자 애플리케이션은 발행물을 보유하는 데 사용할 큐를 지정하고 해당 큐를 열고 닫고 발행물로 큐를 채우도록 구독을 발행합니다. 이를 관리되지 않는 구독자라고 합니다.

이러한 스타일을 이해하기 위한 한 가지 방법은 각 스타일에 대해 829 페이지의 표 137에 나열된 C 프로그램 예제를 검토하는 것입니다. 예제는 823 페이지의 『발행자 애플리케이션 작성』에 있는 발행자 예제와 함께 실행되도록 설계되었습니다.

표 137. 포인트-투-포인트 대 구독 IBM MQ 프로그램 패턴				
단계	MQ 메시지 이용자	830 페이지의 『예제 1: MQ 발행 이용자』	832 페이지의 『예 제 2: 관리되는 MQ 구독자』	837 페이지의 『예제 3: 관리되지 않는 MQ 구독자』
큐 관리자에 연결	MQCONN	MQCONN	MQCONN	MQCONN
큐 열기	MQOPEN	MQOPEN		MQOPEN
구독			MQSUB	MQSUB
메시지 가져오기	MQGET	MQGET	MQGET	MQGET
큐 닫기	MQCLOSE	MQCLOSE	(MQCLOSE)	MQCLOSE
구독 닫기			MQCLOSE	MQCLOSE
큐 관리자와의 연결 끊 기	MQDISC	MQDISC	MQDISC	MQDISC

MQCLOSE 사용은 항상 선택사항입니다. 자원을 해제하기 위해 MQCLOSE 옵션을 전달하거나 단순히 MQOPEN에 대한 대칭으로 사용됩니다. 관리되는 MQ 구독자 사례에서는 구독 큐가 처리완료될 때 MQCLOSE 옵션을 지정할 필요가 없고 대칭 인수도 관련이 없기 때문에, 예제 2: 관리되는 MQ 구독자에서는 구독 큐가 명시적으로 처리완료되지 않습니다.

발행/구독 애플리케이션 패턴을 이해하기 위한 또 다른 방법은 관련된 여러 엔티티 사이의 상호작용을 검토하는 것입니다. 라이프라인 또는 UML 순서 다이어그램은 상호작용을 검토할 수 있는 좋은 방법입니다. 세 가지 라이프라인 예제는 845 페이지의 『발행/구독 라이프사이클』에서 설명합니다.

예제 1: MQ 발행 이용자

MQ 발행 이용자는 토픽 자체를 구독하지 않는 IBM MQ 메시지 이용자입니다.

이 예제의 구독 및 발행 큐를 작성하려면 다음 명령을 실행하거나 IBM MQ Explorer를 사용하여 오브젝트를 정의하십시오.

```
DEFINE QLOCAL(STOCKTICKER) REPLACE;  
DEFINE SUB(IBMSTOCKPRICESUB) DEST(STOCKTICKER) TOPICOBJ(IBMSTOCKPRICE) REPLACE;
```

IBMSTOCKPRICESUB 구독은 발행자 예제에 대해 작성된 IBMSTOCK 토픽 오브젝트 및 로컬 큐 STOCKTICKER를 참조합니다. 토픽 오브젝트 IBMSTOCK은 구독에서 사용되는 토픽 문자열 NYSE/IBM/PRICE를 정의합니다. 발행물을 수신하는 데 사용되는 큐와 토픽 오브젝트는 구독을 작성하기 전에 정의해야 합니다.

MQ 발행 이용자 패턴에 중요한 여러 패킷이 있습니다.

1. 멀티프로세싱: 발행물 읽기 작업의 공유. 발행물은 모두 구독 토픽과 연관된 단일 큐로 이동합니다. 여러 이용자가 MQ00_INPUT_SHARED를 사용하여 큐를 열 수 있습니다.
2. 중앙 집중식으로 관리되는 구독. 애플리케이션은 자체 구독 토픽 또는 구독을 구성하지 않으며, 관리자는 발행물이 송신되는 위치를 담당합니다.
3. 구독 집중: 여러 다른 구독을 단일 큐에 송신할 수 있습니다.
4. 구독 지속 가능성: 큐는 이용자가 활성 상태인지 여부에 관계없이 모든 발행물을 수신합니다.
5. 마이그레이션 및 공존: 이용자 코드는 포인트-투-포인트 및 발행/구독 시나리오에서 동일하게 잘 작동합니다.

구독은 토픽 문자열 NYSE/IBM/PRICE와 큐 STOCKTICKER 사이의 관계를 작성합니다. 발행물(현재 보유한 모든 발행물 포함)은 구독이 작성되는 순간부터 STOCKTICKER로 전달됩니다.

관리상으로 작성된 구독은 관리되거나 관리되지 않을 수 있습니다. 관리되는 구독은 관리되지 않는 구독과 같이, 작성되는 즉시 적용됩니다. 모든 패턴 패킷이 관리되는 구독에서 사용 가능하지는 않습니다. [837 페이지의 『예제 3: 관리되지 않는 MQ 구독자』](#)의 내용을 참조하십시오.

참고: 압축 코딩 스타일은 프로덕션용이 아닌 읽기용입니다.

결과는 831 페이지의 그림 100에 표시됩니다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    MQCHAR    publicationBuffer[101];
    MQCHAR48  subscriptionQueueDefault = "STOCKTICKER";
    MQCHAR48  qmName = "";          /* Use default queue manager */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN;    /* connection handle */
    MQHOBJ   Hobj = MQHO_NONE;               /* object handle sub queue */
    MQLONG   CompCode = MQCC_OK;             /* completion code */
    MQLONG   Reason = MQRC_NONE;             /* reason code */
    MQLONG   messlen = 0;
    MQOD     od = {MQOD_DEFAULT};           /* Unmanaged subscription queue */
    MQMD     md = {MQMD_DEFAULT};           /* Message Descriptor */
    MQGMO     gmo = {MQGMO_DEFAULT};        /* Get message options */
    char *    publication=publicationBuffer;
    char *    subscriptionQueue = subscriptionQueueDefault;

    switch(argc){                          /* Replace defaults with args if provided */
    default:
        subscriptionQueue = argv[1]
    case(1):
        printf("Optional parameter: subscriptionQueue\n");
    }

    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
        MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        printf("Waiting %d seconds for publications from %s\n", gmo.WaitInterval/1000,
            subscriptionQueue);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            memset(publication, 0, sizeof(publicationBuffer));
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen,
                &CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        } while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

그림 99. MQ 발행 이용자

```
X:\Subscribe1\Debug>Subscribe1
Optional parameter: subscriptionQueue
Waiting 10 seconds for publications from STOCKTICKER
Received publication "129"
Completion code 0 and Return code 0
```

그림 100. MQ 발행 이용자의 출력

유의해야 할 몇 가지 표준 IBM MQ C 언어 프로그래밍 팁이 있습니다.

memset(publication, 0, sizeof(publicationBuffer));

printf를 사용하여 쉽게 형식화할 경우 메시지에 후행 널이 있는지 확인합니다. 발행자 예제는 strlen(publication)에 1을 추가하여 MQPUT에 전달된 메시지 버퍼에 후행 널을 포함합니다. MQCHAR 버퍼를 널로 설정하는 것은 버퍼를 사용하여 문자열을 저장하는 IBM MQ C 프로그램에 적절한 프로그래밍 스타일이며, 버퍼를 완전히 채우지 못한 문자 배열 뒤에는 널이 따릅니다.

MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen, &CompCode, &Reason);

메시지 버퍼의 끝에 하나의 널을 예약하여, if (messlen == strlen(publication));이 true인 경우 리턴된 메시지에 후행 널이 있는지 확인합니다. 이 팁은 선행 팁 정보를 보완하며, publication의 콘텐츠로 덮여지지 않은 publicationBuffer에 최소 하나의 널이 있는지 확인합니다.

관련 개념

832 페이지의 『예제 2: 관리되는 MQ 구독자』

관리되는 MQ 구독자는 대부분의 구독자 애플리케이션에서 자주 사용하는 패턴입니다. IBM MQ은(는) 관리되는 구독을 통해 구독을 처리하고 등록 및 등록 취소를 수행합니다. 예제에서는 큐, 토픽 또는 구독의 관리 정의가 필요하지 않습니다.

837 페이지의 『예제 3: 관리되지 않는 MQ 구독자』

관리되지 않는 구독자는 구독자 애플리케이션의 중요한 클래스입니다. 이 이 구독자의 경우 발행물의 큐잉 및 이용 제어를 통해 발행/구독의 이점을 결합합니다. 관리되지 않는 구독은 애플리케이션이 책임을 지는 구독입니다. 를 사용하여 구독이 저장되는 대기열을 지정할 수 있습니다. 예제에서는 구독과 큐를 결합하는 여러 가지 방법을 보여줍니다.

823 페이지의 『발행자 애플리케이션 작성』

두 가지 예제를 검토하여 발행자 애플리케이션을 작성하는 작업을 시작합니다. 첫 번째 예제는 큐에 메시지를 넣는 포인트 투 포인트 애플리케이션에 가능한 한 가깝게 모델화되었고, 두 번째 예제는 동적으로 토픽을 작성하는 방법(즉, 발행자 애플리케이션에 대한 보다 일반적인 패턴)에 대해 설명합니다.

예제 2: 관리되는 MQ 구독자

관리되는 MQ 구독자는 대부분의 구독자 애플리케이션에서 자주 사용하는 패턴입니다. IBM MQ은(는) 관리되는 구독을 통해 구독을 처리하고 등록 및 등록 취소를 수행합니다. 예제에서는 큐, 토픽 또는 구독의 관리 정의가 필요하지 않습니다.

이런 가장 단순한 종류의 관리되는 구독자는 일반적으로 지속 불가능 구독을 사용합니다. 예제에서는 지속 불가능 구독에 초점을 맞추고 있습니다. 구독은 MQSUB에서 구독 핸들의 활성 시간 동안만 지속됩니다. 구독의 활성 시간 동안 토픽 문자열과 일치하는 모든 발행물이 구독 큐로 송신됩니다. (그리고 가능한 경우, 발행물이 작성된 이후로 MQSO_NEW_PUBLICATIONS_ONLY 플래그가 설정 또는 기본 설정되지 않았고 토픽 문자열과 일치하는 이전 발행물이 보유되었으며 발행물이 지속적이거나 큐 관리자가 종료되지 않은 경우에는 보유된 발행물도 송신됩니다.)

또한 이 패턴으로 지속 가능 구독을 사용할 수도 있습니다. 일반적으로 관리되는 지속 가능 구독을 사용하는 경우, 오류 발생 없이 구독자보다 더 오래 지속되는 구독을 설정하지 않도록 신뢰성 확보 차원에서 완료됩니다. 관리, 비관리, 지속 가능 및 지속 불가능 구독과 연관되는 다양한 라이프사이클에 대한 자세한 정보는 관련 항목 절을 참조하십시오.

지속 가능 구독은 지속 발행물과 연관되고 지속 불가능 구독은 비지속 발행물과 연관되는 경우가 많지만, 구독 지속 가능성과 발행 지속성 사이에 필요한 관계는 없습니다. 지속성과 지속 가능성의 4가지 조합이 모두 가능합니다.

큐 관리자는 관리되는 지속 불가능 사례를 고려하여 큐가 처리완료될 때 영구 제거되고 삭제되는 구독 큐를 작성합니다. 지속 불가능 구독이 처리완료되면 발행물이 큐에서 제거됩니다.

이 코드로 예시된 관리되는 지속 불가능 패턴의 중요한 패킷은 다음과 같습니다.

1. 요청 시 구독: 구독 토픽 문자열이 동적입니다. 애플리케이션 실행 시 애플리케이션이 이런 문자열을 제공합니다.
2. 자체 관리 큐: 구독 큐는 자체 정의하고 관리합니다.
3. 자체 관리 구독 라이프사이클: 지속 불가능 구독은 구독자 애플리케이션의 지속 기간 동안에만 존재합니다.
 - 지속 가능한 관리되는 구독을 정의하면 결과적으로 구독 큐가 영구적이 되고 구독자 프로그램이 활성화되지 않은 경우에도 발행물이 계속 해당 큐에 저장됩니다. 큐 관리자는 애플리케이션 또는 관리자가 구독을 삭

제하기로 선택한 후에만 큐를 삭제하고 검색되지 않은 모든 발행물을 해당 큐에서 지웁니다. 구독은 관리 명령을 사용하거나 MQCO_REMOVE_SUB 옵션으로 구독을 닫는 방법으로 삭제할 수 있습니다.

- 큐에 발행물 송신을 중단하고 구독자가 남아 있는 모든 발행물을 이용한 후에 구독을 제거하고 큐 관리자를 통해 큐 및 큐에 남아 있는 모든 발행물을 삭제할 수 있도록 하려면 지속 가능 구독에 대해 SubExpiry 설정을 고려하십시오.
- 4. 유연한 토픽 문자열 배치: 관리적으로 정의된 토픽을 사용하여 구독의 루트 부분을 정의하면 구독 토픽 관리가 단순화됩니다. 이를 통해 토픽 트리의 루트 부분이 애플리케이션에서 숨겨집니다. 애플리케이션이 부주의로 다른 인스턴스나 다른 애플리케이션에 의해 작성된 다른 토픽 트리와 겹치는 토픽 트리를 작성하지 않는다면 루트 부분을 숨겨서 애플리케이션을 배치할 수 있습니다.
- 5. 관리 대상 토픽: 첫 번째 부분이 관리적으로 정의된 토픽 오브젝트와 일치하는 토픽 문자열을 사용하면 토픽 오브젝트의 속성에 따라 발행물이 관리됩니다.
 - 예를 들어, 토픽 문자열의 첫 번째 부분이 클러스터된 토픽 오브젝트와 연관된 토픽 문자열과 일치하는 경우 구독은 클러스터의 다른 멤버로부터 발행물을 수신할 수 있습니다.
 - 관리적으로 정의된 토픽 오브젝트와 프로그래밍 방식으로 정의된 구독을 선택적으로 일치시켜 양쪽의 이점을 결합할 수 있습니다. 관리자는 토픽에 대한 속성을 제공하고 프로그래머는 토픽 관리에 관여하지 않고 하위 토픽을 동적으로 정의합니다.
 - sd.Objectname에 이름 지정된 토픽 오브젝트는 아니지만 토픽과 연관된 속성을 제공하는 토픽 오브젝트와 일치시키는 데 사용되는 결과적인 토픽 문자열입니다. 하지만 두 오브젝트는 일반적으로 동일한 것으로 나타납니다. 826 페이지의 『예제 2: 가변 토픽에 대한 발행자』를 참조하십시오.

예제에서 구독을 지속 가능으로 설정하면 구독자가 MQCO_KEEP_SUB 옵션으로 구독을 닫은 후에도 계속 구독 큐로 발행물이 송신됩니다. 큐는 구독자가 활성 상태가 아닐 때에도 계속 발행물을 수신합니다. 이 작동은 MQSO_PUBLICATIONS_ON_REQUEST 옵션으로 구독을 작성하고 MQSUBRQ를 사용하여 보유된 발행물을 요청하는 방법으로 대체할 수 있습니다.

MQCO_RESUME 옵션으로 구독을 열어 나중에 구독을 재개할 수 있습니다.

다양한 방법으로 MQSUB에서 리턴하는 큐 핸들 Hobj를 사용할 수 있습니다. 큐 핸들은 예제에서 구독 큐의 이름으로 조회하는 데 사용됩니다. 관리되는 큐는 기본 모델 큐 SYSTEM.NDURABLE.MODEL.QUEUE 또는 SYSTEM.DURABLE.MODEL.QUEUE를 사용하여 열립니다. 토픽별로 사용자 자체의 지속 가능 및 지속 불가능 모델 큐를 구독과 연관되는 토픽 오브젝트의 특성으로 제공하여 기본값을 대체할 수 있습니다.

모델 큐에서 상속되는 속성에 관계없이, 관리되는 큐 핸들을 재사용하여 추가 구독을 작성할 수 없습니다. 리턴된 큐 이름을 사용하여 관리되는 큐를 다시 열어 관리되는 큐의 다른 핸들을 확보할 수도 없습니다. 큐는 독점 입력으로 열리듯이 작동합니다.

관리되지 않는 큐는 관리되는 큐보다 더 유연합니다. 예를 들면, 관리되지 않는 큐를 공유하거나 하나의 큐에 여러 구독을 정의할 수 있습니다. 다음 예에서는 구독을 관리되지 않는 구독 큐와 결합하는 방법을 설명합니다.

참고: 압축 코딩 스타일은 프로덕션용이 아닌 읽기용입니다.

결과는 835 페이지의 그림 103에 표시됩니다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault = "STOCKS";
    char      topicStringDefault[] = "IBM/PRICE";
    MQCHAR48 qmName = "";          /* Use default queue manager */
    MQCHAR48 qName = "";          /* Allocate to query queue name */
    char      publicationBuffer[101]; /* Allocate to receive messages */
    char      resTopicStrBuffer[151]; /* Allocate to resolve topic string */

    MQHCONN Hconn = MQHC_UNUSABLE_HCONN; /* connection handle */
    MQHOBJ Hobj = MQHO_NONE;             /* publication queue handle */
    MQHOBJ Hsub = MQSO_NONE;             /* subscription handle */
    MQLONG CompCode = MQCC_OK;           /* completion code */
    MQLONG Reason = MQRC_NONE;           /* reason code */
    MQLONG messlen = 0;
    MQSD sd = {MQSD_DEFAULT};           /* Subscription Descriptor */
    MQMD md = {MQMD_DEFAULT};           /* Message Descriptor */
    MQGMO gmo = {MQGMO_DEFAULT};        /* get message options */

    char * topicName = topicNameDefault;
    char * topicString = topicStringDefault;
    char * publication = publicationBuffer;
    char * resTopicStr = resTopicStrBuffer;
    memset(resTopicStr, 0, sizeof(resTopicStrBuffer));

    switch(argc){                      /* Replace defaults with args if provided */
    default:
        topicString = argv[2];
    case(2):
        if (strcmp(argv[1],"/"))        /* "/" invalid = No topic object */
            topicName = argv[1];
        else
            *topicName = '\0';
    case(1):
        printf("Optional parameters: topicName, topicString\nValues \"%s\" \"%s\"\n",
            topicName, topicString);
    }
}
```

그림 101. 관리되는 MQ 구독자 - 파트 1: 선언 및 매개변수 핸들링

이 예제에는 선언에 대한 추가 주석이 있습니다.

MQHOBJ Hobj = MQHO_NONE;

지속 불가능한 관리되는 구독 큐를 명시적으로 열어 발행물을 수신할 수 없지만, 큐가 자동으로 열릴 때 큐 관리자가 리턴하는 오브젝트 핸들에 대한 스토리지를 할당해야 합니다. 핸들을 MQHO_OBJECT로 초기화하는 것이 중요합니다. 이는 큐 핸들을 구독 큐로 리턴해야 함을 큐 관리자에게 표시합니다.

MQSD sd = {MQSD_DEFAULT};

MQSUB에서 사용되는 새 구독 디스크립터입니다.

MQCHAR48 qName;

예제에서는 구독 큐에 대한 지식이 필요하지 않지만 구독 큐의 이름을 조회합니다. MQINQ 바인딩은 C 언어로 처리하기가 조금 곤란하므로 예제의 이 부분을 검토하면 도움이 됩니다.

```

do {
    MQCONN(qmName, &Hconn, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
    sd.ObjectString.VSPtr = topicString;
    sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
    sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF QUIESCING ;
    sd.ResObjectString.VSPtr = resTopicStr;
    sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
    MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
    gmo.WaitInterval = 10000;
    inquireQname(Hconn, Hobj, qName);
    printf("Waiting %d seconds for publications matching \"%s\" from \"%-0.48s\\\"\\n",
        gmo.WaitInterval/1000, resTopicStr, qName);
    do {
        memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
        memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
        md.Encoding = MQENC_NATIVE;
        md.CodedCharSetId = MQCCSI_Q_MGR;
        memset(publicationBuffer, 0, sizeof(publicationBuffer));
        MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1,
            publication, &messlen, &CompCode, &Reason);
        if (Reason == MQRC_NONE)
            printf("Received publication \"%s\\\"\\n", publication);
    }
    while (CompCode == MQCC_OK);
    if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
    MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
printf("Completion code %d and Return code %d\\n", CompCode, Reason);
return;
}

void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors */
    MQLONG intAttrs[_intAttrs]; /* Array of integer attributes */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
        &CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\\n", CompCode, Reason);
        strcpy(qName, "unknown queue");
    }
    return;
}

```

그림 102. 관리되는 MQ 구독자 - 파트 2: 코드 본문

```

W:\Subscribe2\Debug>solution2
Optional parameters: topicName, topicString
Values "STOCKS" "IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.48A0AC7403300020 "
Received publication "150"
Completion code 0 and Return code 0

W:\Subscribe2\Debug>solution2 / NYSE/IBM/PRICE
Optional parameters: topicName, topicString
Values "" "NYSE/IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.48A0AC7403310020 "
Received publication "150"
Completion code 0 and Return code 0

```

그림 103. MQ 구독자

이 예제에는 코드에 대한 추가 주석이 있습니다.

strncpy(sd.ObjectName, topicName, MQ_Q_NAME_LENGTH);

topicName이 널이거나 비어 있는 경우(기본값), 토픽 이름이 해석된 토픽 문자열을 처리하는 데 사용되지 않습니다.

sd.ObjectString.VSPtr = topicString;

이 예제에서 프로그래머는 단독으로 사전정의된 토픽 오브젝트를 사용하지 않고 MQSUB에 의해 결합되는 토픽 오브젝트와 토픽 문자열을 제공합니다. 토픽 문자열이 MQCHARV 구조라는 점에 주의하십시오.

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;

MQCHARV 필드의 길이를 설정하는 대안입니다.

sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING;

토픽 문자열을 정의한 후, sd.Options 플래그는 가장 세심한 주의를 요합니다. 많은 옵션이 있지만, 예제에서는 가장 일반적으로 사용되는 옵션만 지정합니다. 기타 옵션은 기본값을 사용합니다.

1. 구독이 지속 불가능한 경우 즉, 애플리케이션에서 열린 구독의 활성 시간이 있는 경우, MQSO_CREATE 플래그를 설정하십시오. 또한 읽기용으로 (기본값) MQSO_NON_DURABLE 플래그를 설정할 수 있습니다.
2. MQSO_RESUME은 MQSO_CREATE를 보완한 것입니다. 두 플래그는 함께 설정할 수 있고, 큐 관리자는 새 구독을 작성하거나 기존 구독을 재개하거나 어느 쪽이든 적절한 작업을 수행합니다. 하지만 MQSO_RESUME을 지정하면 재개할 구독이 없는 경우에도 sd.SubName의 MQCHARV 구조를 초기화해야 합니다. SubName 초기화에 실패하면 MQSUB에서 2440: MQRC_SUB_NAME_ERROR의 리턴 코드가 발생합니다.

참고: MQSO_RESUME은 지속 불가능한 관리되는 구독에 대해 항상 무시됩니다. 하지만 sd.SubName의 MQCHARV 구조를 초기화하지 않고 지정하면 오류가 발생합니다.

3. 또한 구독을 여는 방법에 영향을 미치는 세 번째 플래그(MQSO_ALTER)가 있습니다. 올바른 권한이 부여 되면, 재개된 구독의 특성이 MQSUB에 지정된 다른 속성과 일치하도록 변경됩니다.

참고: MQSO_CREATE, MQSO_RESUME 및 MQSO_ALTER 플래그 중 최소 하나는 지정되어야 합니다. 옵션 (MQLONG)을 참조하십시오. 837 페이지의 『예제 3: 관리되지 않는 MQ 구독자』에는 세 가지 플래그를 모두 사용한 예제가 있습니다.

4. 큐 관리자에서 자동으로 구독을 관리하려면 MQSO_MANAGED를 설정하십시오.

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;

선택적으로 널(Null) 종료 문자열에 대해 MQCHARV의 길이 설정을 생략하고 대신 널 종료자 플래그를 사용합니다.

sd.ResObjectString.VSPtr = resTopicStr;

결과 토픽 문자열은 프로그램의 첫 번째 printf에서 반환됩니다. 해결된 문자열을 프로그램으로 IBM MQ 반환하도록 MQCHARV ResObjectString 설정합니다.

참고: resTopicStringBuffer는 memset(resTopicStr, 0, sizeof(resTopicStrBuffer))에서 널로 초기화됩니다. 리턴된 토픽 문자열은 후행 널로 끝나지 않습니다.

sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;

sd.ResObjectString의 버퍼 크기를 실제 크기보다 1만큼 작게 설정합니다. 이렇게 하면 해석된 토픽 문자열이 전체 버퍼를 채운 경우에 제공되는 널 종료자를 덮어씁니다.

참고: 토픽 문자열이 sizeof(resTopicStrBuffer)-1보다 길면 오류가 리턴되지 않습니다.

VSLength > VSBufSiz인 경우에도 sd.ResObjectString.VSLength에 리턴되는 길이는 리턴된 문자열의 길이가 아니라 완료 문자열의 길이입니다. 테스트 sd.ResObjectString.VSLength < sd.ResObjectString.VSBufSiz: 주제 문자열이 완전한지 확인합니다.

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);

MQSUB 함수는 구독을 작성합니다. 구독이 지속 불가능하면 대체로 해당 구독의 이름에 관심을 두지 않지만, IBM MQ Explorer에서 구독의 상태를 검사할 수는 있습니다. 입력으로 sd.SubName 매개변수를 제공하여 찾으려는 이름을 알 수 있지만, 분명히 다른 구독과의 이름 충돌은 피해야 합니다.

MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);

구독과 구독 큐를 모두 닫는 것은 선택사항입니다. 예제에서 구독은 처리완료되었으나 큐는 아닙니다. 게다가 이 경우에는 구독이 지속 불가능하므로 MQCLOSE MQCO_REMOVE_SUB 옵션이 기본값입니다.

MQCO_KEEP_SUB를 사용하면 오류입니다.

참고: 구독 큐는 MQSUB에서 처리완료되지 않으며, 해당 핸들 Hobj는 큐가 MQCLOSE 또는 MQDISC에서 처리완료될 때까지 유효합니다. 애플리케이션이 초기에 종료되면, 간혹 애플리케이션 종료 후에 큐와 구독이 큐 관리자에서 정리되기도 합니다.

관련 개념

830 페이지의 『예제 1: MQ 발행 이용자』

MQ 발행 이용자는 토픽 자체를 구독하지 않는 IBM MQ 메시지 이용자입니다.

837 페이지의 『예제 3: 관리되지 않는 MQ 구독자』

관리되지 않는 구독자는 구독자 애플리케이션의 중요한 클래스입니다. 이 이 구독자의 경우 발행물의 큐잉 및 이용 제어를 통해 발행/구독의 이점을 결합합니다. 관리되지 않는 구독은 애플리케이션이 책임을 지는 구독입니다. 이를 사용하여 구독이 저장되는 대기열을 지정할 수 있습니다. 예제에서는 구독과 큐를 결합하는 여러 가지 방법을 보여줍니다.

823 페이지의 『발행자 애플리케이션 작성』

두 가지 예제를 검토하여 발행자 애플리케이션을 작성하는 작업을 시작합니다. 첫 번째 예제는 큐에 메시지를 넣는 포인트 투 포인트 애플리케이션에 가능한 한 가깝게 모델화되었고, 두 번째 예제는 동적으로 토픽을 작성하는 방법(즉, 발행자 애플리케이션에 대한 보다 일반적인 패턴)에 대해 설명합니다.

예제 3: 관리되지 않는 MQ 구독자

관리되지 않는 구독자는 구독자 애플리케이션의 중요한 클래스입니다. 이 이 구독자의 경우 발행물의 큐잉 및 이용 제어를 통해 발행/구독의 이점을 결합합니다. 관리되지 않는 구독은 애플리케이션이 책임을 지는 구독입니다. 이를 사용하여 구독이 저장되는 대기열을 지정할 수 있습니다. 예제에서는 구독과 큐를 결합하는 여러 가지 방법을 보여줍니다.

관리되지 않는 패턴은 지속 불가능 구독보다 지속 가능 구독과 더 흔히 연관됩니다. 일반적으로 관리되지 않는 구독자가 작성한 구독의 라이프사이클은 구독 애플리케이션 자체의 라이프사이클과는 별개입니다. 구독을 지속 가능하게 하면 구독 애플리케이션이 활성 상태가 아닐 때도 구독이 발행물을 수신합니다.

동일한 결과를 얻기 위해 지속 가능한 관리되는 구독을 작성할 수도 있으나, 일부 애플리케이션은 관리되는 구독을 사용하여 작성하는 것보다 큐 및 메시지에 대해 더 많은 유연성 및 제어를 필요로 합니다. 지속 가능한 관리되는 구독의 경우, 큐 관리자는 구독 토픽과 일치하는 발행물에 대한 영구적 큐를 작성합니다. 큐 관리자는 구독이 삭제될 때 큐 및 연관된 발행물을 삭제합니다.

일반적으로 지속 가능한 관리되는 구독은 애플리케이션 및 구독의 라이프사이클이 본질적으로 동일하나 보장하기 어려운 경우에 사용됩니다. 구독을 지속 가능하게 하고 발행자가 지속 발행물을 작성하도록 하면 큐 관리자 또는 구독자가 너무 일찍 종료하여 복구해야 할 경우 손실되는 메시지가 없습니다.

비 JMS 애플리케이션 또는 공유 구독을 사용하지 않는 JMS 애플리케이션 또는 비 JMS 애플리케이션의 경우, 큐 관리자는 큐의 공유 처리가 불가능하도록 구독자에 대해 지속 가능한 관리되는 구독 큐를 암시적으로 엽니다. 또한 애플리케이션이 JMS 공유 구독을 사용하지 않는 한, 관리되는 각 큐에 대해 둘 이상의 구독을 작성할 수 없으며 큐의 이름에 대한 제어 권한이 부족하므로 큐를 관리하기 어려울 수도 있습니다. 이러한 이유로, 지속 가능 구독이 필요한 애플리케이션에 관리되는 MQ 구독자보다 관리되지 않는 MQ 구독자가 더 적합한지 고려해 보십시오.

842 페이지의 그림 106의 코드는 관리되지 않는 지속 가능 구독 패턴을 보여줍니다. 쉽게 설명하기 위해, 이 코드는 관리되지 않는 지속 불가능 구독도 작성합니다. 이 예제는 다음과 같은 패턴 패킷에 대해 설명합니다.

- 요청 시 구독: 구독 토픽 문자열이 동적입니다. 애플리케이션 실행 시 애플리케이션이 해당 문자열을 제공합니다.
- 단순화된 구독 토픽 관리: 관리적으로 정의된 토픽을 사용하여 구독 토픽 문자열의 루트 부분을 정의함으로써 구독 토픽 관리가 단순화됩니다. 이를 통해 애플리케이션에서 토픽 트리의 루트 부분을 숨깁니다. 루트 부분을 숨겨서 구독자를 서로 다른 토픽 트리에 배치할 수 있습니다.
- 유연한 구독 관리: 구독을 관리적으로 정의하거나 구독자 프로그램에서 요청 시 작성할 수 있습니다. 구독을 작성한 방법을 보여주는 속성을 제외하고 관리 또는 프로그래밍 방식으로 작성된 구독 사이에 차이는 없습니다. 구독의 분배를 위해 큐 관리자가 자동으로 작성하는 세 번째 구독 유형이 있습니다. 모든 구독은 IBM MQ Explorer에 표시됩니다.

- 구독과 큐의 유연한 연관: 사전정의된 로컬 큐가 MQSUB 함수에 의해 구독과 연관됩니다. MQSUB를 사용하여 구독을 큐와 연관시키는 다른 방법도 있습니다.
 - 구독을 기존 구독이 없는 큐, MQSO_CREATE + (Hobj from MQOPEN)과 연관시킵니다.
 - 새 구독을 기존 구독이 있는 큐, MQSO_CREATE + (Hobj from MQOPEN)과 연관시킵니다.
 - 기존 구독을 다른 큐, MQSO_ALTER + (Hobj from MQOPEN)으로 이동합니다.
 - 기존 큐, MQSO_RESUME + (Hobj = MQHO_NONE) 또는 MQSO_RESUME + (Hobj = from MQOPEN of queue with existing subscription)과 연관된 기존 구독을 계속합니다.
 - 여러 가지 조합으로 MQSO_CREATE | MQSO_RESUME | MQSO_ALTER를 결합하여 각기 다른 sd.Options 값을 가진 다중 버전의 MQSUB를 코딩할 필요 없이 구독 및 큐의 다양한 입력 상태 요구를 충족할 수 있습니다.
 - 또는 MQSUB에 입력으로 제공된 구독과 큐의 상태가 sd.Options의 값과 일치하지 않는 경우 큐 관리자는 MQSO_CREATE | MQSO_RESUME | MQSO_ALTER의 특정 선택을 코딩하여 오류(839 페이지의 표 138)를 리턴합니다. 845 페이지의 그림 112에서는 sd.Options 플래그의 다른 개별 설정을 사용하여 구독 X에 대해 MQSUB를 발행하고 이를 세 가지의 다른 오브젝트 핸들에 전달한 결과를 보여줍니다.

841 페이지의 그림 105의 예제 프로그램에 다른 값을 입력해 보면서 이런 여러 가지 종류의 오류를 익히십시오. 표에 나열된 사례에 포함되지 않은 일반적인 오류인 RC = 2440은 구독 이름 오류입니다. 이 오류는 MQSO_RESUME 또는 MQSO_ALTER를 포함한 널 또는 올바르지 않은 구독 이름을 전달하여 발생하는 경우가 많습니다.

- 멀티프로세싱: 발행물을 읽는 작업을 많은 이용자 사이에서 공유할 수 있습니다. 발행물은 모두 구독 토픽과 연관된 단일 큐로 이동합니다. 이용자는 MQOPEN을 사용하여 큐를 직접 열거나 MQSUB를 사용하여 구독을 계속할 수 있습니다.
- 구독 집중: 동일한 큐에서 다중 구독을 작성할 수 있습니다. 이 기능을 사용할 때는 구독이 겹쳐서 동일한 발행물이 여러 번 수신될 수 있으므로 주의하십시오. MQSO_GROUP_SUB 옵션은 겹치는 구독으로 인한 중복 발행물을 제거합니다.
- 구독자와 이용자 분리: 예제에서 설명하는 세 가지 이용자 모델 외에 또 하나의 모델은 이용자를 구독자로부터 분리하는 것입니다. 이 모델은 관리되지 않는 MQ 구독자의 변형이지만, 같은 프로그램에서 MQOPEN과 MQSUB를 발행하는 것이 아니라 한 프로그램이 발행물을 구독하고 다른 한 프로그램이 해당 발행물을 이용합니다. 예를 들어, 구독자는 발행/구독 클러스터에 속하고 이용자는 큐 관리자 클러스터 외부의 큐 관리자에 연결될 수 있습니다. 이용자는 구독 큐를 리모트 큐 정의로 정의하여 표준 분산 큐잉을 통해 발행물을 수신합니다.

특히 이러한 옵션 조합을 사용하여 코드를 단순화할 계획인 경우 MQSO_CREATE | MQSO_RESUME | MQSO_ALTER의 작동을 이해하는 것이 중요합니다. 각기 다른 큐 핸들을 MQSUB에 전달한 결과 및 843 페이지의 그림 107에서 845 페이지의 그림 112까지 표시된 예제 프로그램을 실행한 결과를 보여주는 839 페이지의 표 138 표를 검토하십시오.

테이블을 생성하는 데 사용되는 시나리오에는 하나의 구독 X 및 두 개의 큐(A 및 B)가 있습니다. 구독 이름 매개 변수 'sd.SubName'은 대기열 'A'에 연결된 구독의 이름인 'X'로 설정됩니다. B 큐에는 구독이 첨부되어 있지 않습니다.

839 페이지의 표 138에서, MQSUB에 구독 X 및 큐 A에 대한 큐 핸들이 전달됩니다. 구독 옵션 결과는 다음과 같습니다.

- MQSO_CREATE는 큐 핸들이 이미 X에 대한 구독이 있는 큐 A에 대응하기 때문에 실패합니다. 이 동작을 성공적인 호출과 비교하십시오. 큐 B에는 X에 대한 구독이 연결되어 있지 않기 때문에 해당 호출은 성공합니다.
- MQSO_RESUME '은 큐 핸들이 이미 'X'에 대한 구독이 있는 큐 'A'에 해당하기 때문에 성공합니다. 이와 반대로, 구독 X가 큐 A에 존재하지 않는 호출은 실패합니다.
- MQSO_ALTER는 구독 및 큐 열기에서 MQSO_RESUME과 유사한 방법으로 작동합니다. 하지만 MQSUB에 전달되는 구독 디스크립터에 포함된 속성이 구독의 속성과 다른 경우, MQSO_RESUME은 실패하는 반면 프로그램 인스턴스에 속성을 대체할 권한이 있는 한 MQSO_ALTER는 성공합니다. 구독의 토픽 문자열은 절대 변경할 수 없지만, MQSUB는 오류를 리턴하기 보다는 구독 디스크립터의 토픽 이름 및 토픽 문자열 값을 무시하고 기존 구독의 값을 사용합니다.

다음으로, MQSUB에 구독 X 및 큐 B에 대한 큐 핸들이 전달된 [839 페이지의 표 138](#)을(를) 확인하십시오. 구독 옵션 결과는 다음과 같습니다.

- MQSO_CREATE가 성공하고 큐 B의 새 구독이기 때문에 큐 B에 구독 X를 작성합니다.
- MQSO_RESUME이 실패합니다. MQSUB는 큐 B에서 구독 X를 검토만 하고 찾지는 않지만 RC = 2428 - 구독 X가 없음을 리턴하지 않고 RC = 2019 - 구독 큐가 큐 오브젝트 핸들과 일치하지 않음을 리턴합니다. 세 번째 옵션 MQSO_ALTER의 작동은 이 예상치 못한 오류의 이유를 제안합니다. MQSUB는 큐 핸들이 구독이 있는 큐를 가리킬 것으로 예상합니다. 그리고 이를 먼저 확인한 후 sd.SubName에서 이름 지정된 구독이 존재하는지 확인합니다.
- MQSO_ALTER가 성공하고 구독을 큐 A에서 큐 B로 이동합니다.

표에 표시되지 않은 사례는 큐 A에 있는 구독의 구독 이름이 sd.SubName의 구독 이름과 일치하지 않는 경우입니다. 해당 호출은 실패하고 RC = 2428 - 구독 X가 큐 A에 없음이 표시됩니다.

표 138. 다양한 큐 핸들 및 구독 조합에서 MQSUB의 오류		
큐 핸들	큐 A 구독 X 대기열 B 구독 없음	큐 A 구독 없음 대기열 B 구독 없음
큐 A에 대한 Hobj가 MQSUB에 전달됨	MQSO_CREATE RC = 2432 - 구독 X가 큐 A에 이미 있음 MQSO_RESUME 큐 A에서 구독 X를 재개함 MQSO_ALTER 큐 A에서 구독 X를 재개하고 허용되는 대체를 수행함	MQSO_CREATE 큐 A에서 구독 X를 작성함 MQSO_RESUME RC = 2428 - 구독 X가 큐 A에 없음 MQSO_ALTER RC = 2428 - 구독 X가 큐 A에 없음
큐 B에 대한 Hobj가 MQSUB에 전달됨	MQSO_CREATE 큐 B에서 구독 X를 새로 작성함 MQSO_RESUME RC = 2019 - 구독 큐가 큐 오브젝트 핸들과 일치하지 않음 MQSO_ALTER 큐 A에서 큐 B로 구독 X를 이동함	MQSO_CREATE 큐 B에서 구독 X를 새로 작성함 MQSO_RESUME RC = 2428 - 구독 X가 큐 B에 없음 MQSO_ALTER RC = 2428 - 구독 X가 큐 B에 없음
MQHO_NONE이 MQSUB에 전달됨	MQSO_CREATE RC = 2019 - 잘못된 오브젝트 핸들임. 관리되는 구독 및 관리되는 큐를 작성하도록 MQSO_MANAGED 플래그를 설정함 MQSO_RESUME 큐 A에서 구독 X를 재개하고 Hobj를 큐 A로 리턴함 MQSO_ALTER 큐 A에서 구독 X를 재개하고 Hobj를 큐 A로 리턴하고 허용되는 대체를 수행함	MQSO_CREATE RC = 2019 - 잘못된 오브젝트 핸들임. 관리되는 구독 및 관리되는 큐를 작성하도록 MQSO_MANAGED 플래그를 설정함 MQSO_RESUME RC = 2428 - 구독 X 없음 MQSO_ALTER RC = 2019 - 잘못된 오브젝트 핸들임. 큐 A 또는 B 없음

참고: 압축 코딩 스타일은 프로덕션용이 아닌 읽기용입니다.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault      = "STOCKS";
    char      topicStringDefault[] = "IBM/PRICE";
    char      subscriptionNameDefault[] = "IBMSTOCKPRICESUB";
    char      subscriptionQueueDefault[] = "STOCKTICKER";
    char      publicationBuffer[101];    /* Allocate to receive messages */
    char      resTopicStrBuffer[151];    /* Allocate to resolve topic string */
    MQCHAR48 qmName = "";                /* Default queue manager */
    MQCHAR48 qName = "";                 /* Allocate storage for MQINQ */

    MQHCONN Hconn = MQHC_UNUSABLE_HCONN; /* connection handle */
    MQHOBJ Hobj = MQHO_NONE;              /* subscription queue handle */
    MQHOBJ Hsub = MQSO_NONE;              /* subscription handle */
    MQLONG CompCode = MQCC_OK;            /* completion code */
    MQLONG Reason = MQRC_NONE;            /* reason code */
    MQLONG messlen = 0;
    MQOD od = {MQOD_DEFAULT};             /* Unmanaged subscription queue */
    MQSD sd = {MQSD_DEFAULT};             /* Subscription Descriptor */
    MQMD md = {MQMD_DEFAULT};             /* Message Descriptor */
    MQGMO gmo = {MQGMO_DEFAULT};          /* get message options */
    MQLONG sdOptions = MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE |
MQSO_FAIL_IF QUIESCING;

    char * topicName = topicNameDefault;
    char * topicString = topicStringDefault;
    char * subscriptionName = subscriptionNameDefault;
    char * subscriptionQueue = subscriptionQueueDefault;
    char * publication = publicationBuffer;
    char * resTopicStr = resTopicStrBuffer;
    memset(resTopicStrBuffer, 0, sizeof(resTopicStrBuffer));

```

그림 104. 관리되지 않는 MQ 구독자 - 파트 1: 선언

```

        switch(argc){
        default:
            switch((argv[5][0])) {
            case('A'): sdOptions = MQSO_ALTER | MQSO_DURABLE | MQSO_FAIL_IF QUIESCING;
                        break;
            case('C'): sdOptions = MQSO_CREATE | MQSO_DURABLE | MQSO_FAIL_IF QUIESCING;
                        break;
            case('R'): sdOptions = MQSO_RESUME | MQSO_DURABLE | MQSO_FAIL_IF QUIESCING;
                        break;
            default:
                ;
            }
        case(5):
            if (strcmp(argv[4],"/")) /* "/" invalid = No subscription */
                subscriptionQueue = argv[4];
            else {
                *subscriptionQueue = '\0';
                if (argc > 5) {
                    if (argv[5][0] == 'C') {
                        sdOptions = sdOptions + MQSO_MANAGED;
                    }
                }
                else
                    sdOptions = sdOptions + MQSO_MANAGED;
            }
        case(4):
            if (strcmp(argv[3],"/")) /* "/" invalid = No subscription */
                subscriptionName = argv[3];
            else {
                *subscriptionName = '\0';
                sdOptions = sdOptions - MQSO_DURABLE;
            }
        case(3):
            if (strcmp(argv[2],"/")) /* "/" invalid = No topic string */
                topicString = argv[2];
            else
                *topicString = '\0';
        case(2):
            if (strcmp(argv[1],"/")) /* "/" invalid = No topic object */
                topicName = argv[1];
            else
                *topicName = '\0';
        case(1):
            sd.Options = sdOptions;
            printf("Optional parameters: "
                printf("topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|
                R(esume)\n");
            printf("Values \"%-48s\" \"%s\" \"%s\" \"%-48s\" sd.Options=%d\n",
                topicName, topicString, subscriptionName, subscriptionQueue, sd.Options);
        }

```

그림 105. 관리되지 않는 MQ 구독자 - 파트 2: 매개변수 핸들링

이 예제에서 매개변수 핸들링에 대한 추가 주석은 다음과 같습니다.

switch((argv[5][0]))

매개변수 5에 **A lter** | **C reate** | **R esume**을 입력하여 예제에서 기본적으로 사용된 MQSUB 옵션 설정의 일부를 대체하는 효과를 테스트하는 옵션을 선택할 수 있습니다. 예제에서 사용된 기본 설정은 MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE입니다.

참고: MQSO_DURABLE을 설정하지 않고 MQSO_ALTER 또는 MQSO_RESUME를 설정하는 것은 오류이며, sd.SubName을 설정해야 하고 재개되거나 대체될 수 있는 구독을 참조해야 합니다.

***subscriptionQueue = '\0';**

sdOptions = sdOptions + MQSO_MANAGED;

기본 구독 큐, STOCKTICKER가 널 문자열로 바뀌는 경우, MQSO_CREATE가 설정되어 있는 동안은 예제에서 MQSO_MANAGED 플래그를 설정하고 동적 구독 큐를 작성합니다. 5번째 매개변수에 **Alter or Resume**이 설정되어 있는 경우 예제의 작동은 subscriptionName의 값에 따라 결정됩니다.

```
*subscriptionName = '\0';
```

```
sdOptions = sdOptions - MQSO_DURABLE;
```

기본 구독, IBMSTOCKPRICESUB가 널 문자열로 바뀌는 경우에는 예제에서 MQSO_DURABLE 플래그를 제거합니다. 기타 매개변수에 대해 기본값을 제공하는 예제를 실행하는 경우 STOCKTICKER로 목적지가 지정된 추가 임시 구독이 작성되고 중복 발행물을 수신합니다. 다음에 매개변수 없이 예제를 실행하면 하나의 발행물만 다시 수신합니다.

```
do {
    MQCONN(qmName, &Hconn, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    if (strlen(subscriptionQueue)) {
        strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
        MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING | MQOO_INQUIRE,
            &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
    }
    strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
    sd.ObjectString.VSPtr = topicString;
    sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
    sd.SubName.VSPtr = subscriptionName;
    sd.SubName.VSLength = MQVS_NULL_TERMINATED;
    sd.ResObjectString.VSPtr = resTopicStr;
    sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
    MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
    gmo.WaitInterval = 10000;
    gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
    memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
    inquireQname(Hconn, Hobj, qName);
    printf("Waiting %d seconds for publications matching \"%s\" from %-0.48s\n",
        gmo.WaitInterval/1000, resTopicStr, qName);
    do {
        memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
        memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
        md.Encoding = MQENC_NATIVE;
        md.CodedCharSetId = MQCCSI_Q_MGR;
        MQGET(Hconn, Hobj, &md, &gmo, sizeof(publication), publication, &messlen,
            &CompCode, &Reason);
        if (Reason == MQRC_NONE)
            printf("Received publication \"%s\"\n", publication);
    }
    while (CompCode == MQCC_OK);
    if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
    MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors */
    MQLONG intAttrs[_intAttrs]; /* Array of integer attributes */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
        &CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strncpy(qName, "unknown queue", MQ_Q_NAME_LENGTH);
    }
    return;
}
```

그림 106. 관리되지 않는 MQ 구독자 - 파트 3: 코드 본문

이 예제에서 코드에 대한 추가 주석은 다음과 같습니다.

if (strlen(subscriptionQueue))

구독 큐 이름이 없는 경우에는 예제에서 MQHO_NONE을 Hobj의 값으로 사용합니다.

MQOPEN(...);

구독 큐가 열리고 큐 핸들이 Hobj에 저장됩니다.

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);

MQOPEN(구독 큐 이름이 없는 경우에는 MQHO_NONE)에서 전달된 Hobj를 사용하여 구독이 열립니다. 관리되지 않는 큐는 MQOPEN으로 명시적으로 열지 않고도 재개할 수 있습니다.

MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);

구독 핸들을 사용하여 구독이 닫힙니다. 구독이 지속 가능한지 여부에 따라서 암시적 MQCO_KEEP_SUB 또는 MQCO_REMOVE_SUB를 사용하여 구독이 닫힙니다. MQCO_REMOVE_SUB로 지속 가능 구독을 닫을 수 있지만, MQCO_KEEP_SUB로 지속 불가능 구독을 닫을 수는 없습니다. MQCO_REMOVE_SUB의 조치는 구독 큐로 송신되는 추가적인 발행물을 중지하는 구독을 제거하는 것입니다.

MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);

구독이 관리되지 않는 경우에는 특별한 조치가 수행되지 않습니다. 큐가 관리되고 구독이 명시적 또는 암시적 MQCO_REMOVE_SUB를 사용하여 닫힌 경우 모든 발행물이 큐에서 제거되고 큐가 이 지점에서 제거됩니다.

gmo.MatchOptions = MQMO_MATCH_CORREL_ID;

memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);

수신된 메시지가 구독에 대한 것인지 확인하십시오.

예제의 결과는 발행/구독의 측면을 설명합니다.

843 페이지의 그림 107에서는 예제가 NYSE/IBM/PRICE 토픽에 대해 130을 발행하는 것으로 시작합니다.

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

그림 107. NYSE/IBM/PRICE에 130 발행

843 페이지의 그림 108에서 기본 매개변수를 사용하여 예제를 실행하면 보유한 발행물 130이 수신됩니다.

845 페이지의 그림 112에 표시된 것처럼, 제공되는 토픽 오브젝트와 토픽 문자열은 무시됩니다. 토픽 오브젝트 및 토픽 문자열은 하나가 제공되면 항상 구독 오브젝트에서 가져오고 토픽 문자열은 불변입니다. 예제의 실제 작동은 MQSO_CREATE, MQSO_RESUME 및 MQSO_ALTER의 선택 또는 조합에 따라 결정됩니다. 이 예제에서는 MQSO_RESUME이 선택된 옵션입니다.

```
W:\Subscribe3\Debug>solution3
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8206
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

그림 108. 보유한 발행물 수신

844 페이지의 그림 109에서는 지속 가능 구독이 이미 보유한 발행물을 수신했기 때문에 발행물이 수신되지 않습니다. 이 예제에서는 큐 이름 없이 구독 이름만을 제공하여 구독을 재개합니다. 큐 이름이 제공된 경우에는 큐가 먼저 열리고 핸들이 MQSUB에 전달됩니다.

참고: MQINQ에서 발생하는 2038 오류는 MQOO_INQUIRE 옵션을 포함하지 않는 MQSUB에 의한 STOCKTICKER의 암시적 MQOPEN 때문입니다. 큐를 명시적으로 열어 MQINQ에서 2038 리턴 코드를 배제하십시오.

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE IBMSTOCKPRICESUB / Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(1ter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "" sd.Options=8204
MQINQ failed with Condition code 2 and Reason 2038
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from unknown queue
Completion code 0 and Return code 0
```

그림 109. 구독 계속

844 페이지의 그림 110의 예제에서는 STOCKTICKER를 목적지로 사용하여 지속 불가능한 관리되지 않는 구독을 작성합니다. 이는 새 구독이므로 보유된 발행물을 수신합니다.

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(1ter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

그림 110. 관리되지 않는 지속 불가능한 새 구독을 사용하여 보유된 발행물 수신

844 페이지의 그림 111에서는 겹치는 구독을 보여주기 위해 보유된 발행물을 변경하여 또 하나의 발행물을 송신합니다. 그런 다음, 구독 이름을 제공하지 않고 지속 불가능한 관리되지 않는 구독을 새로 작성합니다. 보유된 발행물은 두 번 수신됩니다. 한 번은 새 구독용이고 다른 한 번은 STOCKTICKER 큐에서 여전히 사용 중인 지속 가능한 IBMSTOCKPRICESUB 구독용입니다. 예제는 애플리케이션이 아니라 구독이 있는 큐에 대한 설명입니다. 애플리케이션의 이런 호출에서 IBMSTOCKPRICESUB 구독을 참조하지는 않지만, 애플리케이션은 발행물을 두 번 수신합니다. 한 번은 관리상으로 작성된 지속 가능 구독에서, 한 번은 애플리케이션 자체가 작성한 지속 불가능 구독에서 수신합니다.

```
W:\Subscribe3\Debug>...\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(1ter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Received publication "130"
Completion code 0 and Return code 0
```

그림 111. 구독 겹침

845 페이지의 그림 112의 예제에서는 새 토픽 문자열과 기존 구독을 제공해도 구독이 변경되지 않음을 보여줍니다.

1. 첫 번째 사례에서는 예상대로 Resume이 기존 구독을 재개하고 변경된 토픽 문자열을 무시합니다.
2. 두 번째 사례에서는 Alter로 인해 RC = 2510, Topic not alterable 오류가 발생합니다.
3. 세 번째 예제에서 'Create' RC = 2432, Sub already exists' 오류를 발생시킵니다.


```

W:\Subscribe3\Debug>solution3 "" NASDAQ/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAQ/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8204
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 "" NASDAQ/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Alter
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAQ/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8201
Completion code 2 and Return code 2510

W:\Subscribe3\Debug>solution3 "" NASDAQ/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAQ/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8202
Completion code 2 and Return code 2432

```

그림 112. 구독 토픽을 변경할 수 없음

관련 개념

830 페이지의 『예제 1: MQ 발행 사용자』

MQ 발행 사용자는 토픽 자체를 구독하지 않는 IBM MQ 메시지 이용자입니다.

832 페이지의 『예제 2: 관리되는 MQ 구독자』

관리되는 MQ 구독자는 대부분의 구독자 애플리케이션에서 자주 사용하는 패턴입니다. IBM MQ은(는) 관리되는 구독을 통해 구독을 처리하고 등록 및 등록 취소를 수행합니다. 예제에서는 큐, 토픽 또는 구독의 관리 정의가 필요하지 않습니다.

823 페이지의 『발행자 애플리케이션 작성』

두 가지 예제를 검토하여 발행자 애플리케이션을 작성하는 작업을 시작합니다. 첫 번째 예제는 큐에 메시지를 넣는 포인트 투 포인트 애플리케이션에 가능한 한 가깝게 모델화되었고, 두 번째 예제는 동적으로 토픽을 작성하는 방법(즉, 발행자 애플리케이션에 대한 보다 일반적인 패턴)에 대해 설명합니다.

발행/구독 라이프사이클

발행/구독 애플리케이션을 설계할 때에는 토픽, 구독, 구독자, 발행, 발행자 및 큐의 라이프사이클을 숙고하십시오.

구독과 같은 오브젝트의 라이프사이클은 오브젝트 작성으로 시작해서 오브젝트 삭제로 끝납니다. 또한 오브젝트는 다른 상태 및 변경(예: 임시 중단)을 거치므로 상위와 하위 토픽에 만기 및 삭제가 있을 수도 있습니다.

일반적으로 IBM MQ 오브젝트(예: 큐)는 관리상으로 작성되거나 PCF(Programmable Command Format)를 사용하여 관리 프로그램에 의해 작성됩니다. 발행/구독은 구독을 작성하고 삭제하기 위해 MQSUB 및 MQCLOSE API verb를 제공하고, 큐를 작성하고 삭제할 뿐만 아니라 이용되지 않는 메시지를 정리하는 관리되는 구독의 개념을 가지며, 관리상으로 작성된 토픽 오브젝트와 관리 또는 프로그래밍 방식으로 작성된 토픽 문자열 사이에 연관이 있다는 점이 다릅니다.

이러한 기능 보강으로 광범위한 발행/구독 요구사항을 만족시키고 발행/구독 애플리케이션의 공용 패턴 설계를 단순화합니다. 예를 들어, 관리되는 구독은 프로그램이 구독을 작성하는 동안만 지속되도록 구독의 관리 및 프로그래밍을 모두 단순화합니다. 관리되지 않는 구독은 구독 중인 발행물과 이용 중인 발행물 사이의 연결이 느슨해지도록 프로그래밍을 단순화합니다. 패턴이 중앙 집중식 제어 모델(예: 자동화된 게이트에 항공편 정보 보내기)을 기반으로 이용자에게 발행 트래픽을 라우팅하는 패턴인 경우에는 중앙 집중식으로 작성된 구독이 유용하지만, 게이트 직원이 게이트에서 항공편 번호를 입력하여 해당 항공편에 대한 승객 레코드를 구독해야 할 경우에는 프로그래밍 방식으로 작성된 구독이 사용됩니다.

이 마지막 예에서는 관리되는 지속 가능한 구독이 적합할 수 있습니다. 구독이 매우 자주 작성되고 있으며 게이트가 닫힐 때 명확한 엔드포인트가 있고 구독을 프로그래밍 방식으로 제거할 수 있기 때문에 관리되는 방식이 적합하고, 어떠한 이유로 인해 작동이 중지되는 게이트 구독자 프로그램으로 인해 승객 레코드가 손실되지 않도록 하기 때문에 지속 가능 방식이 적합합니다.⁸게이트에 승객 레코드의 발행을 시작하기 위한 가능한 디자인은 게이트 애플리케이션이 게이트 번호를 사용하여 승객 레코드를 구독하고 게이트 번호를 사용하여 게이트 열기 이벤트를 발행하는 것입니다. 발행자는 승객 레코드를 발행하여 게이트 열기 이벤트에 응답합니다. 이 경우 해당 레코드는 항공편 수속을 기록하도록 다른 이해 관계자(예: 대금 청구)에게 제공되고 승객의 휴대전화에 게이트 번호 알림을 문자로 보내기 위해 고객 서비스에도 제공됩니다.

중앙 집중식으로 관리되는 구독은 지속 가능한 관리되지 않는 모델을 사용하여 각 게이트에 대해 사전정의된 큐를 통해 승객 목록을 게이트로 라우팅합니다.

⁸ 발행자는 다른 가능한 실패를 방지하기 위해 승객 레코드를 지속 메시지로 전송해야 합니다.

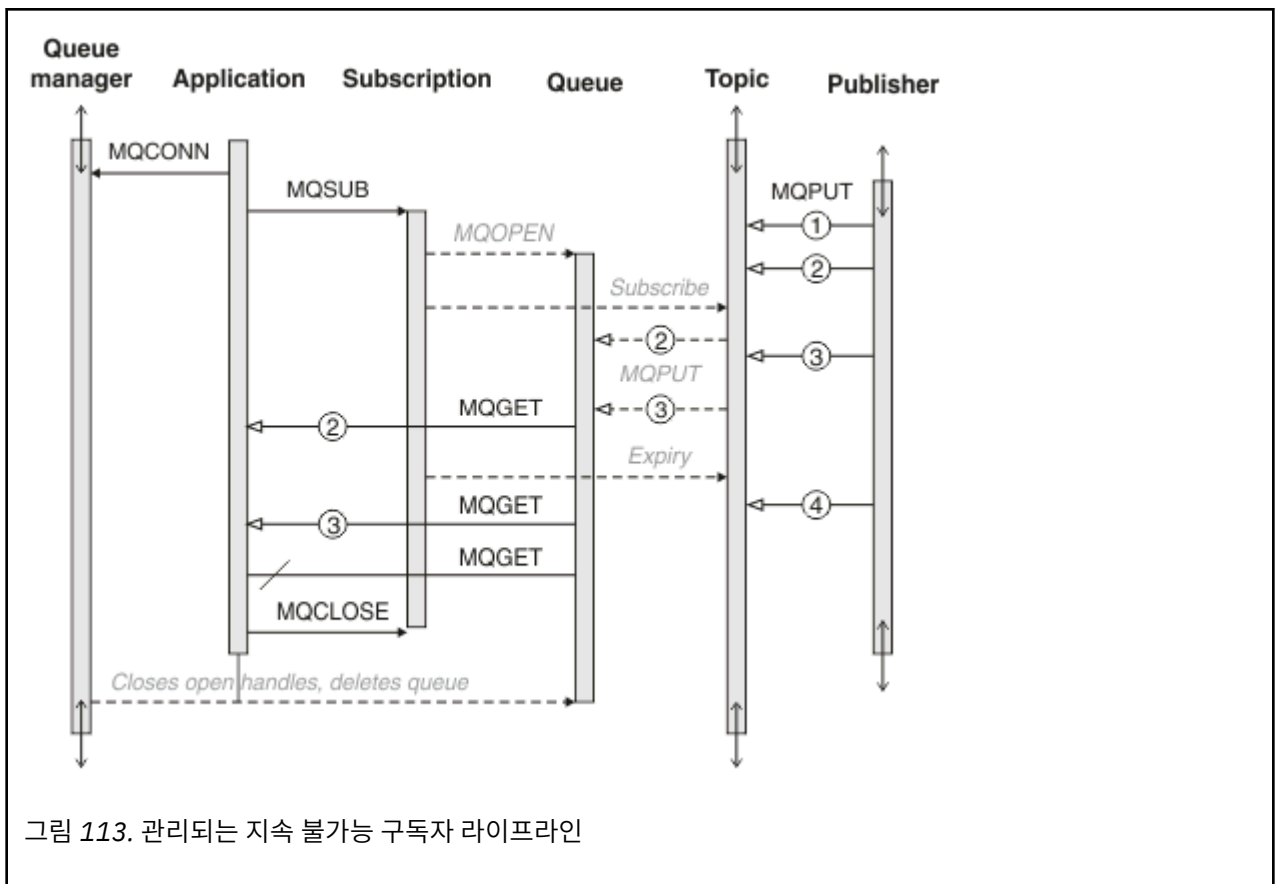
다음 세 가지의 발행/구독 라이프사이클 예제에서는 관리되는 지속 불가능, 관리되는 지속 가능 및 관리되지 않는 지속 가능 구독자가 구독, 토픽, 큐, 발행자 및 큐 관리자와 상호작용하는 방법과 관리와 구독자 프로그램 간에 책임을 분배하는 방법에 대해 설명합니다.

관리되는 지속 불가능 구독자

846 페이지의 그림 113에서는 관리되는 지속 불가능 구독을 작성하고 구독에서 식별된 토픽으로 발행되는 두 가지 메시지를 가져오고 종료하는 애플리케이션을 보여줍니다. 점선 화살표가 있는 이탤릭체 회색 글꼴로 레이블 지정된 상호작용은 암시적입니다.

몇 가지 참고사항이 있습니다.

1. 애플리케이션은 이미 발행된 토픽에 대해 구독을 두 번 작성합니다. 구독자는 첫 번째 발행물을 수신할 때, 현재 보유된 발행물인 두 번째 발행물을 수신합니다.
2. 큐 관리자는 토픽에 대한 구독을 작성하고 임시 구독 큐도 작성합니다.
3. 구독은 만기가 있습니다. 구독이 만료되면 토픽에 대한 발행물이 더 이상 이 구독으로 송신되지 않지만, 구독자는 구독이 만료되기 전에 발행된 메시지를 계속해서 가져옵니다. 발행 만기는 구독 만기에 영향을 받지 않습니다.
4. 네 번째 발행물은 구독 큐에 배치되지 않으며, 따라서 마지막 MQGET이 발행물을 리턴하지 않습니다.
5. 구독자는 구독을 닫아도 큐 또는 큐 관리자에 대한 연결은 닫지 않습니다.
6. 큐 관리자는 애플리케이션이 종료된 후 바로 정리합니다. 구독이 관리되고 지속 불가능하기 때문에 구독 큐는 삭제됩니다.



관리되는 지속 가능 구독자

관리되는 지속 가능 구독자는 이전 예제보다 한 단계 더 진행하고, 구독 애플리케이션을 종료 후 재시작했을 때 관리되는 구독이 존속하는 것을 보여줍니다.

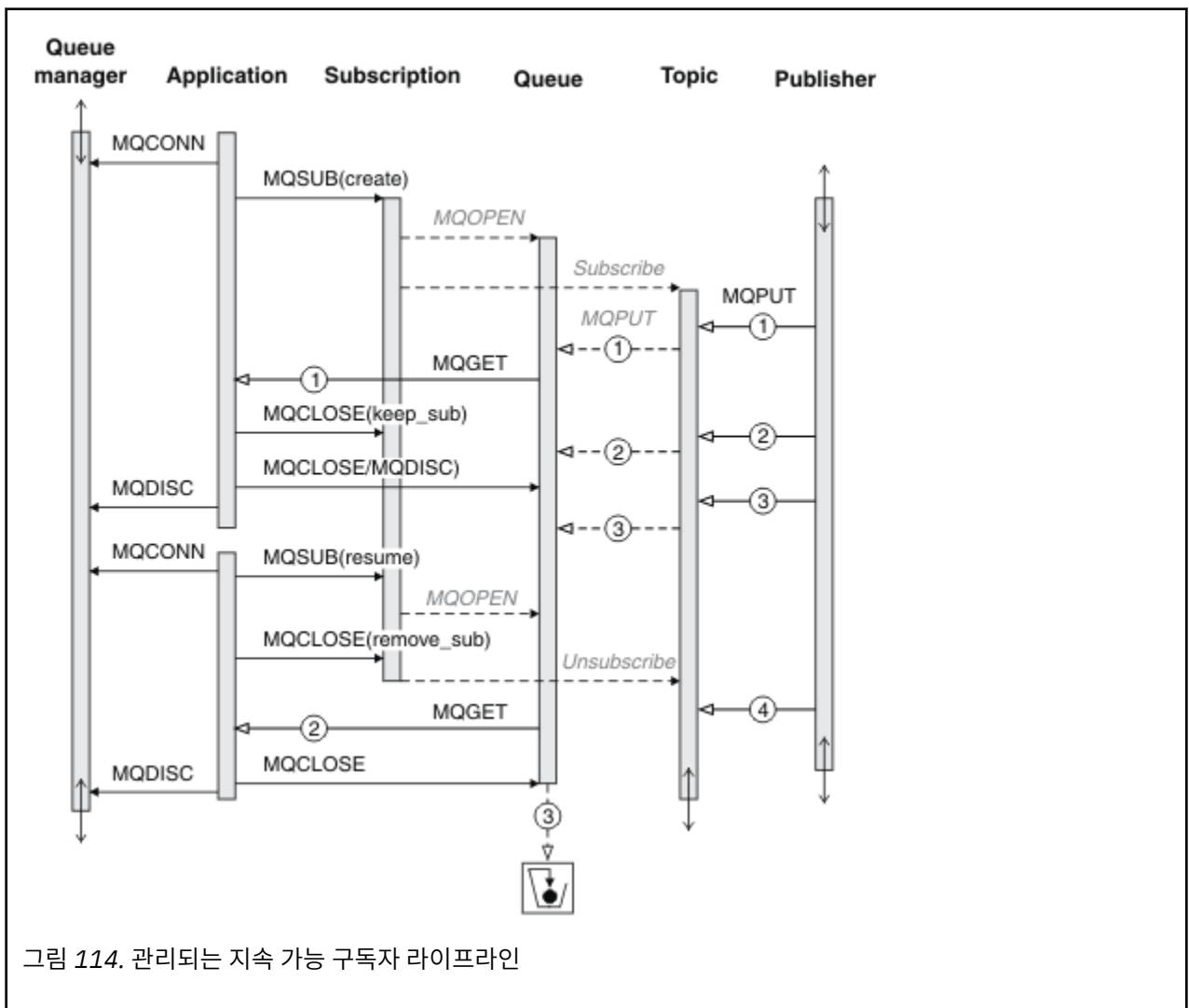
몇 가지 새로운 참고사항이 있습니다.

1. 이 예제에서는 지난 예제와 달리 구독에 정의되기 전에 발행 토픽이 없었습니다.
2. 구독자는 처음 종료할 때, MQCO_KEEP_SUB 옵션으로 구독을 닫습니다. 이는 관리되는 지속 가능 구독을 암시적으로 닫기 위한 기본 작동입니다.
3. 구독자가 구독을 재개할 때 구독 큐가 다시 열립니다.
4. 새 발행물 2는 다시 열리기 전에 큐에 배치되므로 구독이 제거된 후에도 MQGET에서 사용 가능합니다.

구독이 지속 가능하더라도 구독자는 구독이 지속 가능하고 메시지가 지속적이라는 조건이 모두 충족된 경우에만 발행자가 보낸 모든 메시지를 확실히 수신합니다. 메시지 지속성은 발행자가 보낸 메시지의 MQMD에 있는 Persistent 필드의 설정에 따라 다릅니다. 구독자는 이를 제어할 수 없습니다.

5. MQCO_REMOVE_SUB 플래그로 구독을 닫으면 구독이 제거되어 추가 발행물이 구독 큐에 배치되지 않습니다. 구독 큐가 처리완료되면, 큐 관리자는 읽지 않은 발행물 3을 제거한 후 큐를 삭제합니다. 이 조치는 구독을 관리상으로 삭제하는 것과 같습니다.

참고: 수동으로 큐를 삭제하거나 MQCO_DELETE 또는 MQCO_PURGE_DELETE 옵션으로 MQCLOSE를 실행하지 마십시오. 관리되는 구독에 대한 가시적 구현 세부사항은 지원되는 IBM MQ 인터페이스의 일부가 아닙니다. 큐 관리자는 완전히 제어하지 않는 한 구독을 확실하게 관리할 수 없습니다.

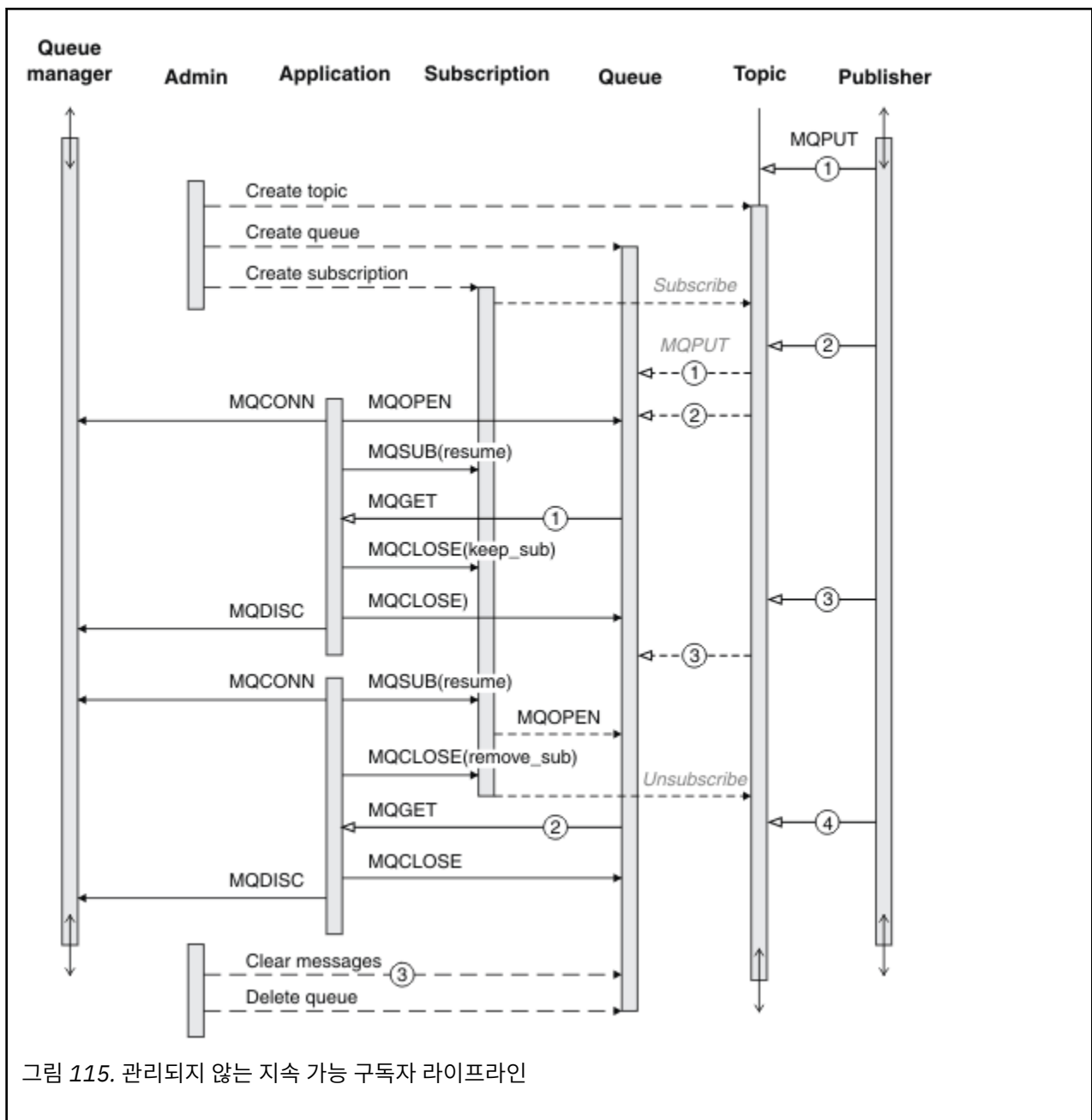


관리되지 않는 지속 가능 구독자

관리자는 세 번째 예제(관리되지 않는 지속 가능 구독자)에서 추가됩니다. 관리자가 발행/구독 애플리케이션과 상호작용하는 방법을 보여주는 좋은 예입니다.

참고사항이 나열되어 있습니다.

1. 발행자는 구독에 사용된 토픽 오브젝트와 나중에 연관이 될 토픽에 메시지(1)를 넣습니다. 토픽 오브젝트는 와일드카드를 사용하여 발행된 토픽과 일치하는 토픽 문자열을 정의합니다.
 2. 토픽에는 보유된 발행물이 있습니다.
 3. 관리자는 토픽 오브젝트, 큐 및 구독을 작성합니다. 구독보다 먼저 토픽 오브젝트와 큐를 정의해야 합니다.
 4. 애플리케이션은 구독과 연관된 큐를 열고 MQSUB에 큐의 핸들을 전달합니다. 또는 단순히 구독을 열어 큐 핸들 MQHO_NONE을 전달할 수도 있습니다. 그 반대는 해당되지 않습니다. 큐에 여러 구독이 있을 수 있으므로 구독 이름 없이 큐 핸들만 전달하면 구독을 재개할 수 없습니다.
 5. 애플리케이션은 구독을 처음 여는 경우에도 MQSO_RESUME 옵션을 사용하여 구독을 엽니다. 관리상으로 작성된 구독을 재개합니다.
 6. 구독자는 보유된 발행물(1)을 수신합니다. 발행물 2는 구독자가 발행물을 수신하기 전에 발행되었지만 구독을 시작한 후에 발행된 것으로 구독 큐에서 두 번째 발행물입니다.
- 참고:** 보유된 발행물이 지속 메시지로 발행되지 않은 경우에는 큐 관리자가 재시작된 후 손실됩니다.
7. 이 예제에서 구독은 지속 가능합니다. 프로그램이 관리되지 않는 지속 불가능 구독을 작성할 수 있고, 분명히 관리자가 할 수 있는 작업이 아니어야 합니다.
 8. 구독을 닫을 때 MQCO_REMOVE_SUB 옵션의 영향으로 관리자가 삭제한 것처럼 구독이 제거됩니다. 이 결과 추가 애플리케이션이 큐로 송신되지 않지만, 관리되는 지속 가능 구독과는 달리 큐가 처리완료된 경우에도 이미 큐에 있는 발행물에는 영향을 주지 않습니다.
 9. 관리자는 나중에 남아 있는 메시지(3)를 삭제하고 큐를 삭제합니다.



발행/구독 메시지 특성

PubAccountingToken

PubApplIdentityData

이는 이 subscription과 일치하는 모든 발행 메시지에 대한 메시지 디스크립터(MQMD)의 ApplIdentityData 필드에 있는 값입니다. ApplIdentityData는 메시지의 ID 컨텍스트에 포함됩니다. 메시지 컨텍스트에 대한 자세한 정보는 51 페이지의 『메시지 컨텍스트』의 내용을 참조하십시오. MQMD의 ApplIdentityData 필드에 대한 자세한 정보는 ApplIdentityData를 참조하십시오.

MQSO_SET_IDENTITY_CONTEXT 옵션을 지정하지 않으면 기본 컨텍스트 정보로 이 구독에 대해 발행된 각 메시지에 설정될 ApplIdentityData가 비어 있습니다.

MQSO_SET_IDENTITY_CONTEXT 옵션을 지정하면 사용자가 PubApplIdentityData를 생성하며, 이 필드는 각 발행에서 이 구독에 대해 설정할 ApplIdentityData를 포함하는 입력 필드입니다.

PubPriority

이는 이 구독과 일치하는 모든 발행 메시지에 대한 메시지 디스크립터(MQMD)의 우선순위 필드에 있는 값입니다. MQMD의 우선순위 필드에 대한 자세한 정보는 우선순위를 참조하십시오.

값은 0보다 크거나 같아야 하며, 0은 가장 낮은 우선순위입니다. 다음 특수 값을 사용할 수도 있습니다.

- MQPRI_PRIORITY_AS_Q_DEF - MQSUB 호출의 Hobj 필드에 구독 큐가 제공되고 해당 큐가 관리되는 핸들이 아니면 메시지의 우선순위를 이 큐의 DefPriority 속성에서 가져옵니다. 이렇게 식별된 큐가 클러스터 큐이거나 큐 이름 해석 경로에 둘 이상의 정의가 있으면 MQMD의 우선순위에 대해 설명된 대로 발행 메시지를 큐에 넣을 때 우선순위가 판별됩니다. MQSUB 호출이 관리되는 핸들을 사용하면 구독된 토픽과 연관된 모델 큐의 DefPriority 속성에서 메시지의 우선순위를 가져옵니다.
- MQPRI_PRIORITY_AS_PUBLISHED - 메시지의 우선순위는 원래 발행물의 우선순위입니다. 이는 이 필드의 초기값입니다.

SubCorrelId



주의: 상관 ID는 계층이 아닌 발행/구독 클러스터의 큐 관리자 사이에만 전달될 수 있습니다.

이 구독과 일치시키기 위해 송신된 모든 발행물에는 메시지 디스크립터의 이 상관 ID가 들어 있습니다. 다중 구독이 동일한 큐를 사용하여 발행을 가져오는 경우 상관 ID로 MQGET를 사용하면 특정 구독에 대한 발행만 가져올 수 있습니다. 이 상관 ID는 큐 관리자나 사용자가 생성할 수 있습니다.

MQSO_SET_CORREL_ID 옵션을 지정하지 않으면 큐 관리자가 상관 ID를 생성하며, 이 필드는 이 구독에 대해 발행된 각 메시지에 설정될 상관 ID를 포함하는 출력 필드입니다.

MQSO_SET_CORREL_ID 옵션을 지정하면 사용자가 상관 ID를 생성하며, 이 필드는 각 발행에서 이 구독에 대해 설정할 상관 ID를 포함하는 입력 필드가 됩니다. 이 경우 이 필드에 MQCI_NONE이 포함되어 있으면 이 구독에 대해 발행된 각 메시지에 설정될 상관 ID가 원래 메시지 넣기로 작성된 상관 ID가 됩니다.

MQSO_GROUP_SUB 옵션이 지정되고 지정된 상관 ID가 동일한 큐 및 겹치는 토픽 문자열을 사용하여 그룹화된 기존 구독과 동일한 경우 그룹에서 가장 중요한 구독에만 발행물 사본이 제공됩니다.

SubUserData

이는 구독 사용자 데이터입니다. 이 필드에 제공된 구독에 대한 데이터는 이 구독에 송신된 모든 발행물의 MQSubUserData 메시지 특성으로 포함됩니다.

발행 특성

851 페이지의 표 139에는 발행 메시지와 함께 제공되는 발행 특성이 나열되어 있습니다.

MQRFH2 폴더에서 직접 이 특성에 액세스하거나, **MQINQMP**를 사용하여 해당 특성을 검색할 수 있습니다. **MQINQMP**는 조회할 특성 이름으로 해당 특성 이름 또는 **MQRFH2** 이름을 수락합니다.

표 139. 발행 특성			
특성 이름	MQRFH2 이름	유형	설명
MQTopicString	mmps.Top	MQTYPE_STRING	토픽 문자열
MQSubUserData	mmps.Sud	MQTYPE_STRING	구독자 사용자 데이터
MQIsRetained	mmps.Ret	MQTYPE_BOOLEAN	보유된 발행물
MQPubOptions	mmps.Pub	MQTYPE_INT32	발행물 옵션
MQPubLevel	mmps.Pbl	MQTYPE_INT32	발행 레벨
MQPubTime	mmpse.Pts	MQTYPE_STRING	발행 시간
MQPubSeqNum	mmpse.Seq	MQTYPE_INT32	발행 순서 번호
MQPubStrIntData	mmpse.Sid	MQTYPE_STRING	발행자가 추가한 문자열/ 정수 데이터
MQPubFormat	mmpse.Pfmt	MQTYPE_INT32	메시지 형식: MQRFH1 MQRFH2 PCF

메시지 정렬

특정 토픽의 경우, 큐 관리자는 발행 애플리케이션에서 수신되는 순서(메시지 우선순위에 따라 재정렬될 수 있음)와 같은 순서로 메시지를 발행합니다.

메시지 정렬은 일반적으로 각 구독자가 특정 발행자로부터 특정 토픽에 대한 메시지를 특정 큐 관리자에서 해당 발행자가 발행하는 순서대로 수신하는 것을 의미합니다.

그러나 모든 IBM MQ 메시지에서와 같이 경우에 따라서는 메시지를 순서 상관없이 전달할 수 있습니다. 이는 다음 상황에서 발생할 수 있습니다.

- 네트워크의 링크가 중단되고 후속 메시지가 다른 링크를 따라 다시 라우팅되는 경우
- 큐가 임시로 가득 차거나 넣지 못하게 되면 메시지가 데드-레터 큐에 넣어져서 지연되지만 후속 메시지는 곧바로 전달됩니다.
- 발행자와 구독자가 여전히 작동하고 있을 때 관리자가 큐 관리자를 삭제하면, 큐에 있는 메시지가 데드-레터 큐에 넣어지고 구독이 인터럽트됩니다.

이러한 상황이 발생할 수 없는 경우에는 발행물이 항상 순서대로 전달됩니다.

참고: 그룹화 또는 세그먼트된 메시지는 발행/구독과 함께 사용할 수 없습니다.

발행물 인터셉트

발행물이 다른 구독자에게 도달하기 전에 발행물을 인터셉트하고 수정한 후 재발행할 수 있습니다.

다음 조치 중 하나를 수행하기 위해 발행물이 구독자에게 도달하기 전에 발행물을 인터셉트할 수도 있습니다.

- 메시지에 추가 정보 첨부
- 메시지 차단
- 메시지 변환

각 메시지에 대해 동일한 조작을 수행하거나 구독, 메시지 또는 메시지 헤더에 따라서 조작을 다르게 할 수 있습니다.

관련 참조

[MQ PUBLISH_EXIT - 발행 엑시트](#)

구독 레벨

최종 구독자에 도달하기 전에 발행물을 인터셉트하도록 구독의 구독 레벨을 설정합니다. 인터셉트 구독자는 상위 구독 레벨에서 구독하고 하위 발행 레벨에서 재발행합니다. 발행물이 최종 구독자에게 전달되기 전에 발행물에 대한 메시지 처리를 수행하는 인터셉트 구독자 체인을 빌드합니다.

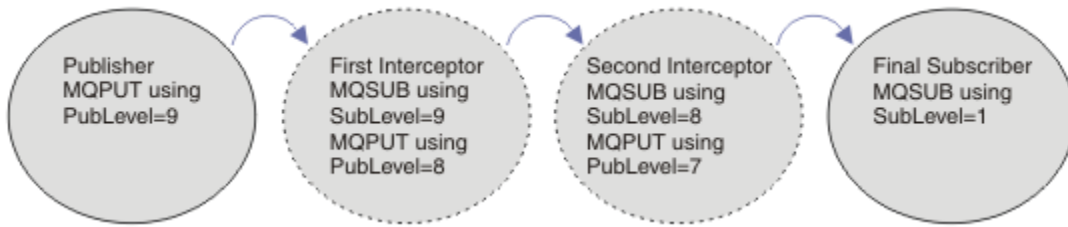


그림 116. 인터셉트 구독자의 순서

발행물을 인터셉트하려면 **MQSD** SubLevel 속성을 사용하십시오. 메시지가 인터셉트된 후에는 이를 변환한 다음 **MQPMO** PubLevel 속성을 변경하여 하위 발행 레벨에서 재발행할 수 있습니다. 그런 다음 메시지는 최종 구독자에게로 가거나 하위 구독 레벨에서 중간 구독자에 의해 다시 인터셉트됩니다.

인터셉트 구독자는 일반적으로 재발행 전에 메시지를 변환합니다. 인터셉트 구독자의 순서는 메시지 플로우를 형성합니다. 또는 인터셉트된 발행물을 재발행하지 않을 수 있습니다. 그러면 하위 구독 레벨의 구독자는 메시지를 수신하지 않습니다.

인터셉터가 다른 구독자보다 먼저 발행물을 수신하는지 확인하십시오. 인터셉터의 구독 레벨을 다른 구독자보다 높게 설정하십시오. 디폴트로 구독자의 SubLevel은 1입니다. 가장 높은 값은 9입니다. 발행물은 적어도 최고 하위 레벨만큼 높은 PubLevel로 시작해야 합니다. 처음에는 9의 기본 PubLevel로 발행하십시오.

- 토픽에 하나의 인터셉트 구독자가 있는 경우, SubLevel을 9로 설정하십시오.
- 토픽의 인터셉트 애플리케이션이 여럿인 경우, 각각의 연속 인터셉트 구독자에 대해 하위 SubLevel을 설정하십시오.
- 9부터 2까지의 구독 레벨을 사용하여 최대 8개의 인터셉트 애플리케이션을 구현할 수 있습니다. 메시지의 최종 수신인은 1의 SubLevel을 가집니다.

발행물의 PubLevel보다 낮거나 같은 최상위 구독 레벨을 가진 인터셉터가 맨 먼저 발행물을 수신합니다. 특정 구독 레벨에서 토픽의 인터셉트 구독자를 하나만 구성하십시오. 특정 구독 레벨에 여러 구독자가 있으면 결과적으로 발행물의 다중 사본이 최종 구독 애플리케이션 세트에 송신됩니다.

SubLevel이 0인 구독자는 캐치올(catchall)로 사용됩니다. 메시지를 가져오는 최종 구독자가 없으면 발행물을 수신합니다. SubLevel이 0인 구독자는 다른 구독자가 수신하지 않은 발행물을 모니터링하는 데 사용될 수 있습니다.

인터셉트 구독자 프로그래밍

852 페이지의 표 140에 설명된 구독 옵션을 사용하십시오.

표 140. 인터셉트 구독자의 구독 옵션	
구독 옵션	참고
MQSO_SET_CORREL_ID 및 SubCorrelId가 MQCI_NONE으로 설정됨	인터셉트된 발행물의 CorrelId를 원래 발행물과 동일하게 유지하십시오. 참고: 한 계층에서 발행물의 상관 ID를 전달할 수 없습니다. 이 필드는 큐 관리자에서 사용됩니다.
PubPriority가 MQPRI_PRIORITY_AS_PUBLISHED로 설정됨	인터셉트된 발행물의 우선순위를 원래 발행물과 동일하게 유지하십시오.

852 페이지의 표 140의 옵션은 모든 인터셉트 구독자가 사용해야 합니다. 그 결과, 상관 ID 및 메시지 우선순위는 원래 발행자의 설정에서 수정되지 않습니다.

인터셉트 구독자는 발행물을 처리한 후 고유 구독의 SubLevel보다 1만큼 낮은 PubLevel의 동일 토픽에 메시지를 재발행합니다. 인터셉트 구독자는 9의 SubLevel을 설정한 경우 8의 PubLevel로 메시지를 재발행합니다.

메시지를 올바르게 재발행하려면 원래 발행물에서 얻은 몇 가지 정보가 필요합니다. 원래 메시지에서도 동일한 MQMD를 재사용하고 MQPMO_PASS_ALL_CONTEXT를 설정하여 MQMD의 모든 정보가 다음 구독자에게 전달되었는지 확인하십시오. 853 페이지의 표 141에 표시된 메시지 특성의 값을 재발행된 메시지의 해당 필드로 복사하십시오. 인터셉트 구독자는 이러한 값을 변경할 수 있습니다. OR 연산자를 사용하여 추가 값을 MQPMO.Options 필드에 추가하면 메시지 넣기 옵션을 결합할 수 있습니다.

관리되는 발행 큐를 사용하지 않고 발행 큐를 명시적으로 열어야 합니다. 관리되는 큐에 대해서는 MQSO_SET_CORREL_ID를 설정할 수 없습니다. 또한 관리되는 큐에서는 MQOO_SAVE_ALL_CONTEXT도 설정할 수 없습니다. 853 페이지의 『예』에 나열된 코드 단편을 참조하십시오.

표 141. 재발행된 메시지의 MQPUT 값	
MQPUT을 사용하여 메시지 재발행	발행 메시지의 정보
MQOD. ObjectString	메시지 특성 MQTopicString
MQPMO. Options	메시지 특성 MQPubOptions

최종 구독자는 해당 구독 옵션을 다르게 설정할 수 있습니다. 예를 들어, 발행 우선순위를 MQPRI_PRIORITY_AS_PUBLISHED보다 명시적으로 설정할 수 있습니다. 최종 구독자의 설정은 체인에서 최종 인터셉트 구독자의 발행물에만 영향을 줍니다.

보유된 발행물

보유된 발행물이 인터셉트된 후에는, 원래 메시지 넣기 옵션을 재발행된 메시지로 복사하여 해당 발행물을 보존해야 합니다.

MQPMO_RETAIN 옵션은 발행자에 의해 설정됩니다. 각 가로채기 가입자는 'MQPubOptions'을 '853 페이지의 표 141'에 표시된 대로 다시 게시된 메시지의 풋메시지 옵션으로 전송해야 합니다. 메시지 넣기 옵션을 복사하면 발행물 보유 여부를 포함해서 원래 발행자가 설정한 옵션이 보존됩니다.

발행물은 인터셉트 구독자 체인 아래로 이동을 완료하고 최종 구독자에게 전달되면 마침내 보유됩니다. SubLevel 1에서 보유된 발행물을 요청하는 새 구독자는 추가 인터셉션 없이 해당 발행물을 수신합니다. 1보다 큰 SubLevel의 구독자는 보유된 발행물이 송신되지 않습니다. 그 결과, 보유된 발행물이 인터셉트 구독자 체인에서 다시 수정되지 않습니다.

예

예제는 인터셉트 구독자를 빌드하기 위해 결합될 수 있는 코드 단편입니다. 코드는 프로덕션 품질보다 간략하게 기록됩니다.

854 페이지의 그림 117의 프리프로세서 지시문은 MQINQMP MQI 호출에 필요한 발행 메시지에서 추출할 두 개의 특성을 정의합니다.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
#define MQPUBOPTIONS (MQPTR)(char*) "MQPubOptions",\
0,\
12,\
MQVS_NULL_TERMINATED,\
MQCCSI_APPL

#define MQTOPICSTRING (MQPTR)(char*) "MQTopicString",\
0,\
13,\
MQVS_NULL_TERMINATED,\
MQCCSI_APPL

```

그림 117. 프리프로세서 지시문

854 페이지의 [그림 118](#)에는 코드 단편에서 사용된 선언이 나열되어 있습니다. 강조표시된 용어를 제외하고, 선언은 IBM MQ 애플리케이션에 대한 표준입니다.

강조표시된 Put 및 Get 옵션은 모든 컨텍스트를 전달하도록 초기화됩니다. 강조표시된 MQTOPICSTRING 및 MQPUBOPTIONS는 프리프로세서 지시문에 정의된 특성 이름의 MQCHARV 초기화 프로그램입니다. 이름은 MQINQMP로 전달됩니다.

```

int main(int argc, char **argv) {
    MQLONG Reason = MQRC_NONE;
    MQLONG CompCode = MQCC_OK;
    MQHCONN Hcon = MQHC_UNUSABLE_HCONN;
    MQCHAR QMName[49] = "";
    MQCMHO CrtMsgHOpts = {MQCMHO_DEFAULT};
    MQHMSG Hmsg = MQHM_NONE;
    MQMD md = {MQMD_DEFAULT};
    MQHOBJ gHobj = MQHO_NONE;
    MQOD getOD = {MQOD_DEFAULT};
    MQGMO gmo = {MQGMO_DEFAULT};
    MQLONG GO_Options = MQOO_INPUT_AS_Q_DEF
| MQOO_FAIL_IF QUIESCING
| MQOO_SAVE_ALL_CONTEXT;

    MQLONG GC_Options = MQCO_DELETE_PURGE;
    MQHOBJ Hsub = MQHO_NONE;
    MQSD sd = {MQSD_DEFAULT};
    MQLONG SC_Options = MQCO_NONE;
    MQHOBJ pHobj = MQHO_NONE;
    MQOD putOD = {MQOD_DEFAULT};
    MQLONG PO_Options = MQOO_OUTPUT
| MQOO_FAIL_IF QUIESCING
| MQOO_PASS_ALL_CONTEXT;

    MQLONG PC_Options = MQCO_NONE;
    MQPMO pmo = {MQPMO_DEFAULT};
    MQIMPO InqPropOpts = {MQIMPO_DEFAULT};
    MQPD PropDesc = {MQPD_DEFAULT};
    MQLONG Type = MQTYPE_AS_SET;
    MQCHARV TopStrProp = {MQTOPICSTRING};
    MQCHARV PubOptProp = {MQPUBOPTIONS};
    MQLONG DataLength = 0;
    MQBYTE buffer[256] = "";
    MQLONG buflen = sizeof(buffer) - 1;
    MQLONG messlen = 0;
    char TopStrBuf[256] = "Initial value";
    int i = 0;
}

```

그림 118. 선언

선언에서 쉽게 수행되지 않는 초기화는 [855 페이지의 그림 119](#)에 표시됩니다. 강조표시된 값은 설명이 필요합니다.

SYSTEM.NDURABLE.MODEL.QUEUE

이 예제에서는 관리되는 지속 불가능 구독을 열기 위해 MQSUB를 사용하지 않고 모델 큐, SYSTEM.NDURABLE.MODEL.QUEUE를 사용하여 임시 동적 큐를 작성합니다. 해당 핸들이 MQSUB로 전달됩니다. 큐를 직접 열어 모든 메시지 컨텍스트를 저장하고 구독 옵션, MQSO_SET_CORREL_ID를 설정할 수 있습니다.

MQGMO_CURRENT_VERSION

대부분의 IBM MQ 구조에 대한 현재 버전을 사용하는 것이 중요합니다. gmo.MsgHandle과 같은 필드는 최신 버전의 제어 구조에서만 사용할 수 있습니다.

MQGMO_PROPERTIES_IN_HANDLE

원래 발행에서 설정된 메시지 넣기 옵션과 토픽 문자열은 인터셉트 구독자가 메시지 특성을 사용하여 검색할 수 있습니다. 대안은 메시지에서 **MQRFH2** 구조를 직접 읽는 것입니다.

MQSO_SET_CORREL_ID

결합에서는 MQSO_SET_CORREL_ID를 사용합니다.

```
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
```

이러한 옵션의 효과는 상관 ID를 전달하는 것입니다. 원래 발행자가 설정한 상관 ID는 인터셉트 구독자가 수신하는 발행물의 상관 ID 필드에 배치됩니다. 각 인터셉트 구독자는 동일한 상관 ID를 전달합니다. 따라서 최종 구독자는 동일한 상관 ID를 수신하는 옵션을 가집니다.

참고: 발행물이 발행/구독 계층을 통해 전달되면 상관 ID가 보유되지 않습니다.

MQPRI_PRIORITY_AS_PUBLISHED

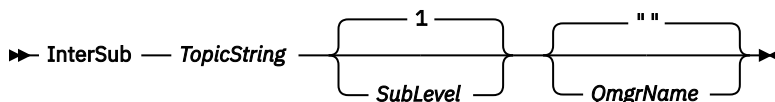
발행물은 발행되었을 때와 동일한 메시지 우선순위로 발행 큐에 배치됩니다.

```
strncpy(getOD.ObjectName, "SYSTEM.NDURABLE.MODEL.QUEUE",
        sizeof(getOD.ObjectName));
gmo.Version = MQGMO_VERSION_4;
gmo.Options = MQGMO_WAIT
              | MQGMO_PROPERTIES_IN_HANDLE
              | MQGMO_CONVERT;
gmo.WaitInterval = 30000;
sd.Options = MQSO_CREATE
            | MQSO_FAIL_IF QUIESCING
            | MQSO_SET_CORREL_ID;
sd.PubPriority = MQPRI_PRIORITY_AS_PUBLISHED;
sd.Version = MQSD_VERSION_1;
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
putOD.ObjectType = MQOT_TOPIC;
putOD.ObjectString.VSPtr = &TopStrBuf;
putOD.ObjectString.VSBufSize = sizeof(TopStrBuf);
putOD.ObjectString.VSLength = MQVS_NULL_TERMINATED;
putOD.ObjectString.VSCCSID = MQCCSI_APPL;
putOD.Version = MQOD_VERSION_4;
pmo.Version = MQPMO_VERSION_3;
```

그림 119. 초기화

856 페이지의 [그림 120](#)는 명령행 매개변수를 읽고 초기화를 완료하고 인터셉트 구독을 작성할 수 있는 코드 단편을 표시합니다.

다음 명령으로 프로그램을 실행하십시오.



각 MQI 호출의 이유 코드는 가능한 한 오류 핸들링에 방해되지 않도록 다른 배열 요소에 저장됩니다. 각 호출 후에 완료 코드가 테스트되고, 값이 MQCC_FAIL이면 제어가 do { } while(0) 코드 블록을 종료합니다.

주목할 만한 두 개의 코드 행이 있습니다.

```
pmo.PubLevel = sd.SubLevel - 1;
```

재발행된 메시지의 발행 레벨을 인터셉트 구독자의 구독 레벨보다 1만큼 작게 설정합니다.

```
gmo.MsgHandle = Hmsg;
```

MQGET에서 메시지 특성을 리턴하도록 메시지 핸들을 제공합니다.

```

do {
    printf("Intercepting subscriber start\n");
    if (argc < 2) {
        printf("Required parameter missing - topic string\n");
        exit(99);
    } else {
        sd.ObjectString.VSPtr = argv[1];
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        printf("TopicString = %s\n", sd.ObjectString.VSPtr);
    }
    if (argc > 2) {
        sd.SubLevel = atoi(argv[2]);
        pmo.PubLevel = sd.SubLevel - 1;
        printf("SubLevel is %d, PubLevel is %d\n", sd.SubLevel, pmo.PubLevel);
    }
    if (argc > 3)
        strncpy(QMName, argv[3], sizeof(QMName));
    MQCONN(QMName, &Hcon, &CompCode, &Reason);
    if (CompCode == MQCC_FAILED)
        break;
    MQOPEN(Hcon, &getOD, GO_Options, &gHobj, &CompCode, &Reason);
    if (CompCode == MQCC_FAILED)
        break;
    MQSUB(Hcon, &sd, &gHobj, &Hsub, &CompCode, &Reason);
    if (CompCode == MQCC_FAILED)
        break;
    MQCRTMH(Hcon, &CrtMsgHOpts, &Hmsg, &CompCode, &Reason);
    if (CompCode == MQCC_FAILED)
        break;
    gmo.MsgHandle = Hmsg;
}

```

그림 120. 발행물 인터셉트 준비

기본 코드 단편(857 페이지의 그림 121)은 발행 큐에서 메시지를 가져옵니다. 메시지 특성을 조회하고 토픽 문자열 및 발행의 원래 **MQPMO**.option 특성을 사용하여 메시지를 재발행합니다.

이 예제에서는 발행물의 변환이 수행되지 않습니다. 재발행된 발행물의 토픽 문자열은 인터셉트 구독자가 구독한 토픽 문자열과 항상 일치합니다. 인터셉트 구독자가 동일한 발행 큐로 송신된 여러 구독을 인터셉트해야 할 경우, 토픽 문자열을 조회하여 다른 구독과 일치하는 발행물을 구별해야 합니다.

MQINQMP에 대한 호출은 강조표시됩니다. 토픽 문자열과 발행 메시지 넣기 옵션의 특성은 출력 제어 구조에 직접 기록됩니다. putOD.ObjectString의 MQCHARV 길이 필드를 명확한 길이에서 널(Null) 종료 문자열로 대체하는 유일한 이유는 printf를 사용하여 문자열을 출력하기 위함입니다.


```

while (CompCode != MQCC_FAILED) {
    memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
    memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
    md.Encoding = MQENC_NATIVE;
    md.CodedCharSetId = MQCCSI_Q_MGR;
    printf("MQGET : %d seconds wait time\n", gmo.WaitInterval/1000);
    MQGET(Hcon, gHobj, &md, &gmo, buflen, buffer, &messlen,
        &CompCode, &Reason);
    if (CompCode == MQCC_FAILED)
        break;
    buffer[messlen] = '\0';
    MQINQMP(Hcon, Hmsg, &InqPropOpts, &TopStrProp, &PropDesc, &Type,
        putOD.ObjectString.VSBufSize, putOD.ObjectString.VSPtr,
        &(putOD.ObjectString.VSLength), &CompCode, &Reason);
    if (CompCode == MQCC_FAILED)
        break;
    memset((void *)((MQLONG)(putOD.ObjectString.VSPtr)
        + putOD.ObjectString.VSLength), '\0', 1);
    putOD.ObjectString.VSLength = MQVS_NULL_TERMINATED;
    MQINQMP(Hcon, Hmsg, &InqPropOpts, &PubOptProp, &PropDesc, &Type,
        sizeof(pmo.Options), &(pmo.Options), &DataLength,
        &CompCode, &Reason);
    if (CompCode == MQCC_FAILED)
        break;
    MQOPEN(Hcon, &putOD, PO_Options, &pHobj, &CompCode, &Reason);
    if (CompCode == MQCC_FAILED)
        break;
    printf("Republish message <%s> on topic <%s> with options %d\n",
        buffer, putOD.ObjectString.VSPtr, pmo.Options);
    MQPUT(Hcon, pHobj, &md, &pmo, messlen, buffer, &CompCode, &Reason);
    if (CompCode == MQCC_FAILED)
        break;
    MQCLOSE(Hcon, &pHobj, PC_Options, &CompCode, &Reason);
    if (CompCode == MQCC_FAILED)
        break;
}

```

그림 121. 발행물 인터셉트 및 재발행

최종 코드 단편은 [857 페이지의 그림 122](#)에 표시됩니다.

```

} while (0);
if (CompCode == MQCC_FAILED && Reason != MQRC_NO_MSG_AVAILABLE)
    printf("MQI Call failed with reason code %d\n", Reason);
if (Hsub != MQHO_NONE)
    MQCLOSE(Hcon, &Hsub, SC_Options, &CompCode, &Reason);
if (Hcon != MQHC_UNUSABLE_HCONN)
    MQDISC(&Hcon, &CompCode, &Reason);
}

```

그림 122. 완료

인터셉트 발행물 및 분신 발행/구독

인터셉트 구독자 또는 발행 엑시트를 분신 발행/구독 토폴로지에 배치할 때 단순 패턴을 따릅니다. 인터셉트 구독자를 발행자와 동일한 큐 관리자에 배치하고, 발행 엑시트를 최종 구독자와 동일한 큐 관리자에 배치합니다.

858 페이지의 그림 123에서는 발행/구독 클러스터에서 연결된 두 개의 큐 관리자를 보여줍니다. 발행자는 발행 레벨 9에서 클러스터 토픽에 대한 발행물을 작성합니다. 번호가 지정된 화살표는 발행물이 클러스터 토픽에 대한 구독자로 플로우할 때 수행하는 단계의 순서를 표시합니다. 발행물은 하위 레벨 9의 구독자에 의해 인터셉트되고 Publevel 8로 다시 발행됩니다. 발행물은 하위 레벨 8에서 구독자에 의해 다시 인터셉트됩니다. 구독자는 Publevel 7에서 재발행합니다. 큐 관리자가 제공하는 프록시 구독자는 발행물을 발행 엑시트가 최종 구독자에게도 배치된 큐 관리자 B로 전달합니다. 발행문은 Sublevel 1에서 최종 구독자가 최종적으로 수신하기 전에 발행 엑시트로 처리됩니다. 인터셉트 구독자 및 발행 엑시트는 끊어진 아웃라인으로 표시됩니다.

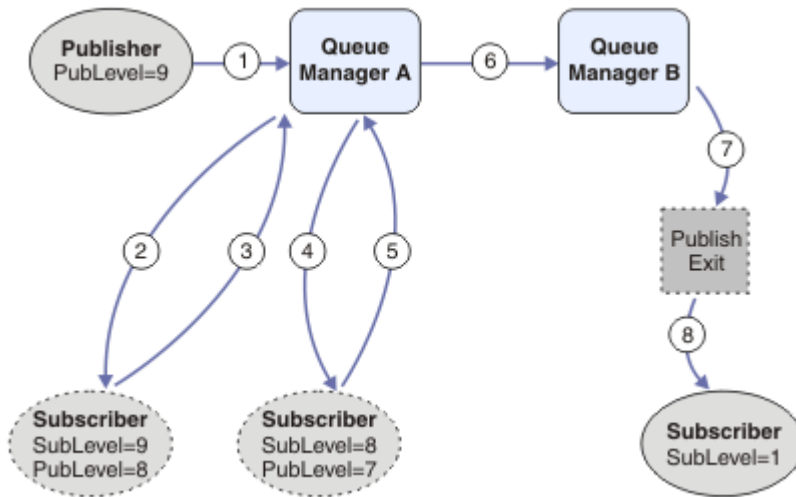


그림 123. 클러스터의 인터셉션 및 발행 엑시트

단순 패턴의 목적은 발행물을 수신하는 모든 구독자가 동일한 발행물을 수신하게 하는 것입니다. 발행물은 구독자가 연결되는 위치에 관계없이 동일한 변환 순서를 거칩니다. 발행자 또는 최종 구독자가 연결되는 위치에 따라 변환 순서가 바뀌는 것을 방지하려고 할 수도 있습니다. 합리적인 예외는 최종적으로 각 개별 구독자에게 전달되는 발행물을 사용자 조정하는 것입니다. 발행물이 최종적으로 전달되는 큐를 기반으로 발행물을 사용자 정의하려면 발행 엑시트를 사용하십시오.

분산 발행/구독 토폴로지에서 인터셉트 구독자 및 발행 엑시트를 배치할 위치를 신중하게 고려해야 합니다. 간단한 패턴은 인터셉트 구독자를 발행자와 동일한 큐 관리자에 배치하고, 발행 엑시트를 최종 구독자와 동일한 큐 관리자에 배치합니다.

안티 패턴

859 페이지의 그림 124는 단순 패턴을 따르지 않는 경우 어떻게 실패할 수 있는지 보여줍니다. 배치를 복잡하게 만들기 위해 최종 구독자를 큐 관리자 A에 추가하고 두 개의 추가 인터셉트 구독자를 큐 관리자 B에 추가했습니다.

발행물은 SubLevel 1에서 최종 구독자가 이용하기 전에 SubLevel 5에서 구독자가 인터셉트하는 PubLevel 7에서 큐 관리자 B에 전달됩니다. 발행 엑시트는 발행물이 큐 관리자 B에서 인터셉트 이용자 및 최종 이용자 모두에게 전달되기 전에 이를 인터셉트합니다. 발행물은 발행 엑시트에서 처리되지 않고 큐 관리자 A의 최종 구독자에 도착합니다.

발행/구독 토폴로지에서 프록시 구독자는 SubLevel 1에서 구독하고, 마지막 인터셉트 구독자가 설정한 PubLevel에서 전달합니다. 859 페이지의 그림 124에서는 결과적으로 발행물이 큐 관리자 B에서 구독자가 SubLevel 9를 사용하여 발행물을 인터셉트하지 않습니다.

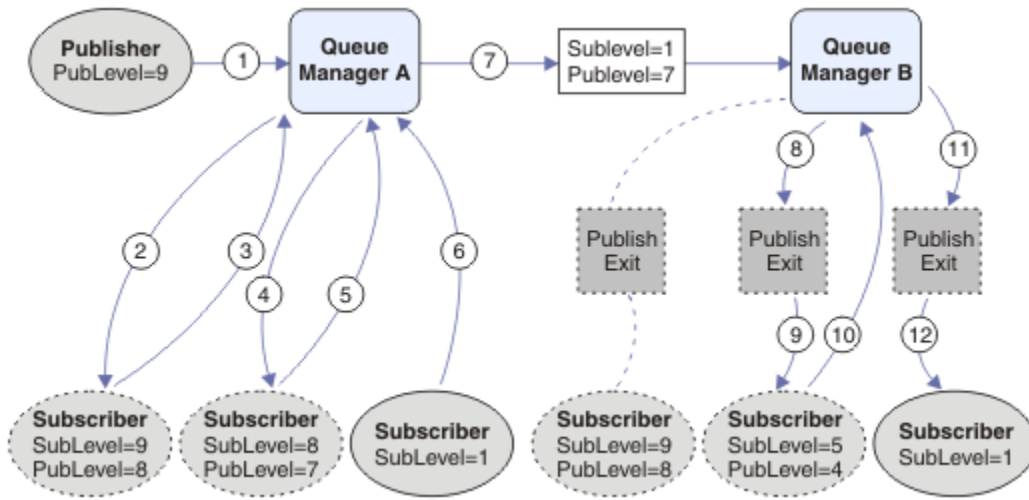


그림 124. 인터셉트 구독자의 복잡한 배치

발행 옵션

메시지를 발행하는 방법을 제어하는 여러 가지 옵션을 사용할 수 있습니다.

구독자의 회신 정보 보류

구독자가 수신한 발행물에 회신하지 못하게 하려는 경우, MQPMO_SUPPRESS_REPLYTO 메시지 넣기 옵션을 사용하여 MQMD의 ReplyToQ 및 ReplyToQmgr 필드의 정보를 보류할 수 있습니다. 이 옵션을 사용하면, 큐 관리자는 발행물을 수신할 때 MQMD에서 해당 정보를 제거한 다음 모든 구독자에게 전달합니다.

이 옵션은 ReplyToQ가 필요한 보고서 옵션과 결합하여 사용할 수 없습니다. 이 결합을 시도하면 호출이 MQRC_MISSING_REPLY_TO_Q와 함께 실패합니다.

발행 레벨

발행 레벨을 사용하는 것은 발행물을 수신하는 구독자를 제어하는 한 가지 방법입니다. 발행 레벨은 발행 대상 구독의 레벨을 선언합니다. 가장 높은 구독 레벨이 발행물의 발행 레벨 이하인 구독만 발행물을 수신합니다. 이 값은 0에서 9 사이여야 합니다. 0은 가장 낮은 발행 레벨입니다. 이 필드의 초기값은 9입니다. 발행 및 구독 레벨의 용도 중 하나는 [발행을 인터셉트](#)하는 것입니다.

발행물이 구독자에게 전달되지 않는지 확인

발행물이 구독자에게 전달되지 않았는지 확인하려면 MQPUT 호출에서 MQPMO_WARN_IF_NO_SUBS_MATCHED 메시지 넣기 옵션을 사용하십시오. 완료 코드 MQCC_WARNING 및 이유 코드 MQRC_NO_SUBS_MATCHED가 Put 조작에서 리턴되면, 발행물이 구독자에게 전달되지 않은 것입니다. MQPMO_RETAIN 옵션이 Put 조작에 지정되면 메시지가 보류되고 이후에 정의된 일치하는 구독에 전달됩니다. 분산 발행/구독 시스템에서는 큐 관리자의 토픽에 대해 등록된 프록시 구독이 없는 경우에만 MQRC_NO_SUBS_MATCHED 이유 코드가 리턴됩니다.

구독 옵션

메시지 구독을 핸들링하는 방식을 제어하는 여러 가지 옵션을 사용할 수 있습니다.

메시지 지속성

큐 관리자는 구독자에게 전달하는 발행물의 지속성을 발행자가 설정한 대로 유지보수합니다. 발행자는 다음 옵션 중 하나로 지속성을 설정합니다.

0

비지속

1

지속

발행/구독의 경우, 발행자가 토픽 오브젝트 및 **topicString**을 해석된 토픽 오브젝트로 해석합니다. 발행자가 지속성을 큐/토픽 정의로 지정하는 경우에는 발행물에 대해 해석된 토픽 오브젝트의 기본 지속성이 설정됩니다.

보유된 발행물

보유된 발행물을 수신하는 시기를 제어하기 위해 구독자가 두 구독 옵션을 사용할 수 있습니다.

요청 시 발행에만 해당(MQSO_PUBLICATIONS_ON_REQUEST)

구독자가 발행물을 수신할 때 제어할 수 있게 하려는 경우 MQSO_PUBLICATIONS_ON_REQUEST 구독 옵션을 사용할 수 있습니다. 그런 다음, 구독자는 토픽의 보유된 발행물을 송신하는 요청에 대한 MQSUBRQ 호출(원래 MQSUB 호출에서 리턴된 Hsub 핸들 지정)을 사용하여 발행물을 수신하는 시기를 제어할 수 있습니다. MQSO_PUBLICATIONS_ON_REQUEST 구독 옵션을 사용하는 구독자는 보유되지 않은 발행물을 수신하지 않습니다.

MQSO_PUBLICATIONS_ON_REQUEST를 지정하는 경우, MQSUBRQ를 사용하여 발행물을 수신해야 합니다. MQSO_PUBLICATIONS_ON_REQUEST를 사용하지 않는 경우에는 발행된 메시지를 가져옵니다.

구독자가 MQSUBRQ 호출을 사용하고 구독의 토픽에 와일드카드를 사용하는 경우, 구독이 토픽 트리에서 여러 토픽 또는 노드와 일치할 수 있습니다. 보유된 메시지(있는 경우)를 가진 이들 토픽 또는 노드는 모두 구독자에게 송신됩니다.

구독자 애플리케이션이 실행되지 않는 경우에도 지속적으로 구독한 경우 큐 관리자가 구독자에게 발행물을 계속 송신하므로 특히 구독 가능 구독에 사용되는 경우 이 옵션이 유용합니다. 이 옵션을 사용하면 구독자 큐에서 메시지가 축적될 수 있습니다. 구독자가 MQSO_PUBLICATIONS_ON_REQUEST 옵션을 사용하여 등록하는 경우에는 이러한 축적이 방지될 수 있습니다. 또는 사용자 애플리케이션에 적합한 경우 지속 불가능 구독을 사용하여 원하지 않는 메시지 축적을 방지할 수 있습니다.

구독이 지속 가능하고 발행자가 보유된 발행물을 사용하는 경우, 구독자 애플리케이션은 재시작 후 MQSUBRQ 호출을 사용하여 해당 상태 정보를 새로 고칠 수 있습니다. 그런 다음, 구독자는 MQSUBRQ 호출을 사용하여 주기적으로 해당 상태를 새로 고쳐야 합니다.

이 옵션을 사용하는 MQSUB 호출의 결과로 발행물이 송신되지 않습니다. 연결을 끊은 다음에 재개된 지속 가능 구독에서는 원래 구독이 이 옵션을 사용하도록 구성된 경우 MQSO_PUBLICATIONS_ON_REQUEST 옵션을 사용합니다.

새 발행물에만 해당(MQSO_NEW_PUBLICATIONS_ONLY)

보유된 발행물이 토픽에 있는 경우, 발행물이 작성된 후 구독한 모든 구독자는 해당 발행물의 사본을 수신합니다. 구독자가 구독 전에 작성된 발행물을 수신하지 않으려면 MQSO_NEW_PUBLICATIONS_ONLY 구독 옵션을 사용합니다.

구독 그룹화

발행물을 수신하도록 큐를 설정했으며 동일한 큐에 발행물을 공급하는 다수의 겹치는 구독이 있는 경우, 구독을 그룹화하는 방법을 고려해 보십시오. 이 상황은 겹치는 구독의 예제와 유사합니다.

토픽을 구독할 때 MQSO_GROUP_SUB 옵션을 설정하면 중복 발행물 수신을 방지할 수 있습니다. 그룹에서 둘 이상의 구독이 발행물의 토픽과 일치할 때 하나의 구독만이 큐에 해당 발행물을 배치할 수 있도록 하는 결과를 얻을 수 있습니다. 발행물 토픽과 일치한 다른 구독은 무시됩니다.

일치하는 가장 긴 토픽 문자열이 있다는 기본 전제에 따라 큐에 발행물을 배치할 수 있는 구독을 선택하면 와일드카드가 발생합니다. 이러한 구독을 가장 일치하는 구독으로 생각할 수 있습니다. 관련 특성은 MQSO_NOT_OWN_PUBS 특성이 있는지 여부를 포함하여 해당 발행물로 전파됩니다. 이러한 경우 일치하는 다른 구독이 MQSO_NOT_OWN_PUBS 특성을 가지고 있지 않더라도 발행물이 큐로 전달되지 않습니다.

중복 발행물을 제거하기 위해 단일 그룹에 모든 구독을 배치할 수는 없습니다. 그룹화된 구독은 다음 조건을 충족해야 합니다.

1. 관리되는 구독이 없습니다.
2. 구독 그룹이 발행물을 동일한 큐에 전달합니다.

3. 각 구독의 레벨이 동일해야 합니다.

4. 그룹 내의 각 구독에 대한 발행 메시지는 동일한 상관 ID를 가집니다.

각 구독의 발행 메시지가 동일한 상관 ID를 갖도록 하려면, 발행물에서 사용자 자체 상관 ID를 작성하도록 MQSO_SET_CORREL_ID를 설정하고 각 구독의 **SubCorrelId** 필드에 같은 값을 설정하십시오.

SubCorrelId를 MQCI_NONE 값으로 설정하지 마십시오.

자세한 정보는 [../refdev/q100015_.dita#q100015_/mqso_group_sub](#)의 내용을 참조하십시오.

오브젝트 속성 조회 및 설정

속성은 IBM MQ 오브젝트의 특성을 정의하는 특성입니다.

속성은 큐 관리자가 오브젝트를 처리하는 방법에 영향을 줍니다. 각 IBM MQ 오브젝트 유형의 속성은 [오브젝트의 속성](#)에 자세히 설명되어 있습니다.

일부 속성은 오브젝트를 정의할 때 설정되며, IBM MQ 명령에 의해서만 변경될 수 있습니다. 이런 속성의 예는 큐에 넣은 메시지의 기본 우선순위입니다. 기타 속성은 큐 관리자 조작의 영향을 받고 시간 경과에 따라 변경될 수 있습니다. 한 예로 큐의 현재 용량이 있습니다.

MQINQ 호출을 사용하여 대부분의 속성의 현재 값에 대해 조회할 수 있습니다. 또한 MQI는 일부 큐 속성을 변경할 수 있는 MQSET 호출을 제공합니다. MQI 호출을 사용하여 다른 유형의 오브젝트 속성을 변경할 수 없습니다. 대신 다음 자원 중 하나를 사용해야 합니다.

- ▶ **ALW** MQSC 기능(MQSC 명령에서 설명).
- ▶ **IBM i** CHGMQMx CL 명령(IBM i의 CL 명령 참조에서 설명) 또는 MQSC 기능
- ▶ **z/OS** ALTER 연산자 명령 또는 REPLACE 옵션이 있는 DEFINE 명령(MQSC 명령에서 설명).

참고: 오브젝트 속성의 이름은 MQINQ 및 MQSET 호출에서 사용되는 양식으로 이 문서에 표시됩니다. IBM MQ 명령을 사용하여 속성을 정의, 대체 또는 표시할 때, 토픽 링크의 명령에 대한 설명에 표시된 키워드를 사용하여 속성을 식별해야 합니다.

MQINQ 및 MQSET 호출은 둘 다 선택자의 배열을 사용하여 조회하거나 설정하려는 속성을 식별합니다. 작업할 수 있는 각 속성에 대한 선택자가 있습니다. 선택자 이름에는 속성의 네이처에 의해 판별되는 접두부가 있습니다.

표 142. 선택자 이름의 접두부	
접두부	설명
MQCA_	이러한 선택자는 문자 데이터(예: 큐의 이름)를 포함하는 속성을 참조합니다.
MQIA_	이러한 선택자는 숫자 값(예: 큐의 메시지 수를 나타내는 <i>CurrentQueueDepth</i>) 또는 상수 값(예: 큐 관리자가 동기점을 지원하는지 여부를 나타내는 <i>SyncPoint</i>)을 포함하는 속성을 참조합니다.

MQINQ 또는 MQSET 호출을 사용하려면 먼저 애플리케이션을 큐 관리자에 연결해야 하며, MQOPEN 호출을 사용하여 속성을 설정하거나 조회할 오브젝트를 열어야 합니다. 이러한 조작은 756 페이지의 [『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』](#) 및 763 페이지의 [『오브젝트 열기 및 닫기』](#)에서 설명합니다.

오브젝트 속성을 가져오고 조회하고 설정하는 방법에 대해 자세히 알아보려면 다음 링크를 사용하십시오.

- 862 페이지의 [『오브젝트의 속성 조회』](#)
- 863 페이지의 [『MQINQ 호출이 실패하는 몇 가지 사례』](#)
- 863 페이지의 [『큐 속성 설정』](#)

관련 개념

744 페이지의 [『MQI\(Message Queue Interface\) 개요』](#)

MQI(Message Queue Interface) 컴포넌트에 대해 학습합니다.

756 페이지의 [『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』](#)

IBM MQ 프로그래밍 서비스를 사용하려면 프로그램을 큐 관리자에 연결해야 합니다. 이 정보를 사용하여 큐 관리자에 연결하고 큐 관리자에서 연결을 끊는 방법에 대해 알아보십시오.

763 페이지의 『오브젝트 열기 및 닫기』

이 정보는 IBM MQ 오브젝트의 열기 및 닫기 조작에 대한 통찰력을 제공합니다.

772 페이지의 『큐에 메시지 넣기』

이 정보를 사용하여 메시지를 큐에 넣는 방법에 대해 알아보십시오.

786 페이지의 『큐에서 메시지 가져오기』

이 정보를 사용하여 큐에서 메시지를 가져오는 방법에 대해 알아봅니다.

863 페이지의 『작업 단위 커밋 및 백아웃』

이 정보는 작업 단위에서 발생한 복구 가능한 가져오기 및 넣기 조작을 커밋하고 백아웃하는 방법을 설명합니다.

873 페이지의 『트리거를 사용한 IBM MQ 애플리케이션 시작』

트리거와 트리거를 사용하여 IBM MQ 애플리케이션을 시작하는 방법에 대해 학습하십시오.

891 페이지의 『MQI 및 클러스터에 대한 작업』

클러스터와 관련된 호출 및 리턴 코드에 대한 특수 옵션이 있습니다.

895 페이지의 『IBM MQ for z/OS의 애플리케이션 사용 및 작성』

IBM MQ for z/OS 애플리케이션은 여러 다른 환경에서 실행되는 프로그램으로 구성할 수 있습니다. 이는 둘 이상의 환경에서 사용 가능한 기능을 활용할 수 있음을 의미합니다.

71 페이지의 『IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션』

이 정보는 IBM MQ을(를) 사용하여 IMS 애플리케이션을 작성하는 데 도움이 됩니다.

오브젝트의 속성 조회

MQINQ 호출을 사용하여 임의의 유형의 IBM MQ 속성에 대해 조회합니다.

이 호출에 대한 입력으로 다음을 제공해야 합니다.

- 연결 핸들.
- 오브젝트 핸들.
- 선택자의 수
- 각 선택자의 양식이 MQCA_* 또는 MQIA_*인 속성 선택자의 배열입니다. 각 선택자는 조회하려는 값이 있는 속성을 나타내며 각 선택자는 오브젝트 핸들이 나타내는 오브젝트 유형에 대해 유효해야 합니다. 선택자는 원하는 순서대로 지정할 수 있습니다.
- 조회할 정수 속성의 수. 정수 속성에 대해 조회하지 않으려면 0을 지정하십시오.
- 문자 속성 버퍼의 길이(CharAttrLength). 이는 각 문자 속성 문자열을 보유하는 데 필요한 길이의 합계 이상이어야 합니다. 문자 속성에 대해 조회하지 않으려면 0을 지정하십시오.

MQINQ의 출력은 다음과 같습니다.

- 배열로 복사된 정수 속성 값 세트. 값의 수는 IntAttrCount로 판별됩니다. IntAttrCount 또는 SelectorCount가 0이면 이 매개변수는 사용되지 않습니다.
- 문자 속성이 리턴되는 버퍼. 버퍼의 길이는 CharAttrLength 매개변수로 제공됩니다. CharAttrLength 또는 SelectorCount가 0이면 이 매개변수는 사용되지 않습니다.
- 완료 코드. 완료 코드가 경고를 주면 호출이 일부만 완료되었다는 의미입니다. 이 경우에는 이유 코드를 조사하십시오.
- 이유 코드. 세 가지 부분 완료 상황이 있습니다.
 - 선택자는 큐 유형에 적용되지 않습니다.
 - 정수 속성에 대해 허용된 공간이 부족합니다.
 - 문자 속성에 대해 허용된 공간이 부족합니다.

이러한 상황 중 둘 이상이 발생하면, 적용되는 첫 번째 상황이 리턴됩니다.

출력 또는 조회할 큐를 열어 해당 큐가 로컬이 아닌 클러스터 큐로 해석되면, 큐 이름, 큐 유형 및 공용 속성만 조회할 수 있습니다. 공용 속성의 값은 MQOO_BIND_ON_OPEN이 사용된 경우에 선택된 큐의 속성입니다. 값은

MQOO_BIND_NOT_FIXED 또는 MQOO_BIND_ON_GROUP이 사용되거나 MQOO_BIND_AS_Q_DEF가 사용되고 **DefBind** 큐 속성이 MQBND_BIND_NOT_FIXED인 경우에 가능한 클러스터 큐 중 임의의 큐의 값입니다. 자세한 정보는 891 페이지의 『MQOPEN 및 클러스터』 및 MQOPEN을 참조하십시오.

참고: 호출에 의해 리턴된 값은 선택된 속성의 스냅샷입니다. 프로그램이 리턴된 값에 따라 작업을 수행하기 전에 속성이 변경될 수 있습니다.

MQINQ에 MQCMIT 호출에 대한 설명이 있습니다.

MQINQ 호출이 실패하는 몇 가지 사례

속성을 조회하기 위해 별명을 여는 경우 기본 큐의 속성이 아닌 알리어스 큐(다른 큐에 액세스하는 데 사용되는 IBM MQ 오브젝트)의 속성이 리턴됩니다.

하지만 알리어스가 확인되는 기본 큐의 정의 역시 큐 관리자에서 열리므로, 다른 프로그램이 MQOPEN 및 MQINQ 호출 사이의 간격 중에 기본 큐의 사용량을 변경하는 경우 MQINQ 호출은 실패하고 MQRC_OBJECT_CHANGED 이유 코드를 리턴합니다. 이 호출은 알리어스 큐 오브젝트의 속성이 변경된 경우에도 실패합니다.

마찬가지로 속성에 대해 조회할 리모트 큐를 열면, 리모트 큐의 로컬 정의 속성만 리턴됩니다.

조회할 큐 속성의 유형에 유효하지 않은 선택자를 하나 이상 지정하면, MQINQ 호출이 경고와 함께 완료되고 다음과 같이 출력을 설정합니다.

- 정수 속성의 경우 *IntAttrs*의 해당 요소는 MQIAV_NOT_APPLICABLE로 설정됩니다.
- 문자 속성의 경우, *CharAttrs* 문자열의 해당 부분이 별표로 설정됩니다.

조회할 오브젝트 속성의 유형에 유효하지 않은 선택자를 하나 이상 지정하면, MQINQ 호출은 실패하고 MQRC_SELECTOR_ERROR 이유 코드를 리턴합니다.

MQINQ를 호출하여 모델 큐를 살펴볼 수 없으므로, MQSC 기능이나 플랫폼에서 사용 가능한 명령을 사용하십시오.

큐 속성 설정

이 정보를 사용하여 MQSET 호출을 통해 큐 속성을 설정하는 방법을 알아봅니다.

MQSET 호출을 사용하여 다음 큐 속성만 설정할 수 있습니다.

- *InhibitGet*(리모트 큐에 대한 것은 아님)

-  *DistList*

- *InhibitPut*
- *TriggerControl*
- *TriggerType*
- *TriggerDepth*
- *TriggerMsgPriority*
- *TriggerData*

MQSET 호출은 MQINQ 호출과 동일한 매개변수를 가집니다. 그러나 MQSET의 경우, 완료 코드 및 이유 코드를 제외한 모든 매개변수는 입력 매개변수입니다. 부분 완료 상황은 없습니다.

참고: MQI를 사용하면 로컬에서 정의된 큐가 아닌 다른 IBM MQ 오브젝트의 속성을 설정할 수 없습니다.

MQSET 호출에 대한 자세한 정보는 MQSET을 참조하십시오.

작업 단위 커밋 및 백아웃

이 정보는 작업 단위에서 발생한 복구 가능한 가져오기 및 넣기 조작을 커밋하고 백아웃하는 방법을 설명합니다.

이 주제에서는 다음 용어가 사용됩니다.

- 커밋

- 백아웃
- 동기점 조정
- 동기점
- 작업 단위
- 1단계 커밋
- 2단계 커밋

이러한 트랜잭션 처리 용어를 잘 알고 있는 경우 [865 페이지의 『IBM MQ 애플리케이션의 동기점 고려사항』](#)으로 건너뛸 수 있습니다.

커밋 및 백아웃

프로그램이 작업 단위 내의 큐에 메시지를 넣으면 프로그램이 작업 단위를 커밋할 때만 해당 메시지가 다른 프로그램에 표시됩니다. 작업 단위를 커밋하려면 데이터 무결성을 보존하기 위해 모든 업데이트를 완료해야 합니다. 프로그램은 오류를 감지하고 Put 조작이 영구적이 아니라고 결정하는 경우 작업 단위를 백아웃할 수 있습니다. 프로그램이 백아웃을 수행할 때 IBM MQ는 해당 작업 단위가 큐에 넣은 메시지를 제거하여 큐를 복원합니다. 프로그램이 커밋 및 백아웃 조작을 수행하는 방법은 프로그램이 실행되고 있는 환경에 따라 다릅니다.

마찬가지로, 프로그램이 작업 단위 내의 큐에서 메시지를 가져오면 프로그램이 작업 단위를 커밋할 때까지 해당 메시지는 큐에 남아 있지만 다른 프로그램에서 검색이 불가능할 수 있습니다. 메시지는 프로그램이 작업 단위를 커밋할 때 큐에서 영구적으로 삭제됩니다. 프로그램이 작업 단위를 백아웃하면, IBM MQ는 다른 프로그램이 메시지를 검색할 수 있게 하여 큐를 복원합니다.

동기점 조정, 동기점, 작업 단위

동기점 조정은 작업 단위를 커밋하거나 데이터 무결성으로 백아웃하는 프로세스입니다.

변경사항을 커밋 또는 백아웃하기 위한 의사결정은 트랜잭션의 마지막에 가장 단순한 케이스에서 내려집니다. 그러나 애플리케이션이 트랜잭션 내의 다른 논리 지점에서 데이터 변경사항을 동기화하는 것이 더 도움이 될 수 있습니다. 이러한 논리 지점은 동기점(또는 동기화 지점)이라고 하고, 두 동기점 사이에 업데이트 세트를 처리하는 기간은 작업 단위라고 합니다. 몇 개의 MQGET 호출 및 MQPUT 호출이 단일 작업 단위의 일부일 수 있습니다.

작업 단위 내의 최대 메시지 수는 [ALTER QMGR](#) 명령의 MAXUMSGS 속성으로 제어할 수 있습니다.

1단계 커밋

1단계 커밋 프로세스는 프로그램이 해당 변경사항을 다른 자원 관리자와 통합하지 않고 큐에 대한 업데이트를 커밋할 수 있는 프로세스입니다.

2단계 커밋

2단계 커밋 프로세스는 프로그램이 IBM MQ 큐에 작성한 업데이트를 다른 자원에 대한 업데이트(예를 들면, Db2의 제어를 받는 데이터베이스)와 통합할 수 있는 프로세스입니다. 이러한 프로세스 하에서 모든 자원에 대한 업데이트는 함께 커밋되거나 백아웃됩니다.

작업 단위 핸들링을 돕기 위해 IBM MQ는 **BackoutCount** 속성을 제공합니다. 이는 작업 단위 안에 있는 메시지가 백아웃될 때마다 증분됩니다. 메시지로 인해 반복해서 작업 단위가 비정상적으로 종료되는 경우, **BackoutCount**의 값은 마침내 **BackoutThreshold**의 값을 초과합니다. 이 값은 큐가 정의될 때 설정됩니다. 이 상황에서 애플리케이션은 **BackoutRequeueQName**에 정의된 대로 작업 단위에서 메시지를 제거하고 다른 큐에 넣을 수 있습니다. 메시지를 이동하면 작업 단위가 커밋될 수 있습니다.

다음 링크를 사용하여 작업 단위 커밋 및 백아웃에 대해 자세히 알아볼 수 있습니다.

- [865 페이지의 『IBM MQ 애플리케이션의 동기점 고려사항』](#)
-  [866 페이지의 『IBM MQ for z/OS 애플리케이션의 동기점』](#)
-  [868 페이지의 『CICS for IBM i 애플리케이션의 동기점』](#)
- [868 페이지의 『IBM MQ for Multiplatforms의 동기점』](#)
-  [872 페이지의 『IBM i 외부 동기점 관리자에 대한 인터페이스』](#)

관련 개념

[744 페이지의 『MQI\(Message Queue Interface\) 개요』](#)

MQI(Message Queue Interface) 컴포넌트에 대해 학습합니다.

[756 페이지의 『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』](#)

IBM MQ 프로그래밍 서비스를 사용하려면 프로그램을 큐 관리자에 연결해야 합니다. 이 정보를 사용하여 큐 관리자에 연결하고 큐 관리자에서 연결을 끊는 방법에 대해 알아보십시오.

[763 페이지의 『오브젝트 열기 및 닫기』](#)

이 정보는 IBM MQ 오브젝트의 열기 및 닫기 조작에 대한 통찰력을 제공합니다.

[772 페이지의 『큐에 메시지 넣기』](#)

이 정보를 사용하여 메시지를 큐에 넣는 방법에 대해 알아보십시오.

[786 페이지의 『큐에서 메시지 가져오기』](#)

이 정보를 사용하여 큐에서 메시지를 가져오는 방법에 대해 알아봅니다.

[861 페이지의 『오브젝트 속성 조회 및 설정』](#)

속성은 IBM MQ 오브젝트의 특성을 정의하는 특성입니다.

[873 페이지의 『트리거를 사용한 IBM MQ 애플리케이션 시작』](#)

트리거와 트리거를 사용하여 IBM MQ 애플리케이션을 시작하는 방법에 대해 학습하십시오.

[891 페이지의 『MQI 및 클러스터에 대한 작업』](#)

클러스터와 관련된 호출 및 리턴 코드에 대한 특수 옵션이 있습니다.

[895 페이지의 『IBM MQ for z/OS의 애플리케이션 사용 및 작성』](#)

IBM MQ for z/OS 애플리케이션은 여러 다른 환경에서 실행되는 프로그램으로 구성할 수 있습니다. 이는 둘 이상의 환경에서 사용 가능한 기능을 활용할 수 있음을 의미합니다.

[71 페이지의 『IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션』](#)

이 정보는 IBM MQ을(를) 사용하여 IMS 애플리케이션을 작성하는 데 도움이 됩니다.

IBM MQ 애플리케이션의 동기점 고려사항

이 정보를 사용하여 IBM MQ 애플리케이션의 동기점 사용에 대해 알아봅니다.

2단계 커미트는 다음 환경에서 지원됩니다.

- ▶ **Multi** IBM MQ for Multiplatforms
- ▶ **z/OS** CICS 트랜잭션 서버용 z/OS
- ▶ **z/OS** TXSeries
- ▶ **z/OS** IMS/ESA®
- ▶ **z/OS** RRS를 포함한 z/OS
- X/Open XA 인터페이스를 사용하는 기타 외부 조정자

1단계 커미트는 다음 환경에서 지원됩니다.

- ▶ **Multi** IBM MQ for Multiplatforms
- ▶ **z/OS** z/OS 배치

외부 인터페이스에 대한 추가 정보는 [871 페이지의 『멀티플랫폼에서 외부 동기점 관리자에 대한 인터페이스』](#) 및 Open Group에서 발행한 XA 문서 *CAE* 스펙 분산 트랜잭션 처리: XA 스펙을 참조하십시오. 트랜잭션 관리자(예: CICS, IMS, Encina 및 Tuxedo)는 기타 복구 가능한 자원과 통합된 2단계 커미트에 참여할 수 있습니다. 이는 트랜잭션 관리자가 관리하는 작업 단위의 범위 내에서 IBM MQ가 제공하는 큐잉 함수를 가져올 수 있다는 의미입니다.

IBM MQ와 함께 제공된 샘플은 XA 준수 데이터베이스를 통합하는 IBM MQ를 보여줍니다. 이 샘플에 대한 자세한 내용은 [71 페이지의 『멀티플랫폼용 샘플 절차적 프로그램 사용』](#) 1079 페이지의 『다음의 샘플 절차적 프로그램을 사용하여 z/OS』 참조하십시오.

IBM MQ 애플리케이션에서 호출이 동기점 제어를 받게 할지 여부를 모든 Put 및 Get 호출에 지정할 수 있습니다. 동기점 제어 하에서 Put 조작을 작동시키려면, MQPUT 호출 시 MQPMO 구조의 *Options* 필드에 있는 MQPMO_SYNCPOINT 값을 사용하십시오. Get 조작의 경우, MQGMO 구조의 *Options* 필드에 있는

MQGMO_SYNCPOINT 값을 사용하십시오. 옵션을 명시적으로 선택하지 않는 경우 기본 조치는 플랫폼에 따라 달라집니다.

- ▶ **Multi** 동기점 제어 기본값은 아니오입니다.
- ▶ **z/OS** 동기점 제어 기본값은 예입니다.

MQPUT1 호출이 MQPMO_SYNCPOINT로 발행되면 기본 작동이 변경되어 Put 조작이 비동기적으로 완료됩니다. 이 결과, MQOD 및 MQMD 구조의 특정 필드를 필요로 하지만 지금 정의되지 않은 값이 포함되어 있는 일부 애플리케이션의 작동이 변경될 수 있습니다. 애플리케이션은 MQPMO_SYNC_RESPONSE를 지정하여 Put 조작이 동기적으로 수행되고 모든 적절한 필드 값이 완료되는지 확인할 수 있습니다.

애플리케이션은 동기점 아래에서 MQPUT 또는 MQGET에 응답하여 MQRC_BACKED_OUT 이유 코드를 수신할 때, 일반적으로 MQBACK을 사용하여 현재 트랜잭션으로 백아웃한 다음 적절한 경우 전체 트랜잭션을 다시 시도해야 합니다. 애플리케이션은 MQCMIT 또는 MQDISC 호출에 응답하여 MQRC_BACKED_OUT을 수신하는 경우 MQBACK을 호출할 필요가 없습니다.

MQGET 호출이 백아웃될 때마다 적용된 메시지의 MQMD 구조의 *BackoutCount* 필드가 증분됩니다. *BackoutCount*는 반복적으로 백아웃된 메시지를 표시합니다. 이는 이 메시지에 대해 조사해야 할 문제점을 표시합니다. 자세한 내용은 [을 BackoutCount 참조하십시오 BackoutCount](#).

프로그램이 커밋되지 않은 요청이 있는 상태에서 MQDISC 호출을 수행하면 암시적 동기화 지점이 발생합니다 (RRS가 설정된 z/OS 배치 모드 제외). 프로그램이 비정상적으로 종료되면 암시적 백아웃이 발생합니다.

▶ **z/OS** z/OS에서 먼저 MQDISC를 호출하지 않고 프로그램이 정상적으로 종료되면 암시적 동기점도 발생합니다. MQ에 연결된 TCB가 정상적으로 종료되면 프로그램이 정상 종료된 것으로 간주됩니다. z/OS UNIX System Services 및 LE(Language Environment)에서 실행할 경우, 이상종료 또는 신호에 대해 기본 조건 핸들링이 호출됩니다. LE 조건 핸들러가 오류 조건을 처리하고 TCB가 정상적으로 종료됩니다. 이러한 조건 하에서 MQ는 작업 단위를 커밋합니다. 자세한 정보는 [LE\(Language Environment\) 조건 처리 소개](#)를 참조하십시오.

▶ **z/OS** IBM MQ for z/OS 프로그램에 대해 MQGMO_MARK_SKIP_BACKOUT 옵션을 사용하면 백아웃이 발생하는 경우 (*MQGET-error-backout* 루프 방지를 위해) 메시지가 백아웃되지 않도록 지정할 수 있습니다. 이 옵션 사용에 대한 정보는 [813 페이지의 『백아웃 건너뛰기』](#)의 내용을 참조하십시오.

(MQSET 호출 또는 명령에 의한) 큐 속성 변경사항은 작업 단위 커밋 또는 백아웃의 영향을 받지 않습니다.

▶ **z/OS** IBM MQ for z/OS 애플리케이션의 동기점

이 주제는 트랜잭션 관리자(CICS 및 IMS) 및 배치 애플리케이션에서 동기점을 사용하는 방법에 대해 설명합니다.

▶ **z/OS** CICS Transaction Server for z/OS 애플리케이션의 동기점

CICS 애플리케이션에서 EXEC CICS SYNCPOINT 명령을 사용하여 동기점을 설정합니다.

이전 동기점의 모든 변경사항을 백아웃하려면 EXEC CICS SYNCPOINT ROLLBACK 명령을 사용할 수 있습니다. 자세한 정보는 [CICS 애플리케이션 프로그래밍 참조서](#)를 참조하십시오.

다른 복구 가능한 자원이 작업 단위에 포함된 경우 큐 관리자는 CICS 동기점 관리자와 함께 2단계 커밋 프로토콜에 참여하고, 그렇지 않으면 큐 관리자는 1단계 커밋 프로세스를 수행합니다.

CICS 애플리케이션이 MQDISC 호출을 실행하는 경우 암시적 동기점이 사용되지 않습니다. 애플리케이션이 정상적으로 종료되면, 열려 있는 모든 큐가 처리완료되고 암시적 커밋이 발생합니다. 애플리케이션이 비정상적으로 종료되면, 열려 있는 모든 큐가 처리완료되고 암시적 백아웃이 발생합니다.

▶ **z/OS** IMS 애플리케이션의 동기점

IMS 애플리케이션에서, GU(get unique)와 같은 IMS 호출을 사용하여 IOPCB 및 CHPK(체크포인트)에 동기점을 설정하십시오.

이전 체크포인트 이후로 모든 변경을 백아웃하려면 IMS ROLB(롤백) 호출을 사용할 수 있습니다. 자세한 정보는 [IMS 문서](#)를 참조하십시오.

작업 단위에 다른 복구 가능한 자원도 포함된 경우 큐 관리자는 (IMS 동기점 관리자와 함께) 2단계 커밋 프로토콜에 참여합니다.

열려 있는 모든 핸들은 (배치 또는 비메시지 구동 BMP 환경을 제외한) 동기점에서 IMS 어댑터에 의해 처리완료됩니다. 이는 다른 사용자가 다음 작업 단위를 시작할 수 있고 MQPUT 또는 MQGET 호출 시가 아니라 MQCONN, MQCONNX 및 MQOPEN 호출 시에 IBM MQ 보안 검사가 수행되기 때문입니다.

그러나 입력 대기(WFI) 또는 의사 입력 대기(PWFI) 환경에서 IMS는 다음 메시지가 도착하거나 QC 상태 코드가 애플리케이션으로 리턴될 때까지 핸들을 닫도록 IBM MQ에 알리지 않습니다. 애플리케이션이 IMS 리전에서 대기 중이고 이러한 핸들 중 하나라도 트리거된 큐에 속하면 해당 큐가 열려 있기 때문에 트리거가 발생하지 않습니다. 이런 이유로, WFI 또는 PWFI 환경에서 실행 중인 애플리케이션은 다음 메시지의 IOPCB에 대해 GU를 수행하기 전에 큐 핸들을 명시적으로 MQCLOSE해야 합니다.

IMS 애플리케이션(BMP 또는 MPP)이 MQDISC 호출을 발행하면, 열린 큐는 처리완료되지만 암시적 동기점이 사용되지 않습니다. 애플리케이션이 정상적으로 종료되면, 열려 있는 모든 큐가 처리완료되고 암시적 커미트가 발생합니다. 애플리케이션이 비정상적으로 종료되면, 열려 있는 모든 큐가 처리완료되고 암시적 백아웃이 발생합니다.

▶ z/OS z/OS 배치 애플리케이션의 동기점

배치 애플리케이션의 경우, IBM MQ 동기점 관리 호출(MQCMIT 및 MQBACK)을 사용할 수 있습니다. 이전 버전과의 호환성을 위해 CSQBCMT 및 CSQBBAK는 동의어로 사용 가능합니다.

참고: 단일 작업 단위 내에서 각기 다른 자원 관리자(예: IBM MQ 및 Db2)가 관리하는 자원에 대한 업데이트를 커미트 또는 백아웃해야 할 경우 RRS를 사용할 수 있습니다. 추가 정보는 [867 페이지의 『트랜잭션 관리 및 복구 가능한 자원 관리자 서비스』](#)의 내용을 참조하십시오.

MQCMIT 호출을 사용한 변경사항 커미트

입력으로 MQCONN 또는 MQCONNX 호출에 의해 리턴되는 연결 핸들(Hconn)을 제공해야 합니다.

MQCMIT의 출력은 완료 코드 및 이유 코드입니다. 동기점이 완료되었지만 큐 관리자가 이전 동기점 이후에 Put 및 Get 조작을 백아웃한 경우, 호출은 경고와 함께 완료됩니다.

MQCMIT 호출은 정상 완료 시, 애플리케이션이 동기점에 도달하였고 이전 동기점이 영구적으로 된 후에 모든 Put 및 Get 조작이 수행되었음을 큐 관리자에 표시합니다.

모든 실패 응답이 MQCMIT가 완료되지 않았음을 의미하지는 않습니다. 예를 들어, 애플리케이션은 MQRC_CONNECTION_BROKEN을 수신할 수 있습니다.

[MQCMIT](#)에 MQCMIT 호출에 대한 설명이 있습니다.

MQBACK 호출을 사용한 변경사항 백아웃

입력으로 연결 핸들(Hconn)을 제공해야 합니다. MQCONN 또는 MQCONNX 호출에서 반환된 핸들을 사용하십시오.

MQBACK의 출력은 완료 코드 및 이유 코드입니다.

출력은 애플리케이션이 동기점에 도달하였고 마지막 동기점 이후로 수행된 모든 Get 및 Put이 백아웃되었음을 큐 관리자에 표시합니다.

[MQBACK](#)에 MQBACK 호출에 대한 설명이 있습니다.

트랜잭션 관리 및 복구 가능한 자원 관리자 서비스

트랜잭션 관리 및 복구 가능한 자원 관리자 서비스(RRS)는 참여 중인 자원 관리자를 통해 2단계 동기점 지원을 제공하는 z/OS 기능입니다.

애플리케이션은 및 IBM MQ 등 다양한 z/OS 리소스 관리자가 관리하는 Db2 복구 가능한 리소스를 업데이트한 후, 이러한 업데이트를 단일 작업 단위로 커밋하거나 롤백할 수 있습니다. RRS는 정상 실행 중에 필요한 작업 단위 상태 로깅을 제공하고, 동기점 처리를 통합하며, 서브시스템 재시작 시 적절한 작업 단위 상태 정보를 제공합니다.

IBM MQ for z/OS RRS 참가자 지원은 배치, TSO 및 Db2 저장 프로시저 환경의 애플리케이션이 단일 논리적 작업 단위 내에서 리소스와 IBM MQ 비리소스(예: Db2/IBM MQ)를 IBM MQ 모두 업데이트할 수 있도록 합니다. RRS 참가자 지원에 관한 정보는 [프로그래밍: 리소스 z/OSMVS 복구](#)를 참조하십시오.

IBM MQ 애플리케이션은 MQCMIT 및 MQBACK 또는 해당 RRS 호출 SRRCMIT 및 SRRBACK을 사용할 수 있습니다. 자세한 정보는 898 페이지의 『RRS 배치 어댑터』의 내용을 참조하십시오.

RRS 가용성

RRS가 z/OS 시스템에서 활성 상태가 아닌 경우, RRS 스텝(CSQBRSTB 또는 CSQBRSI)과 링크된 프로그램에서 발행된 IBM MQ 호출은 MQRC_ENVIRONMENT_ERROR를 리턴합니다.

Db2 스토어드 프로시저

RRS에서 Db2 스토어드 프로시저를 사용하는 경우 다음에 유의하십시오.

- RRS를 사용하는 Db2 스토어드 프로시저는 워크로드 관리자(WLM 관리)에서 관리해야 합니다.
- Db2 관리 스토어드 프로시저가 IBM MQ 호출을 포함하며 RRS 스텝(CSQBRSTB 또는 CSQBRSI)과 링크된 경우, MQCONN 또는 MQCONNx 호출은 MQRC_ENVIRONMENT_ERROR를 리턴합니다.
- WLM 관리 스토어드 프로시저가 IBM MQ 호출을 포함하며 비RRS 스텝과 링크된 경우, MQCONN 또는 MQCONNx 호출은 스토어드 프로시저 주소 공간이 시작된 이후에 실행된 첫 번째 IBM MQ 호출이 아니면 MQRC_ENVIRONMENT_ERROR를 리턴합니다.
- Db2 스토어드 프로시저가 IBM MQ 호출을 포함하고 비RRS 스텝과 링크된 경우, 해당 스토어드 프로시저에서 업데이트된 IBM MQ 자원은 스토어드 프로시저 주소 공간이 종료되거나 후속 스토어드 프로시저가 (IBM MQ 배치/TSO 스텝을 사용하여) MQCMIT를 수행할 때까지 커밋되지 않습니다.
- 동일한 스토어드 프로시저의 여러 사본은 동일한 주소 공간에서 동시에 실행될 수 있습니다. Db2가 스토어드 프로시저의 단일 사본을 사용하도록 하려면 재입력 방식으로 프로그램이 코드화되었는지 확인하십시오. 그렇지 않으면 프로그램에서 IBM MQ 호출 시 MQRC_HCONN_ERROR를 수신할 수 있습니다.
- WLM 관리 Db2 스토어드 프로시저에서 MQCMIT 또는 MQBACK은 코딩하지 마십시오.
- LE(Language Environment)에서 실행할 모든 프로그램을 설계하십시오.

IBM i CICS for IBM i 애플리케이션의 동기점

IBM MQ for IBM i는 CICS for IBM i 작업 단위에 참여합니다. IBM i용 CICS 애플리케이션 내에서 MQI를 사용하여 현재 작업 단위 내에서 메시지를 넣고 가져올 수 있습니다.

EXEC CICS SYNCPOINT 명령을 사용하여 IBM MQ for IBM i 조작을 포함하는 동기점을 설정할 수 있습니다. 이전 동기점까지 모든 변경사항을 백아웃하려면 EXEC CICS SYNCPOINT ROLLBACK 명령을 사용할 수 있습니다.

IBM i용 CICS 애플리케이션에서 MQPMO_SYNCPOINT 또는 MQGMO_SYNCPOINT 옵션이 설정된 MQPUT, MQPUT1 또는 MQGET을 사용하는 경우, IBM MQ for IBM i가 API 커밋 자원으로 해당 등록을 제거할 때까지 IBM i에 대한 CICS를 로그오프할 수 없습니다. 큐 관리자에서 연결을 끊기 전에 보류 중인 모든 Put 또는 Get 조작을 커밋하거나 백아웃하십시오. 이렇게 하면 IBM i에 대해 CICS에서 로그오프할 수 있습니다.

Multi IBM MQ for Multiplatforms의 동기점

동기점 지원은 두 가지 유형(로컬 및 글로벌)의 작업 단위에 작용합니다.

로컬 작업 단위에서는 업데이트된 유일한 자원이 IBM MQ 큐 관리자의 자원입니다. 여기서 동기점 조정은 큐 관리자 자체에서 1단계 커밋 프로시저를 통해 제공됩니다.

로컬 작업 단위에서는 다른 자원 관리자(데이터베이스)에 속하는 자원도 업데이트됩니다. IBM MQ는 이러한 작업 단위 자체를 조정할 수 있습니다. 외부 커밋 제어기로 조정할 수도 있습니다. 예를 들면, 다음과 같습니다.

- 다른 트랜잭션 관리자
-  IBM i 커밋 제어기

완전한 무결성을 보장하려면 2단계 커밋 프로시저를 사용하십시오. 2단계 커밋은 XA 준수 트랜잭션 관리자와 데이터베이스에서 제공할 수 있습니다. 예를 들면, 다음과 같습니다.

- TXSeries
- UDB
-  IBM i 커밋 제어기

 IBM MQ 제품은 2단계 커밋 프로세스를 사용하여 글로벌 작업 단위를 통합할 수 있습니다.

IBM i IBM MQ for IBM i는 WebSphere Application Server 환경 내에서 글로벌 작업 단위의 자원 관리자 역할을 할 수 있지만 트랜잭션 관리자로는 사용할 수 없습니다.

암시적 동기점

지속 메시지를 넣는 경우, IBM MQ는 동기점 아래에 지속 메시지를 넣을 수 있도록 최적화됩니다. 해당 애플리케이션에서 동기점을 사용하는 경우 여러 애플리케이션이 동일한 큐에 지속 메시지를 더 잘 넣을 수 있습니다. 이는 동기점을 사용하여 지속 메시지를 넣는 경우 큐에 대한 경합이 줄어들기 때문입니다.

ImplSyncOpenOutput은 애플리케이션이 동기점 외부에 지속 메시지를 넣는 경우 암시적 동기점을 추가합니다. 이는 애플리케이션이 암시적 동기점을 파악하지 못해도 성능을 향상시킵니다.

암시적 동기점은 큐에 대한 경합이 감소하므로 여러 애플리케이션이 큐에 넣는 경우 성능이 향상됩니다. 그러므로 **ImplSyncOpenOutput**은 암시적 동기점을 추가하기 전에 출력을 위해 큐가 열리는 최소 애플리케이션 수를 지정합니다. 기본값은 2입니다. 이는 **ImplSyncOpenOutput**을 지정하지 않으면 여러 애플리케이션이 큐에 넣는 경우에만 암시적 동기점이 추가됨을 의미합니다.

자세한 정보는 [매개변수 성능 조정](#)을 참조하십시오.

멀티플랫폼의 로컬 작업 단위

큐 관리자만 포함하는 작업 단위는 로컬 작업 단위라고 합니다. 동기점 조정은 1단계 커밋 프로세스를 사용하여 큐 관리자 자체(내부 조정)에 제공됩니다.

로컬 작업 단위를 시작하기 위해 애플리케이션은 적절한 동기점 옵션을 지정하는 MQGET, MQPUT 또는 MQPUT1 요청을 발행합니다. 작업 단위는 MQCMIT를 사용하여 커밋되거나 MQBACK을 사용하여 롤백됩니다. 그러나 애플리케이션과 큐 관리자 사이의 연결이 의도적으로 또는 의도하지 않게 끊어진 경우에도 작업 단위는 종료됩니다.

IBM MQ에 의해 통합된 글로벌 작업 단위가 여전히 활성 상태일 때 큐 관리자에서 애플리케이션 연결이 끊어지면(MQDISC) 해당 작업 단위를 커밋하려고 시도합니다. 하지만 애플리케이션이 연결을 끊지 않고 종료된 경우, 애플리케이션이 비정상적으로 종료된 것으로 간주되면 작업 단위가 롤백됩니다.

ALW 글로벌 작업 단위 AIX, Linux, and Windows

다른 자원 관리자에 속하는 자원에 대한 업데이트도 포함해야 할 경우 글로벌 작업 단위를 사용합니다. 조정 작업은 큐 관리자의 내부 또는 외부에서 수행될 수 있습니다.

내부 동기점 조정

글로벌 작업 단위의 큐 관리자 조정은 환경에서 **IBM MQ MQI client** 지원되지 않습니다.

여기서 IBM MQ는 조정을 수행합니다. 글로벌 작업 단위를 시작하기 위해 애플리케이션은 MQBEGIN 호출을 발행합니다.

MQBEGIN 호출에 대한 입력으로 MQCONN 또는 MQCONNX 호출에 의해 리턴되는 연결 핸들(Hconn)을 제공해야 합니다. 이 핸들은 IBM MQ 큐 관리자에 대한 연결을 나타냅니다.

애플리케이션은 적절한 동기점 옵션을 지정하는 MQGET 또는 MQPUT 또는 MQPUT1 요청을 발행합니다. 이는 MQBEGIN을 사용하여 로컬 자원, 다른 자원 관리자에 속하는 자원 또는 두 가지 모두를 업데이트하는 글로벌 작업 단위를 시작할 수 있음을 의미합니다. 다른 자원 관리자에 속하는 자원에 대한 업데이트는 해당 자원 관리자의 API를 사용하여 작성됩니다. 하지만 MQI를 사용하여 다른 큐 관리자에 속하는 큐를 업데이트할 수 없습니다. 추가 작업 단위(로컬 또는 글로벌)를 시작하기 전에 MQCMIT 또는 MQBACK을 발행하십시오.

글로벌 작업 단위는 MQCMIT를 사용하여 커밋됩니다. 이렇게 하면 작업 단위에 관련된 모든 자원 관리자의 2단계 커밋이 시작됩니다. 자원 관리자(예: Db2, Oracle 및 Sybase와 같은 XA 준수 데이터베이스 관리자)가 우선 커밋 준비를 요청받는 2단계 커밋 프로세스가 사용됩니다. 모두 준비된 경우에만 커밋하도록 요청됩니다. 자원 관리자가 커밋할 수 없다는 신호를 보내면 각자 커밋 대신 백아웃하도록 요청받습니다. 또는 MQBACK을 사용하여 모든 자원 관리자의 업데이트를 롤백할 수 있습니다.

글로벌 작업 단위가 여전히 활성 상태일 때 애플리케이션 연결이 끊어지면(MQDISC) 해당 작업 단위는 커밋됩니다. 하지만 애플리케이션이 연결을 끊지 않고 종료된 경우, 애플리케이션이 비정상적으로 종료된 것으로 간주되면 작업 단위가 롤백됩니다.

MQBEGIN의 출력은 완료 코드 및 이유 코드입니다.

MQBEGIN을 사용하여 글로벌 작업 단위를 시작하는 경우, 큐 관리자로 구성된 모든 외부 자원 관리자가 포함됩니다. 하지만 호출은 작업 단위를 시작하지만 다음의 경우 경고와 함께 완료됩니다.

- 참여 중인 자원 관리자가 없습니다. (즉, 큐 관리자로 구성된 자원 관리자가 없습니다.)

또는

- 하나 이상의 자원 관리자가 사용 불가능합니다.

이러한 경우, 작업 단위가 시작될 때 사용 가능한 자원 관리자의 업데이트만 작업 단위에 포함되어야 합니다.

자원 관리자 중 하나가 해당 업데이트를 커밋할 수 없는 경우, 모든 자원 관리자는 해당 업데이트를 롤백하도록 지시받고, MQCMIT가 경고와 함께 완료됩니다. 특별한 상황(일반적으로 운영자 개입)에서 일부 자원 관리자는 해당 업데이트를 커밋하지만 다른 자원 관리자가 롤백하는 경우, MQCMIT 호출은 실패할 수 있습니다. 작업이 여러 가지 결과로 완료된 것으로 간주되기 때문입니다. 이러한 발생 항목이 큐 관리자의 오류 로그에서 진단되면 시정 조치가 수행될 수 있습니다.

관련된 모든 자원 관리자가 해당 업데이트를 커밋하는 경우 글로벌 작업 단위의 MQCMIT가 성공합니다.

MQBEGIN 호출에 대한 설명은 [MQBEGIN](#)을 참조하십시오.

외부 동기점 조정

IBM MQ 이외의 동기점 조정자(예를 들면, CICS, Encina 또는 Tuxedo)가 선택된 경우에 발생합니다.

이 상황에서 필요한 만큼 커밋되지 않은 가져오기 또는 넣기 조작을 커밋 또는 롤백할 수 있도록 IBM MQ for AIX, Linux, and Windows 시스템은 작업 단위 결과의 관심사항을 동기점 조정자에 등록합니다. 외부 동기점 조정자는 1단계 또는 2단계 커밋 프로토콜의 제공 여부를 판별합니다.

외부 조정자를 사용할 때는 MQCMIT, MQBACK 및 MQBEGIN을 발행할 수 없습니다. 이러한 함수에 대한 호출은 이유 코드 MQRC_ENVIRONMENT_ERROR와 함께 실패합니다.

외부에서 통합된 작업 단위를 시작하는 방법은 동기점 조정자가 제공하는 프로그래밍 인터페이스에 따라 다릅니다. 명시적 호출이 필요할 수 있습니다. 명시적 호출이 필요하고, 작업 단위가 시작되지 않을 때 MQPMO_SYNCPOINT 옵션을 지정하는 MQPUT 호출을 발행하는 경우 완료 코드 MQRC_SYNCPOINT_NOT_AVAILABLE이 리턴됩니다.

작업 단위의 범위는 동기점 조정자에 의해 판별됩니다. 애플리케이션과 큐 관리자 사이의 연결 상태는 작업 단위의 상태가 아니라 애플리케이션이 발행하는 MQI 호출의 성공 또는 실패에 영향을 미칩니다. 애플리케이션은 예를 들어 작업 단위가 활성 상태인 동안 큐 관리자와 연결을 끊은 후 다시 연결하고 같은 작업 단위 내에서 추가 MQGET 및 MQPUT 조작을 수행할 수 있습니다. 이는 보류 중 연결 끊기라고 합니다.

CICS의 XA 기능을 사용하든 그렇지 않든, 프로그램에서 CICS API 호출을 사용할 IBM MQ 수 있습니다. XA를 사용하지 않으면, 큐에 대한 메시지 넣기 및 가져오기가 CICS의 원자적 작업 단위 내에서 관리되지 않습니다. 이 방법을 선택하는 한 가지 이유는 작업 단위의 전반적인 일관성이 중요하지 않기 때문입니다.

작업 단위의 무결성이 중요하면 XA를 사용해야 합니다. XA를 사용하는 경우, CICS는 2단계 커밋 프로토콜을 사용하여 작업 단위 내의 모든 자원이 함께 업데이트되었는지 확인합니다.

트랜잭션 지원 설정에 대한 자세한 정보는 [트랜잭션 지원 시나리오](#) 및 TXSeries CICS 문서(예: *TXSeries for Multiplatforms CICS Administration Guide for Open Systems*)를 참조하십시오.

Multi

멀티플랫폼의 암시적 동기점

암시적 동기점 지원을 사용하면 동기점 외부에서 지속 메시지 넣기가 가능합니다.

지속 메시지를 넣는 경우, IBM MQ는 동기점 아래에 지속 메시지를 넣을 수 있도록 최적화됩니다. 여러 애플리케이션이 동일한 큐에 지속 메시지를 동시에 넣는 경우 동기점을 사용하면 더 잘 수행됩니다. IBM MQ의 잠금 전략은 지속 메시지를 넣을 때 동기점이 사용되는 경우 더 효율적이기 때문입니다.

qm.ini 파일의 **ImplSyncOpenOutput** 매개변수는 애플리케이션이 동기점 외부에 지속 메시지를 넣을 때 암시적 동기점을 추가할 수 있는지 여부를 제어합니다. 이로 인해 애플리케이션이 암시적 동기점을 인식하지 못해도 성능이 향상될 수 있습니다.

암시적 동기점은 잠금 경합을 감소시키므로 여러 애플리케이션이 동시에 큐에 넣는 경우 성능을 향상시킵니다. **ImplSyncOpenOutput**은 암시적 동기점을 추가하기 전에 출력을 위해 열릴 수 있는 최소 애플리케이션 수를 지정합니다. 기본값은 2입니다. 이는 **ImplSyncOpenOutput**을 명시적으로 지정하지 않은 상태에서 여러 애플리케이션이 큐에 넣는 경우 암시적 동기점만이 추가됨을 의미합니다.

암시적 동기점을 추가하는 경우, 통계에서는 발생하는 내용을 반영하고 **runmqsc display conn**에서 트랜잭션 출력이 표시될 수 있습니다.

암시적 동기점이 추가되지 않도록 하려는 경우 **ImplSyncOpenOutput=OFF**로 설정하십시오.

자세한 정보는 [매개변수 성능 조정](#)을 참조하십시오.

멀티플랫폼에서 외부 동기점 관리자에 대한 인터페이스

IBM MQ for Multiplatforms에서는 X/Open XA 인터페이스를 사용하는 외부 동기점 관리자에 의한 트랜잭션 조정을 지원합니다.

일부 XA 트랜잭션 관리자(TXSeries)의 경우 각 XA 자원 관리자가 해당 이름을 제공해야 합니다. 이는 XA 스위치 구조에서 name이라는 문자열입니다.

-  AIX, Linux, and Windows 에서 IBM MQ 용 자원 관리자의 이름은 MQSeries_XA_RMI입니다.
-  IBM i의 경우, 자원 관리자 이름은 MQSeries XA RMI입니다.

XA 인터페이스에 대한 추가 세부사항은 The Open Group에서 발행한 XA 문서 CAE 스펙 분산 트랜잭션 처리: XA 스펙을 참조하십시오.

XA 구성에서는 IBM MQ for Multiplatforms가 XA 자원 관리자의 역할을 수행합니다. XA 동기점 조정자는 XA 자원 관리자 세트를 관리할 수 있고, 자원 관리자 모두에서 트랜잭션의 커밋 또는 백아웃을 동기화할 수 있습니다. 이는 통계적으로 등록된 자원 관리자에 대한 작동 방식입니다.

1. 애플리케이션은 트랜잭션을 시작하겠다고 동기점 조정자에 알립니다.
2. 동기점 조정자는 식별할 수 있는 모든 자원 관리자에 호출을 발행하여 현재 트랜잭션에 대해 알립니다.
3. 애플리케이션은 현재 트랜잭션과 연관된 자원 관리자가 관리하는 자원을 업데이트하도록 호출을 발행합니다.
4. 애플리케이션은 동기점 조정자가 트랜잭션을 커밋하거나 롤백하도록 요청합니다.
5. 동기점 조정자는 2단계 커밋 프로토콜을 통해 각 자원 관리자에 대한 호출을 발행하여 요청받은 대로 트랜잭션을 완료합니다.

XA 스펙에서는 각 자원 관리자가 XA 스위치라는 구조를 제공해야 합니다. 이 구조는 자원 관리자의 기능 및 동기점 조정자가 호출하는 기능을 선언합니다.

이 구조의 두 가지 버전이 있습니다.

표 143. XA 스위치 버전	
버전	설명
MQRMIASwitch	정적 XA 자원 관리
MQRMIASwitchDynamic	동적 XA 자원 관리

이 구조를 포함하는 라이브러리 목록은 [IBM MQ XA 스위치 구조](#)를 참조하십시오.

XA 동기점 조정자에 링크하는 데 사용해야 할 메소드는 조정자에 의해 정의됩니다. 해당 조정자가 IBM MQ를 통해 XA 동기점 조정자와 협업할 수 있는 방법을 판별하도록 제공한 문서를 참조하십시오.

동기점 조정자의 *xa_open* 호출로 전달되는 *xa_info* 구조는 관리되는 큐 관리자의 이름이 될 수 있습니다. 이는 MQCONN 또는 MQCONNx에 전달된 큐 관리자 이름과 같은 양식을 가지며, 기본 큐 관리자가 사용되는 경우 공백으로 둘 수 있습니다. 그러나 두 개의 추가 매개변수 TPM 및 AXLIB를 사용할 수 있습니다.

TPM을 사용하면 IBM MQ 트랜잭션 관리자 이름(예: CICS)을 지정할 수 있습니다. AXLIB를 사용하면 XA AX 시작점이 위치하는 트랜잭션 관리자에서 실제 라이브러리 이름을 지정할 수 있습니다.

이러한 매개변수나 기본이 아닌 큐 관리자를 사용하는 경우 QMNAME 매개변수를 사용하여 큐 관리자 이름을 지정해야 합니다. 추가 정보는 [xa_open 문자열의 CHANNEL, TRPTYPE, CONNAME 및 QMNAME 매개변수를 참조하십시오.](#)

제한사항

1. 글로벌 작업 단위는 공유 Hconn에서 허용되지 않습니다(760 페이지의 『MQCONN를 사용한 공유(스레드 독립) 연결』의 설명 참조).
2.  IBM MQ for IBM i는 XA 자원 관리자의 동적 등록을 지원하지 않습니다.
지원되는 유일한 트랜잭션 관리자는 WebSphere Application Server입니다.
3.  Windows 시스템에서 XA 스위치에서 선언된 모든 함수는 _cdecl 함수로 선언됩니다.
4. 외부 동기점 조정자는 동시에 하나의 큐 관리자만 관리할 수 있습니다. 이는 조정자가 각 큐 관리자에 효과적으로 연결되어 있어서 한 번에 하나의 연결만 허용되는 규칙에 따라야 하기 때문입니다.

참고: 참고: JEE 서버에서 실행 중인 JMS 클라이언트 애플리케이션 (CLIENT JEE 애플리케이션)에는 이 제한사항이 없으므로 단일 JEE 서버 관리 트랜잭션은 동일한 트랜잭션에서 여러 큐 관리자를 조정할 수 있습니다. 하지만 바인딩 모드에서 실행 중인 JMS 서버 애플리케이션은 여전히 한 번에 하나의 연결만 허용되는 규칙을 따라야 합니다.

5. 동기점 조정자를 사용하여 실행되는 모든 애플리케이션은 이미 효과적으로 해당 큐 관리자에 연결되었기 때문에 조정자가 관리하는 큐 관리자에만 연결할 수 있습니다. 연결 핸들을 확보하려면 MQCONN 또는 MQCONNX를 발행해야 하고, 종료 전에는 MQDISC를 발행해야 합니다. 또는 TXSeries CICS에 엑시트 UE014015를 사용할 수 있습니다.

IBM i 외부 동기점 관리자에 대한 인터페이스

IBM MQ for IBM i은(는) 외부 동기점 코디네이터로 기본 IBM i 커밋 제어를 사용할 수 있습니다.

스레드 독립적인(공유) 연결은 커밋 제어에서 허용되지 않습니다. 'IBM i'의 커밋 제어 기능에 대한 자세한 내용은 'IBM i 프로그래밍: 백업 및 복구 가이드, 'SC21-8079'를 참조하세요.

IBM i 커밋 제어 기능을 시작하려면 STRCMTCTL 시스템 명령을 사용하십시오. 커밋 제어를 종료하려면 ENDCMTCTL 시스템 명령을 사용하십시오.

참고: 커밋 정의 범위의 기본값은 *ACTGRP입니다. IBM MQ for IBM i에는 *JOB으로 정의되어야 합니다. 예를 들면, 다음과 같습니다.

```
STRCMTCTL LCKLVL(*ALL) CMTSCOPE(*JOB)
```

또한 IBM MQ for IBM i는 IBM MQ 자원에 대한 업데이트만을 포함하는 로컬 작업 단위를 수행할 수 있습니다. 개별 애플리케이션이 MQPMO_SYNCPOINT 또는 MQGMO_SYNCPOINT를 지정하는 MQPUT, MQPUT1 또는 MQGET을 호출하거나 MQBEGIN을 호출할 경우, 해당 애플리케이션의 로컬 작업 단위 사이에서 선택하거나 IBM i에 의해 통합된 글로벌 작업 단위에 참여할 수 있습니다. 최초의 호출이 발행될 때 커밋 제어가 활성 상태가 아닌 경우, IBM MQ는 로컬 작업 단위를 시작하고 또한 이 IBM MQ 연결에 대한 모든 추가 작업 단위는 커밋 제어가 시작되는지 여부에 관계없이 로컬 작업 단위를 사용합니다. 로컬 작업 단위를 커밋하려면 MQCMIT를 사용하십시오. 로컬 작업 단위를 백아웃하려면 MQBACK을 사용하십시오. IBM i 커밋 및 롤백 호출(예: CL 명령 COMMIT)은 IBM MQ 로컬 작업 단위에 영향을 미치지 않습니다.

네이티브 'IBM i 커밋 제어가 있는 'IBM MQ for IBM i'를 외부 동기화 지점 코디네이터로 사용하려면 커밋 제어가 있는 모든 작업이 활성화되어 있고 단일 스레드 작업에서 'IBM MQ'를 사용하고 있는지 확인하세요. 커밋 제어가 시작된 멀티스레드 작업에서 MQPMO_SYNCPOINT 또는 MQGMO_SYNCPOINT를 지정하는 MQPUT, MQPUT1 또는 MQGET을 호출하는 경우, 이유 코드 MQRC_SYNCPOINT_NOT_AVAILABLE과 함께 호출이 실패합니다.

멀티스레드 작업에서는 로컬 작업 단위와 MQCMIT 및 MQBACK 호출을 사용할 수 있습니다.

커밋 제어를 시작한 후 MQPMO_SYNCPOINT 또는 MQGMO_SYNCPOINT를 지정하는 MQPUT, MQPUT1 또는 MQGET을 호출하는 경우, IBM MQ for IBM i는 그 자체를 API 커밋 자원으로 커밋 정의에 추가합니다. 이는 일반적으로 작업에서 최초의 호출입니다. 특정 커밋 정의에 따라 등록된 API 커밋 자원이 있는 동안은 해당 정의에 대한 커밋 제어를 종료할 수 없습니다.

현재 작업 단위에 보류 중인 MQI 조작이 없는 경우, IBM MQ for IBM i는 큐 관리자로부터 연결을 끊을 때 API 커미트 자원 등록을 제거합니다.

현재 작업 단위에 보류 중인 MQPUT, MQPUT1 또는 MQGET 조작이 있는 동안 큐 관리자로부터 연결을 끊은 경우에는 IBM MQ for IBM i가 여전히 API 커미트 자원으로 등록되어 있으므로 다음 커미트 또는 롤백에 대한 알림을 받습니다. 다음 동기점에 도달하면, IBM MQ for IBM i는 필요에 따라 변경사항을 커미트하거나 롤백합니다. 애플리케이션은 작업 단위가 활성 상태인 동안 큐 관리자와 연결을 끊은 후 다시 연결하고 같은 작업 단위(보류 중 연결 끊기) 내에서 추가 MQGET 및 MQPUT 조작을 수행할 수 있습니다.

해당 커미트 정의에 대한 ENDCMTCTL 시스템 명령을 실행하려고 시도하면, 보류 중인 변경이 활성화되었음을 나타내는 메시지 CPF8355가 발행됩니다. 이 메시지는 작업이 끝날 때에도 작업 로그에 표시됩니다. 이를 방지하려면, 보류 중인 모든 IBM MQ for IBM i 조작을 커미트하거나 롤백하고 큐 관리자로부터 연결을 끊으십시오. 그러므로 ENDCMTCTL보다 먼저 COMMIT 또는 ROLLBACK 명령을 사용하면 end-commitment control이 완료될 수 있습니다.

IBM i를 외부 동기점 조정자로 사용하는 경우 MQCMIT, MQBACK 및 MQBEGIN 호출을 발행할 수 없습니다. 이러한 함수에 대한 호출은 이유 코드 MQRC_ENVIRONMENT_ERROR와 함께 실패합니다.

작업 단위를 커미트하거나 롤백하려면 커미트 제어를 지원하는 프로그래밍 언어 중 하나를 사용하십시오. 예를 들면, 다음과 같습니다.

- CL 명령: COMMIT 및 ROLLBACK
- ILE C 프로그래밍 함수: _Rcommit 및 _Rollback
- ILE RPG: COMMIT 및 ROLBK
- COBOL/400: COMMIT 및 ROLLBACK

'IBM i 커밋 컨트롤을 'IBM MQ for IBM i'와 함께 외부 동기화 지점 코디네이터로 사용하면 'IBM i'는 'IBM MQ'가 참여하는 2단계 커밋 프로토콜을 수행합니다. 각 작업 단위는 두 단계에서 커미트되기 때문에, 큐 관리자가 첫 번째 단계에서 커미트하기로 제안한 후 두 번째 단계에서 사용 불가능하게 될 수 있습니다. 예를 들어, 큐 관리자의 내부 작업이 종료되는 경우에 발생할 수 있습니다. 이런 상황에서는 커미트를 수행하는 작업 로그에 커미트 또는 롤백 조작이 실패한 것을 나타내는 메시지 CPF835F가 포함됩니다. 이에 선행하는 메시지는 문제점의 원인, 커미트 또는 롤백 조작 중 문제점 발생 여부 및 실패한 작업 단위의 논리적 작업 단위 ID(LUWID)를 표시합니다.

준비된 작업 단위의 커미트 또는 롤백 중 IBM MQ API 커미트 자원의 실패로 문제점이 발생한 경우, WRKMQMTRN 명령을 사용하여 조작을 완료하고 트랜잭션의 무결성을 복원할 수 있습니다. 이 명령을 사용하려면 커미트 및 백아웃할 작업 단위의 LUWID를 알고 있어야 합니다.

트리거를 사용한 IBM MQ 애플리케이션 시작

트리거와 트리거를 사용하여 IBM MQ 애플리케이션을 시작하는 방법에 대해 학습하십시오.

큐를 제공하는 일부 IBM MQ 애플리케이션은 계속 실행되므로 큐에 도착하는 메시지를 항상 검색할 수 있습니다. 하지만 큐에 도착하는 메시지 수를 예측할 수 없을 때 이를 원하지 않을 수 있습니다. 이러한 경우에 애플리케이션은 검색할 메시지가 없을 때에도 시스템 자원을 이용할 수 있습니다.

IBM MQ는 애플리케이션이 검색할 수 있는 메시지가 있을 때 자동으로 시작될 수 있게 하는 기능을 제공합니다. 이 기능을 트리거라고 합니다.

트리거 채널에 대한 정보는 [트리거 채널](#)을 참조하십시오.

트리거 개념

큐 관리자는 트리거 이벤트를 구성할 때 특정 조건을 정의합니다.

큐에 대해 트리거를 사용하여 트리거 이벤트가 발생하는 경우, 큐 관리자는 트리거 메시지를 이니시에이션 큐라는 큐로 송신합니다. 이니시에이션 큐에 있는 트리거 메시지는 트리거 이벤트가 발생했음을 표시합니다.

큐 관리자에서 생성된 트리거 메시지는 지속적이지 않습니다. 이는 로깅을 감소시키고(성능을 향상시킴) 재시작 중에 중복을 최소화하여 재시작 시간을 개선합니다.

이니시에이션 큐를 처리하는 프로그램은 트리거 모니터 애플리케이션이라고 하며 이 프로그램의 기능은 트리거 메시지를 읽고 트리거 메시지에 포함된 정보를 기초로 적절한 조치를 수행하는 것입니다. 일반적으로 이 조치는 트리거 메시지를 생성한 큐를 처리하기 위해 다른 애플리케이션을 시작하는 것입니다. 큐 관리자의 관점에서는

트리거 모니터 애플리케이션에 대한 특별한 점이 없으며, 이 애플리케이션은 큐(이니시에이션 큐)에서 메시지를 읽는 또 하나의 애플리케이션일 뿐입니다.

큐에 대해 트리거를 사용하는 경우 해당 큐와 연관된 프로세스 정의 오브젝트를 작성할 수 있습니다. 이 오브젝트에는 트리거 이벤트를 발생시킨 메시지를 처리하는 애플리케이션에 대한 정보가 포함되어 있습니다. 프로세스 정의 오브젝트가 작성되면, 큐 관리자가 트리거 모니터 애플리케이션에서 사용하기 위해 이 정보를 추출한 후 트리거 메시지에 배치합니다. 큐와 연관된 프로세스 정의의 이름은 *ProcessName* 로컬 큐 속성에 의해 제공됩니다. 각 큐가 다른 프로세스 정의를 지정하거나 여러 큐가 동일한 프로세스 정의를 공유할 수 있습니다.

채널의 시작을 트리거하려는 경우 프로세스 정의 오브젝트를 정의할 필요가 없습니다. 전송 큐 정의가 대신 사용됩니다.

트리거링은 에서 실행되는 IBM MQAIX, Linux, and Windows 클라이언트에서 지원됩니다. 클라이언트 환경에서 실행되는 애플리케이션은 클라이언트 라이브러리와 링크된다는 것 이외는 전체 IBM MQ 환경에서 실행되는 애플리케이션과 같습니다. 하지만 트리거 모니터 및 시작할 애플리케이션 모두 동일한 환경에 있어야 합니다.

트리거에는 다음이 포함됩니다.

애플리케이션 큐

애플리케이션 큐는 트리거가 설정되고 조건이 충족될 때 트리거 메시지가 기록되어야 할 로컬 큐입니다.

프로세스 정의

애플리케이션 큐는 프로세스 정의 오브젝트가 연관되어 있을 수 있으며 이 오브젝트는 애플리케이션 큐에서 메시지를 가져올 애플리케이션의 세부사항을 보유합니다. (속성 목록은 [프로세스 정의 속성](#)을 참조하십시오.)

채널의 시작을 트리거하려는 경우 프로세스 정의 오브젝트를 정의할 필요는 없음을 기억하십시오.

전송 큐

채널의 시작을 트리거하려는 경우 전송 큐가 필요합니다.

Linux 이외 플랫폼에 있는 전송 큐의 경우, 전송 큐의 *TriggerData* 속성은 시작할 채널 이름을 지정할 수 있습니다. 이는 트리거 채널에 대한 프로세스 정의를 바꿀 수 있지만 프로세스 정의가 작성되지 않을 때에만 사용됩니다.

트리거 이벤트

트리거 이벤트는 트리거 메시지가 큐 관리자에 의해 생성되도록 하는 이벤트입니다. 이는 일반적으로 애플리케이션 큐에 도착하는 메시지이지만 다른 때에 발생할 수도 있습니다. 예를 들어, [879 페이지의 『트리거 이벤트의 조건』](#)의 내용을 참조하십시오.

IBM MQ는 트리거 이벤트를 발생시키는 조건을 제어할 수 있는 다양한 옵션이 있습니다([883 페이지의 『트리거 이벤트 제어』](#) 참조).

트리거 메시지

트리거 이벤트를 인식할 때 큐 관리자는 트리거 메시지를 작성합니다. 이 트리거 메시지는 시작할 애플리케이션에 대한 트리거 메시지 정보에 복사됩니다. 이 정보는 애플리케이션 큐 및 이 애플리케이션 큐와 연관된 프로세스 정의 오브젝트에 있습니다.

트리거 메시지는 고정된 형식을 가지고 있습니다([889 페이지의 『트리거 메시지의 형식』](#) 참조).

이니시에이션 큐

이니시에이션 큐는 큐 관리자가 트리거 메시지를 넣는 로컬 큐입니다. 이니시에이션 큐가 알리어스 큐 또는 모델 큐가 될 수 없음을 유의하십시오.

큐 관리자는 둘 이상의 이니시에이션 큐를 소유할 수 있으며 각 이니시에이션 큐는 하나 이상의 애플리케이션 큐와 연관됩니다.

 큐 공유 그룹에서 큐 관리자가 액세스할 수 있는 로컬 큐인 공유 큐는 IBM MQ for z/OS에서 이니시에이션 큐가 될 수 있습니다.

트리거 모니터

트리거 모니터는 하나 이상의 이니시에이션 큐에 서비스를 제공하면서 연속 실행되는 프로그램입니다. 트리거 메시지가 이니시에이션 큐에 도착하면 트리거 모니터가 메시지를 검색합니다. 트리거 모니터는 트리거 메시지의 정보를 사용합니다. 트리거 모니터는 애플리케이션 시작 명령을 실행하여 애플리케이션 큐에 도착하는 메시지를 검색하고 이 메시지를 애플리케이션 큐의 이름을 포함하는 트리거 메시지 헤더에 포함된 정보에 전달합니다.

모든 플랫폼에서 채널 시작기라고 하는 특수 트리거 모니터가 채널을 시작합니다.

z/OS z/OS에서, 채널 시작기는 일반적으로 수동으로 시작하거나 큐 관리자 시작 JCL에서 CSQINP2를 변경하여 큐 관리자가 시작될 때 자동으로 수행할 수 있습니다.

Multi 다중 플랫폼에서 채널 시작기는 큐 관리자가 시작될 때 자동으로 시작되거나 **runmqchi** 명령을 사용하여 수동으로 시작할 수 있습니다.

자세한 정보는 886 페이지의 『트리거 모니터에 의한 이니시에이션 큐 처리』의 내용을 참조하십시오.

트리거가 작동하는 방식을 이해하려면 트리거 유형 FIRST(MQTT_FIRST)의 예제(875 페이지의 그림 125)를 고려하십시오.

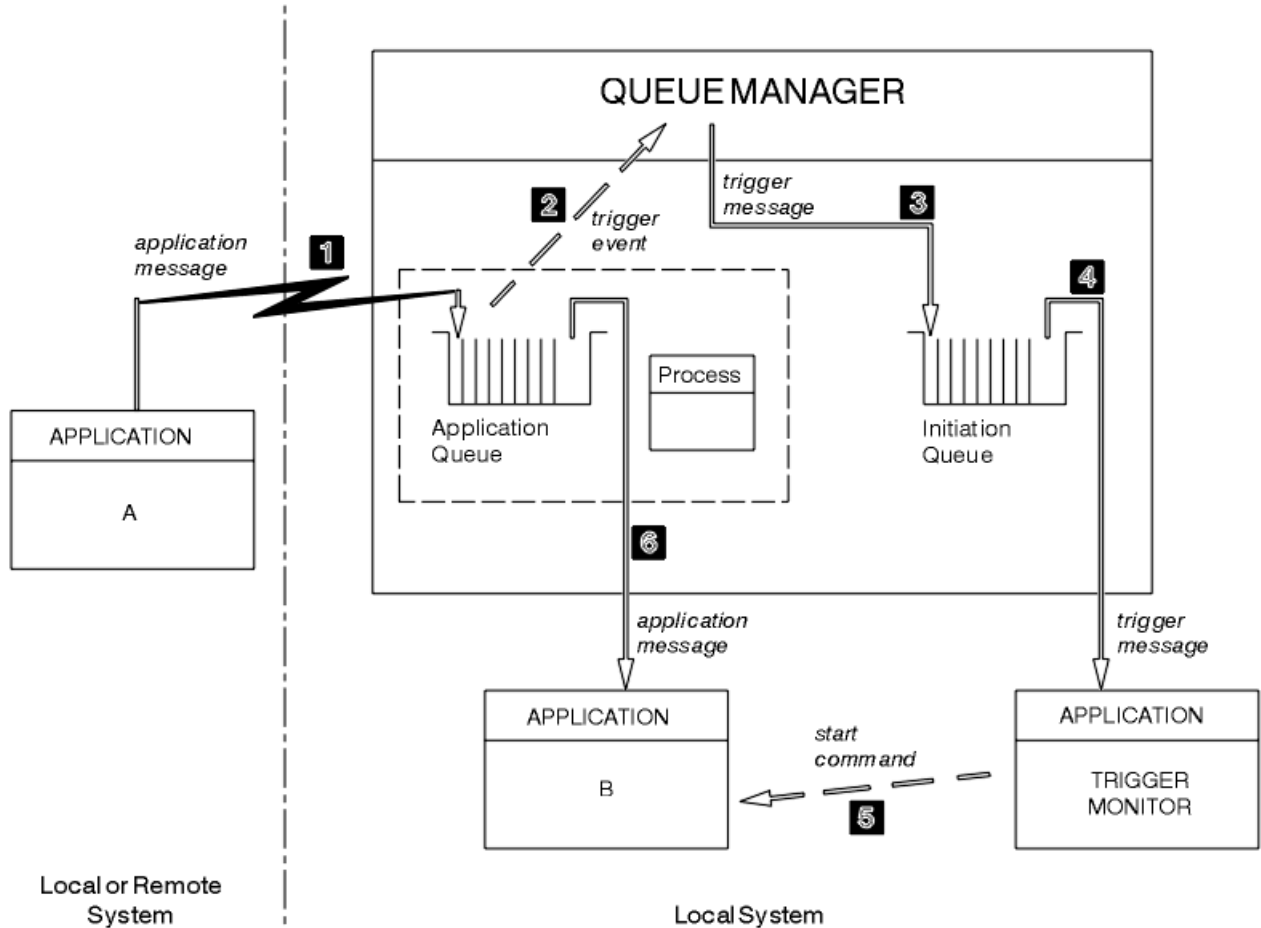


그림 125. 애플리케이션 및 트리거 메시지의 플로우

875 페이지의 그림 125에서 이벤트의 순서는 다음과 같습니다.

1. 큐 관리자에 대해 로컬 또는 리모트일 수 있는 애플리케이션 A는 메시지를 애플리케이션 큐에 넣습니다. 입력을 위해 이 큐를 열어 놓는 애플리케이션은 없습니다. 하지만 이러한 사실은 트리거 유형 FIRST 및 DEPTH에만 관련됩니다.
2. 큐 관리자는 트리거 이벤트를 생성해야 하는 조건이 충족되는지 확인합니다. 이러한 조건이 충족되면 트리거 이벤트가 생성됩니다. 연관된 프로세스 정의 오브젝트 내에 보유된 정보는 트리거 메시지 작성 시에 사용됩니다.
3. 큐 관리자는 트리거 메시지를 작성하고 이 메시지를 이 애플리케이션 큐와 연관된 이니시에이션 큐에 넣습니다. 하지만 애플리케이션(트리거 모니터)이 입력을 위해 이니시에이션 큐를 열어 놓은 경우에만 해당됩니다.
4. 트리거 모니터는 이니시에이션 큐에서 트리거 메시지를 검색합니다.
5. 트리거 모니터는 애플리케이션 B(서버 애플리케이션)를 시작하는 명령을 실행합니다.
6. 애플리케이션 B는 애플리케이션 큐를 열고 메시지를 검색합니다.

참고:

1. 애플리케이션 큐가 모든 프로그램에서 입력할 수 있도록 열려 있고 FIRST 또는 DEPTH에 대해 트리거가 설정된 경우, 큐가 이미 제공되고 있는 상태이기 때문에 트리거 이벤트가 발생하지 않습니다.
2. 이니시에이션 큐가 입력을 위해 열린 상태가 아닌 경우, 큐 관리자는 트리거 메시지를 생성하지 않으며 애플리케이션이 입력을 위해 이니시에이션 큐를 열 때까지 대기합니다.
3. 채널에 대해 트리거를 사용하는 경우 트리거 유형 FIRST 또는 DEPTH를 사용하십시오.
4. 트리거된 애플리케이션은 트리거 모니터를 시작한 사용자, CICS 사용자 또는 큐 관리자를 시작한 사용자의 사용자 ID와 그룹으로 실행됩니다.

지금까지 트리거 내에 있는 큐 사이의 관계는 일대일의 관계였습니다. 876 페이지의 그림 126의 내용을 고려하십시오.

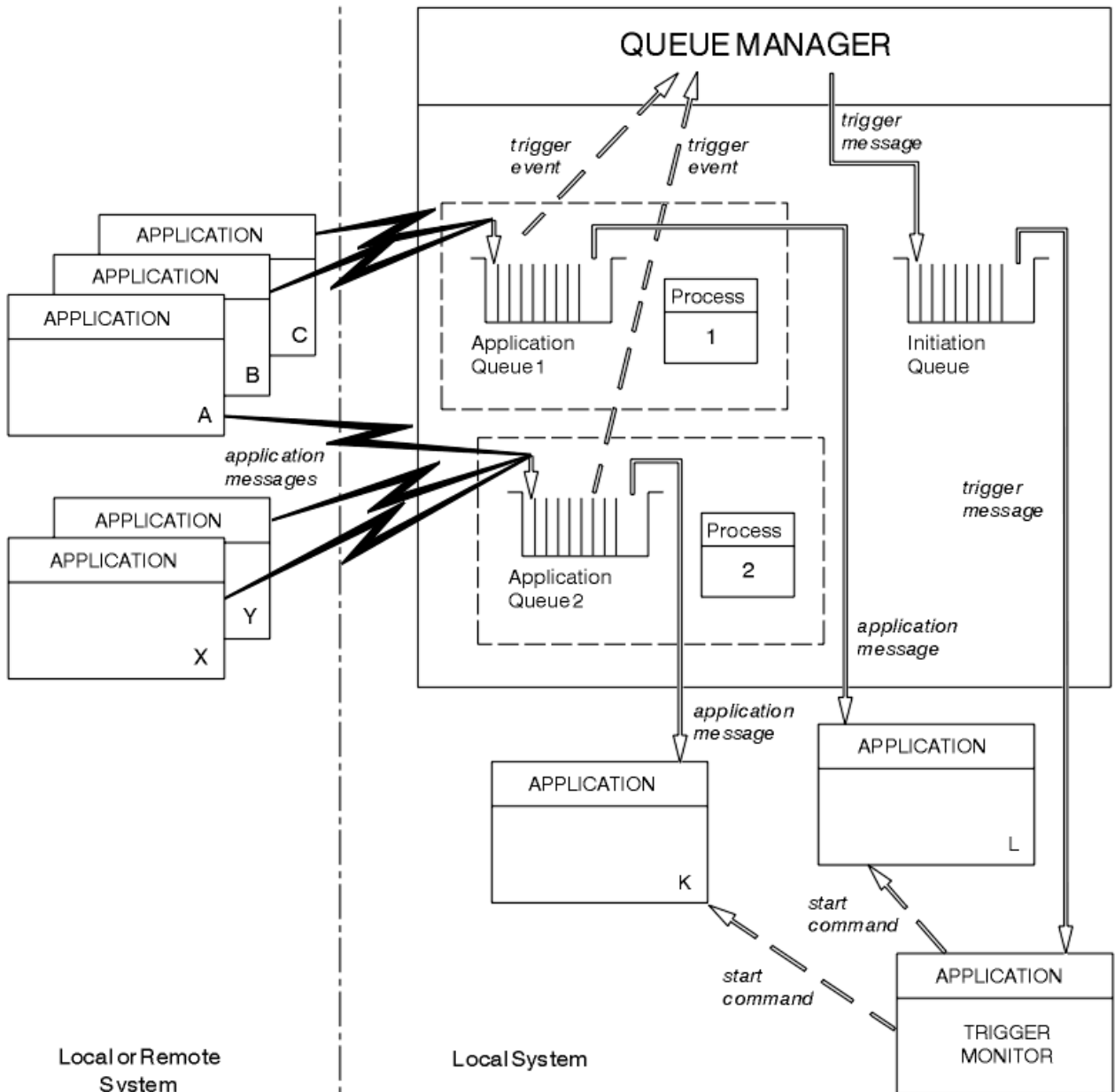


그림 126. 트리거 내에 있는 큐의 관계

애플리케이션 큐는 프로세스 정의 오브젝트가 연관되어 있으며 이 오브젝트는 메시지를 처리할 애플리케이션의 세부사항을 보유합니다. 큐 관리자는 정보를 트리거 메시지에 배치하므로 하나의 이니시에이션 큐만 필요합니다. 트리거 모니터는 이 정보를 트리거 메시지에서 추출하고 관련 애플리케이션을 시작하여 각 애플리케이션 큐에 있는 메시지를 처리합니다.

채널의 시작을 트리거하려는 경우 프로세스 정의 오브젝트를 정의할 필요가 없음을 기억하십시오. 전송 큐 정의는 트리거할 채널을 판별할 수 있습니다.

트리거를 사용하여 IBM MQ 애플리케이션을 시작하는 방법에 대한 자세한 정보를 얻으려면 다음 링크를 사용하십시오.

- [877 페이지의 『트리거 필수조건』](#)
- [879 페이지의 『트리거 이벤트의 조건』](#)
- [883 페이지의 『트리거 이벤트 제어』](#)
- [885 페이지의 『트리거된 큐를 사용하는 애플리케이션 설계』](#)
- [886 페이지의 『트리거 모니터에 의한 이니시에이션 큐 처리』](#)
- [889 페이지의 『트리거 메시지의 특성』](#)
- [890 페이지의 『트리거가 작동하지 않는 경우』](#)

관련 개념

[744 페이지의 『MQI\(Message Queue Interface\) 개요』](#)
MQI(Message Queue Interface) 컴포넌트에 대해 학습합니다.

[756 페이지의 『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』](#)
IBM MQ 프로그래밍 서비스를 사용하려면 프로그램을 큐 관리자에 연결해야 합니다. 이 정보를 사용하여 큐 관리자에 연결하고 큐 관리자에서 연결을 끊는 방법에 대해 알아보십시오.

[763 페이지의 『오브젝트 열기 및 닫기』](#)
이 정보는 IBM MQ 오브젝트의 열기 및 닫기 조작에 대한 통찰력을 제공합니다.

[772 페이지의 『큐에 메시지 넣기』](#)
이 정보를 사용하여 메시지를 큐에 넣는 방법에 대해 알아보십시오.

[786 페이지의 『큐에서 메시지 가져오기』](#)
이 정보를 사용하여 큐에서 메시지를 가져오는 방법에 대해 알아봅니다.

[861 페이지의 『오브젝트 속성 조회 및 설정』](#)
속성은 IBM MQ 오브젝트의 특성을 정의하는 특성입니다.

[863 페이지의 『작업 단위 커밋 및 백아웃』](#)
이 정보는 작업 단위에서 발생한 복구 가능한 가져오기 및 넣기 조작을 커밋하고 백아웃하는 방법을 설명합니다.

[891 페이지의 『MQI 및 클러스터에 대한 작업』](#)
클러스터와 관련된 호출 및 리턴 코드에 대한 특수 옵션이 있습니다.

[895 페이지의 『IBM MQ for z/OS의 애플리케이션 사용 및 작성』](#)
IBM MQ for z/OS 애플리케이션은 여러 다른 환경에서 실행되는 프로그램으로 구성할 수 있습니다. 이는 둘 이상의 환경에서 사용 가능한 기능을 활용할 수 있음을 의미합니다.

[71 페이지의 『IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션』](#)
이 정보는 IBM MQ을(를) 사용하여 IMS 애플리케이션을 작성하는 데 도움이 됩니다.

트리거 필수조건

이 정보를 사용하여 트리거를 사용하기 전에 수행해야 단계에 대해 알아봅니다.

애플리케이션이 트리거를 이용할 수 있게 하려면 먼저 다음 단계를 완료하십시오.

1. 다음 중 하나입니다.

- a. 애플리케이션 큐에 대한 이니시에이션 큐를 작성하십시오. 예를 들면, 다음과 같습니다.

```
DEFINE QLOCAL (initiation.queue) REPLACE +  
LIKE (SYSTEM.DEFAULT.INITIATION.QUEUE) +  
DESCR ('initiation queue description')
```

또는

b. 존재하며 애플리케이션(일반적으로, 이 이름은 SYSTEM.DEFAULT.INITIATION.QUEUE 또는 트리거로 채널을 시작할 경우 SYSTEM.CHANNEL.INITQ임)에서 사용할 수 있는 로컬 큐의 이름을 판별하고 애플리케이션 큐의 *InitiationQName* 필드에 해당 이름을 지정하십시오.

2. 애플리케이션 큐와 이니시에이션 큐를 연관시키십시오. 큐 관리자는 둘 이상의 이니시에이션 큐를 소유할 수 있습니다. 일부 애플리케이션 큐는 각기 다른 애플리케이션에서 제공하도록 할 수 있습니다. 이 경우 각 제공 프로그램에 대해 하나의 이니시에이션 큐를 사용할 수 있지만 꼭 그렇게 할 필요는 없습니다. 애플리케이션 큐를 작성하는 방법의 예는 다음과 같습니다.

```
DEFINE QLOCAL (application.queue) REPLACE +
LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE) +
DESCR ('appl queue description') +
INITQ (initiation.queue) +
PROCESS (process.name) +
TRIGGER +
TRIGTYPE (FIRST)
```

IBM i 다음은 이니시에이션 큐를 작성하는 IBM MQ for IBM i의 CL 프로그램으로부터 추출한 내용입니다.

```
/* Queue used by AMQSINQA */
CRTMQMQ QNAME('SYSTEM.SAMPLE.INQ') +
QTYPE(*LCL) REPLACE(*YES) +
MQMNAME +
TEXT('queue for AMQSINQA') +
SHARE(*YES) /* Shareable */+
DFTMSGPST(*YES)/* Persistent messages OK */+
TRGENBL(*YES) /* Trigger control on */+
TRGTYPE(*FIRST)/* Trigger on first message*/+
PRCNAME('SYSTEM.SAMPLE.INQPROCESS') +
INITQNAME('SYSTEM.SAMPLE.TRIGGER')
```

3. 애플리케이션을 트리거할 경우, 애플리케이션 큐를 제공할 수 있도록 애플리케이션에 관한 정보를 포함하는 프로세스 정의 오브젝트를 작성하십시오. 예를 들면, PAYR이라는 CICS 급여 트랜잭션 트리거 시작 방법은 다음과 같습니다.

```
DEFINE PROCESS (process.name) +
REPLACE +
DESCR ('process description') +
APPLICID ('PAYR') +
APPLTYPE (CICS) +
USERDATA ('Payroll data')
```

IBM i 다음은 프로세스 정의 오브젝트를 작성하는 IBM MQ for IBM i의 CL 프로그램으로부터 추출한 내용입니다.

```
/* Process definition */
CRTQMPPRC PRCNAME('SYSTEM.SAMPLE.INQPROCESS') +
REPLACE(*YES) +
MQMNAME +
TEXT('trigger process for AMQSINQA') +
ENVDATA('JOBPTY(3)') /* Submit parameter */+
APPID('AMQSINQA') /* Program name */
```

큐 관리자는 트리거 메시지를 작성할 때 프로세스 정의 오브젝트의 속성 정보를 트리거 메시지로 복사합니다.

플랫폼	프로세스 정의 오브젝트 작성 방법
AIX, Linux, and Windows 시스템	DEFINE PROCESS를 사용하거나 SYSTEM.DEFAULT.PROCESS를 사용하고 ALTER PROCESS를 사용하여 수정합니다.
 z/OS	DEFINE PROCESS(878 페이지의 『3』 단계의 샘플 코드 참조)를 사용하거나 조작 및 제어 패널을 사용합니다.

플랫폼	프로세스 정의 오브젝트 작성 방법
  IBM i	878 페이지의 『3』 단계에서와 같이 CL 프로그램에 포함된 코드를 사용합니다.

4. 선택사항: 전송 큐 정의를 작성하고 **ProcessName** 속성에 공백을 사용하십시오.

이 **TrigData** 속성에는 트리거할 채널의 이름을 포함할 수 있으며, 비워둘 수도 있습니다. 단, 다음을 제외하고 IBM MQ for z/OS, 비워두면 채널 시작자는 지정된 전송 큐와 연관된 채널을 찾을 때까지 채널 정의 파일을 검색합니다. 큐 관리자는 트리거 메시지를 작성할 때 전송 큐 정의의 **TrigData** 속성 정보를 트리거 메시지로 복사합니다.

5. 애플리케이션 큐를 제공하도록 애플리케이션 특성을 지정하는 프로세스 정의 오브젝트를 작성한 경우, 큐의 **ProcessName** 속성에 해당 이름을 지정하여 애플리케이션 큐와 프로세스 오브젝트를 연관시키십시오.

플랫폼	사용 명령
AIX, Linux, and Windows 시스템	ALTER QLOCAL
  z/OS	ALTER QLOCAL
  IBM i	CHGMQMQ

6. 트리거 모니터  (또는 IBM MQ for IBM i의 트리거 서버)의 인스턴스를 시작하여 사용자가 정의한 이니시이션 큐를 제공하십시오. 자세한 정보는 886 페이지의 『트리거 모니터에 의한 이니시이션 큐 처리』의 내용을 참조하십시오.

전달되지 않는 모든 트리거 메시지를 인식하려면 큐 관리자의 데드-레터(전달되지 않은 메시지) 큐가 정의되어 있는지 확인하십시오. **DeadLetterQName** 큐 관리자 필드에서 큐의 이름을 지정하십시오.

그런 다음, 애플리케이션 큐를 정의하는 큐 오브젝트의 속성을 사용하여 필요한 트리거 조건을 설정할 수 있습니다. 자세한 정보는 883 페이지의 『트리거 이벤트 제어』의 내용을 참조하십시오.

트리거 이벤트의 조건

큐 관리자는 특정 조건이 충족될 때 트리거 메시지를 생성합니다.

다음 조건은 큐 관리자가 트리거 메시지를 작성하도록 야기합니다.

1. 메시지가 큐에 넣어집니다.
2. 메시지가 큐의 임계값 트리거 우선순위보다 크거나 같은 우선순위를 가집니다. 이 우선순위는 **TriggerMsgPriority** 로컬 큐 속성에 설정되며, 0으로 설정된 경우 모든 메시지가 규정됩니다.
3. **TriggerMsgPriority**보다 크거나 같은 우선순위를 가진 큐의 메시지 수는 이전에는 **TriggerType**에 따라 다음과 같았습니다.
 - 0(트리거 유형 MQTT_FIRST)
 - 임의의 수(트리거 유형 MQTT EVERY)
 - **TriggerDepth** - 1(트리거 유형 MQTT_DEPTH)

참고:

- 비공유 로컬 큐의 경우, 큐 관리자는 트리거 이벤트에 대한 조건이 존재하는지 여부를 평가할 때 커밋된 메시지와 커밋되지 않은 메시지를 모두 계수합니다. 따라서 애플리케이션은 큐에 있는 메시지가 커밋되지 않았기 때문에 검색할 메시지가 없을 때 시작될 수 있습니다. 이러한 경우, 적당한 **WaitInterval**을 포함한 대기 옵션을 사용하여 애플리케이션이 해당 메시지가 도착할 때까지 대기하도록 하는 방안을 고려하십시오.
-  로컬 공유 큐의 경우, 큐 관리자는 커밋된 메시지만 계수합니다.

4. 트리거 유형이 FIRST 또는 DEPTH인 경우, 메시지를 제거하기 위해 애플리케이션 큐를 여는 프로그램이 없습니다(즉, **OpenInputCount** 로컬 큐 속성이 0임).

참고: z/OS

- 공유 큐의 경우, 다중 큐 관리자가 큐에 대해 실행 중인 트리거 모니터를 가지고 있을 때 특별한 조건이 적용됩니다. 이 상황에서 하나 이상의 큐 관리자가 입력을 공유하기 위해 큐를 열면, 다른 큐 관리자의 트리거 기준이 *TriggerType* MQTT_FIRST 및 *TriggerMsgPriority* 0으로 처리됩니다. 모든 큐 관리자가 입력을 위해 큐를 닫을 때 트리거 조건은 큐 정의에 지정된 해당 조건으로 되돌아갑니다.

이 조건의 영향을 받는 예제 시나리오는 애플리케이션 큐 A에 대해 실행하는 트리거 모니터가 있는 다중 큐 관리자 QM1, QM2 및 QM3입니다. 트리거링의 조건을 충족하는 A에 메시지가 도착하고 이니시에이션 큐에서 트리거 메시지가 생성됩니다. QM1의 트리거 모니터는 트리거 메시지를 가져오고 애플리케이션을 트리거합니다. 트리거된 애플리케이션이 공유 입력을 위해 애플리케이션 큐를 엽니다. 이 시점부터 애플리케이션 큐 A의 트리거 조건은 *TriggerType* MQTT_FIRST로 평가되며, QM1이 애플리케이션 큐를 닫을 때까지 큐 관리자 QM2 및 QM3의 *TriggerMsgPriority*는 0입니다.

- 공유 큐의 경우, 이 조건은 각 큐 관리자에 대해 적용됩니다. 즉, 해당 큐 관리자가 큐에 대한 트리거 메시지를 생성할 수 있도록 하려면 큐에 대해 큐 관리자의 *OpenInputCount*가 0이 되어야 합니다. 그러나 큐 공유 그룹의 큐 관리자가 MQOO_INPUT_EXCLUSIVE 옵션을 사용하여 큐를 연 경우, 큐 공유 그룹의 큐 관리자는 해당 큐에 대해 아무 트리거 메시지도 생성하지 않습니다.

트리거된 애플리케이션이 입력을 위해 큐를 열 때 트리거 조건이 평가되는 방법이 변경됩니다. 실행 중인 트리거 모니터가 하나밖에 없는 시나리오에서는 다른 애플리케이션도 마찬가지로 입력을 위해 애플리케이션 큐를 열기 때문에 동일한 효과를 얻을 수 있습니다. 애플리케이션 큐를 연 애플리케이션이 트리거 모니터에서 시작된 애플리케이션인지 다른 애플리케이션인지는 문제가 되지 않습니다. 중요한 것은 트리거 기준을 변경시키는 또 다른 큐 관리자에서 입력을 위해 큐를 열었다는 사실입니다.

5. z/OS IBM MQ for z/OS에서 애플리케이션 큐가 MQUS_NORMAL의 **Usage** 속성을 갖는 큐인 경우, 해당 큐에 대한 Get 요청이 억제되지 않습니다(즉, **InhibitGet** 큐 속성이 MQQA_GET_ALLOWED임). 또한 트리거된 애플리케이션 큐가 MQUS_XMITQ의 **Usage** 속성을 갖는 큐인 경우, 해당 큐에 대한 가져오기 요청이 억제되지 않습니다.

6. 다음 중 하나입니다.

- 큐에 대한 **ProcessName** 로컬 큐 속성이 공백이 아니며, 해당 속성이 식별하는 프로세스 정의 오브젝트가 작성되었습니다. 또는
- 큐에 대한 **ProcessName** 로컬 큐 속성이 모두 공백이지만 큐가 전송 큐입니다. 프로세스 정의가 선택적 이므로 **TriggerData** 속성이 시작될 채널의 이름도 포함할 수 있습니다. 이 경우, 트리거 메시지에는 다음 값을 갖는 속성이 포함됩니다.

- **QName:** 큐 이름
- **ProcessName:** 공백
- **TriggerData:** 트리거 데이터
- **ApplType:** MQAT_UNKNOWN
- **ApplId:** 공백
- **EnvData:** 공백
- **UserData:** 공백

7. 이니시에이션 큐가 작성되었으며 **InitiationQName** 로컬 큐 속성에 지정되었습니다. 참조:

- 이니시에이션 큐에 대하여 가져오기 요청이 억제되지 않습니다(즉, **InhibitGet** 큐 속성 값이 MQQA_GET_ALLOWED임).
- 이니시에이션 큐에 대하여 Put 요청이 억제되어서는 안 됩니다(즉, **InhibitPut** 큐 속성 값이 MQQA_PUT_ALLOWED이어야 함).
- 이니시에이션 큐의 **Usage** 속성 값은 MQUS_NORMAL이어야 합니다.
- 동적 큐가 지원되는 환경에서 이니시에이션 큐는 논리적으로 삭제된 것으로 표시된 동적 큐여서는 안 됩니다.

8. 트리거 모니터가 메시지를 제거하기 위해 현재 이니시에이션 큐를 열었습니다(즉, **OpenInputCount** 로컬 큐 속성이 0보다 큼).
9. 애플리케이션 큐에 대한 트리거 제어(**TriggerControl** 로컬 큐 속성)가 MQTC_ON으로 설정됩니다. 이를 수행하려면 큐를 정의할 때 **trigger** 속성을 설정하거나 ALTER QLOCAL 명령을 사용하십시오.
10. 트리거 유형(**TriggerType** 로컬 큐 속성)이 MQTT_NONE이 아닙니다.

모든 필수 조건을 충족하고 트리거 조건을 야기한 메시지를 작업 단위의 일부로 넣으면, 작업 단위가 커미트되었는지 또는 트리거 유형MQTT_FIRST 또는 MQTT_DEPTH의 경우 백아웃되었는지 여부와 관계없이 작업 단위가 완료될 때까지 트리거 메시지를 트리거 모니터 애플리케이션에서 검색할 수 없습니다.

11. **TriggerType**이 MQTT_FIRST 또는 MQTT_DEPTH이고 큐가 다음과 같은 경우 적당한 메시지가 큐에 배치됩니다.

- 이전에 비어 있지 않았습니까(MQTT_FIRST). 또는
- **TriggerDepth** 또는 추가 메시지(MQTT_DEPTH)가 있었습니다.

다음 조건이 879 페이지의 『2』 충족될 881 페이지의 『10』 경우 (제외 879 페이지의 『3』), MQTT_FIRST의 경우 해당 큐에 대한 마지막 트리거 메시지가 작성된 이후 충분한 간격 (**TriggerInterval** 큐 관리자 속성)이 경과한 경우에 한합니다.

이는 큐에 있는 모든 메시지를 처리하기 전에 큐 서버가 종료되도록 합니다. 트리거 간격의 목적은 생성되는 중복 트리거 메시지 수를 줄이는 것입니다.

참고: 큐 관리자를 중지한 후 재시작하면 **TriggerInterval** 타이머가 재설정됩니다. 두 개의 트리거 메시지를 생성할 수 있는 작은 창이 있습니다. 이 창은 큐의 트리거 속성이 메시지 도착과 동시에 사용 가능하도록 설정되고 큐가 이전에 비어 있지 않았거나(MQTT_FIRST) **TriggerDepth** 또는 추가 메시지(MQTT_DEPTH)가 있었을 때 존재합니다.

12. 큐를 제공하는 애플리케이션만 MQTT_FIRST 또는 MQTT_DEPTH의 **TriggerType**에 대해 MQCLOSE 호출을 발행하므로, 최소한

- 하나의 메시지(MQTT_FIRST) 또는
- **TriggerDepth**(MQTT_DEPTH)

메시지가 충분한 우선순위(조건 879 페이지의 『2』)의 큐에 있고, 조건 880 페이지의 『6』 - 881 페이지의 『10』도 충족됩니다.

이는 MQGET 호출을 발행하는 큐 관리자가 큐가 비어 있는 것을 발견하고 종료되도록 합니다. 그러나 MQGET 및 MQCLOSE 호출 중간에 하나 이상의 메시지가 도착합니다.

참고:

- a. 애플리케이션 큐를 제공하는 프로그램이 모든 메시지를 검색하지 않으면 이로 인해 처리완료된 루프가 발생할 수 있습니다. 프로그램이 큐를 닫을 때마다 큐 관리자는 트리거 모니터가 서버 프로그램을 다시 시작하도록 야기하는 또 하나의 트리거 메시지를 작성합니다.
 - b. 애플리케이션 큐를 제공하는 프로그램이 큐를 닫기 전에 Get 요청을 백아웃하는 경우(또는 프로그램이 이상종료되는 경우)에도 동일한 일이 일어납니다. 그러나 프로그램이 Get 요청을 백아웃하기 전에 큐를 닫고 큐가 비어 있는 경우에는 트리거 메시지가 작성되지 않습니다.
 - c. 이러한 루프가 발생하지 않게 하려면 MQMD의 *BackoutCount* 필드를 사용하여 반복적으로 백아웃되는 메시지를 찾아내십시오. 자세한 정보는 50 페이지의 『백아웃된 메시지』의 내용을 참조하십시오.
13. 다음 조건은 MQSET 또는 명령을 사용하여 충족됩니다.
 - a. • **TriggerControl**이 MQTC_ON으로 변경되었거나,
 - **TriggerControl**이 이미 MQTC_ON이고 **TriggerType**, **TriggerMsgPriority** 또는 **TriggerDepth**(관련된 경우)의 값이 변경되었으며,
 최소한
 - 하나의 메시지(MQTT_FIRST 또는 MQTT_EVERY) 또는
 - **TriggerDepth**(MQTT_DEPTH)

메시지가 충분한 우선순위(조건 879 페이지의 『2』)의 큐에 있고, 조건 880 페이지의 『4』 - 881 페이지의 『10』 (881 페이지의 『8』 제외)도 충족됩니다.

이는 트리거 발생 조건을 이미 충족했을 때 애플리케이션 또는 운영자가 트리거 기준을 변경할 수 있도록 합니다.

- b. 이니시에이션 큐의 **InhibitPut** 큐 속성이 MQQA_PUT_INHIBITED에서 MQQA_PUT_ALLOWED로 변경되고, 최소한

- 하나의 메시지(MQTT_FIRST 또는 MQTT_EVERY) 또는
- **TriggerDepth**(MQTT_DEPTH)

메시지(조건 879 페이지의 『2』에 따라 충분한 우선순위를 가짐)가 이니시에이션 큐에 해당되는 큐에 있고, 조건 880 페이지의 『4』 - 881 페이지의 『10』도 충족됩니다. (조건을 충족하는 각 큐에 대해 하나의 트리거 메시지가 생성됩니다.)

이는 이니시에이션 큐의 MQQA_PUT_INHIBITED 조건 때문에 트리거 메시지가 생성되지 않지만 이 조건이 지금 변경될 수 있도록 합니다.

- c. 애플리케이션 큐의 **InhibitGet** 큐 속성이 MQQA_GET_INHIBITED에서 MQQA_GET_ALLOWED로 변경되고, 최소한

- 하나의 메시지(MQTT_FIRST 또는 MQTT_EVERY) 또는
- **TriggerDepth**(MQTT_DEPTH)

큐에 충분한 우선순위(조건 879 페이지의 『2』)를 가진 메시지가 존재하며, 조건 880 페이지의 『4』부터 881 페이지의 『10』까지도 제외하고 880 페이지의 『5』, 충족된다.

이 경우 애플리케이션 큐에서 메시지를 검색할 수 있을 때만 애플리케이션을 트리거할 수 있습니다.

- d. 트리거 모니터 애플리케이션은 이니시에이션 큐에서 입력하도록 MQOPEN 호출을 발행하며, 최소한

- 하나의 메시지(MQTT_FIRST 또는 MQTT_EVERY) 또는
- **TriggerDepth**(MQTT_DEPTH)

메시지(조건 879 페이지의 『2』에 따라 충분한 우선순위를 가짐)가 이니시에이션 큐에 해당되는 애플리케이션 큐에 있고, 조건 880 페이지의 『4』 - 881 페이지의 『10』 (881 페이지의 『8』 제외)도 충족하며, 입력을 위해 이니시에이션 큐를 연 다른 애플리케이션이 없습니다(조건을 충족하는 각 큐에 대해 하나의 트리거 메시지가 생성됨).

이는 트리거 모니터가 실행하지 않는 동안 메시지가 큐에 도착하고 큐 관리자가 재시작하며 트리거 메시지(비지속적임)가 손실될 수 있게 합니다.

14. MSGDLVSQ가 올바르게 설정됩니다. MSGDLVSQ=FIFO를 설정하면 메시지가 FIFO 기준으로 큐에 전달됩니다. 메시지의 우선순위는 무시되고 큐의 기본 우선순위가 메시지에 지정됩니다. **TriggerMsgPriority**가 큐의 기본 우선순위보다 높은 값으로 설정된 경우, 메시지가 트리거되지 않습니다.

TriggerMsgPriority가 큐의 기본 우선순위보다 낮거나 같은 값으로 설정된 경우, FIRST, EVERY 및 DEPTH 유형에 대해 트리거가 발생합니다. 이러한 유형에 대한 정보는 아래의 필드 **TriggerType** 설명을 883 페이지의 『트리거 이벤트 제어』 참조하십시오.

MSGDLVSQ=PRIORITY로 설정하고 메시지 우선순위가 **TriggerMsgPriority** 필드보다 크거나 같은 경우, 메시지만 트리거 이벤트에 가산됩니다. 이 경우 FIRST, EVERY 및 DEPTH 유형에 대해 트리거가 발생합니다. 예를 들어, **TriggerMsgPriority**보다 낮은 우선순위의 메시지 100개를 넣은 경우 트리거에 유효한 큐 용량은 여전히 0입니다. 그런 다음 큐에 또 하나의 메시지를 넣었지만 이번에도 우선순위가 **TriggerMsgPriority**보다 크거나 같은 경우, 유효 큐 용량은 0에서 1로 증가하고 **TriggerType** FIRST의 조건이 충족됩니다.

참고:

1. 881 페이지의 『12』 단계(여기서 트리거 메시지가 애플리케이션 큐에 도착하는 메시지 이외의 다른 이벤트의 결과로 생성됨)부터는 트리거 메시지를 작업 단위의 일부로 넣지 못합니다. 또한 **TriggerType**이 MQTT_EVERY이고 애플리케이션 큐에 하나 이상의 메시지가 있는 경우, 하나의 트리거 메시지만 생성됩니다.
2. IBM MQ가 MQPUT 동안 메시지를 세그먼트화하면, 모든 세그먼트가 큐에 배치될 때까지 트리거 이벤트가 처리되지 않습니다. 그러나 메시지 세그먼트가 큐에 있으면, IBM MQ는 해당 세그먼트를 트리거하기 위한 개별

메시지로 처리합니다. 예를 들어, 단일 논리 메시지가 세 조각으로 분할된 경우 첫 번째 MQPUT이 수행되고 세그먼트될 때 하나의 트리거 이벤트만 처리됩니다. 그러나 세 개의 세그먼트는 각각 고유 트리거 이벤트가 IBM MQ 네트워크를 통해 이동할 때 처리되게 합니다.

3.  공유 큐가 트리거링을 위해 설정된 경우 IBM MQ for z/OS, 해당 공유 큐를 호스팅하는 커플링 시설(Coupling Facility)과의 연결이 끊어지면 트리거 이벤트가 생성되고 메시지가 시작 큐(initiation queue)에 배치될 수 있습니다. 원래 공유 큐 설정에 트리거용 메시지가 전혀 등록되지 않았을 때에도 이런 현상이 발생할 수 있습니다. 이는 [목록 알림 벡터\(The List Notification Vector\)](#)에 문서화된 대로 IXLVECTR 매크로에 의한 비트 과다 지정에 기인합니다.

트리거 이벤트 제어

애플리케이션 큐를 정의하는 일부 속성을 사용하여 트리거 이벤트를 제어합니다. 또한 이 정보는 트리거 유형(EVERY, FIRST 및 DEPTH) 사용의 예를 제공합니다.

트리거를 사용 또는 사용 안함으로 설정할 수 있으며, 트리거 이벤트에 포함되는 메시지의 수 또는 우선순위를 선택할 수 있습니다. [오브젝트 속성](#)에 이러한 속성에 대한 자세한 설명이 있습니다.

관련 속성은 다음과 같습니다.

TriggerControl

이 속성을 사용하여 애플리케이션 큐에 대해 트리거를 사용 또는 사용 안함으로 설정합니다.

TriggerMsgPriority

트리거 이벤트에 포함되기 위해 메시지가 가져야 하는 최소한의 우선순위입니다. *TriggerMsgPriority* 보다 낮은 우선순위의 메시지가 애플리케이션 큐에 도착하면 큐 관리자는 트리거 메시지를 작성할지 여부를 판별할 때 해당 메시지를 무시합니다. *TriggerMsgPriority*가 0으로 설정되면 모든 메시지가 트리거 이벤트에 포함됩니다.

TriggerType

트리거 유형 NONE(*TriggerControl* OFF 설정과 마찬가지로 트리거를 사용 안함 설정) 외에도, 다음 트리거 유형을 사용하여 큐의 민감도를 트리거 이벤트로 설정할 수 있습니다.

EVERY

메시지가 애플리케이션 큐에 도착할 때마다 트리거 이벤트가 발생합니다. 애플리케이션의 다중 인스턴스를 시작하려면 이런 유형의 트리거를 사용하십시오.

FIRST

트리거 이벤트는 애플리케이션 큐에 있는 메시지 수가 0에서 1로 변경될 때에만 발생합니다. 첫 번째 메시지가 큐에 도착할 때 제공 프로그램을 시작해서 처리할 메시지가 더 이상 없을 때까지 계속한 후 종료하려면 이런 유형의 트리거를 사용하십시오. 언제나 큐가 비워질 때까지 큐를 처리해야 합니다. [884 페이지의 『트리거 유형 FIRST의 특수 사례』](#)도 참조하십시오.

DEPTH

트리거 이벤트는 애플리케이션 큐에 있는 메시지 수가 **TriggerDepth** 속성 값에 도달할 때에만 발생합니다. 이런 유형의 트리거는 일반적으로 요청 세트에 대한 모든 응답을 수신할 때 프로그램을 시작하는데 사용됩니다.

용량별 트리거: 용량별 트리거를 사용하는 경우 큐 관리자는 트리거 메시지를 작성한 후에 트리거를 사용 안함으로 설정(*TriggerControl* 속성 사용)해야 합니다. 애플리케이션은 이러한 상황이 발생한 후에 트리거 자체를 다시 사용으로 설정(MQSET 호출 사용)해야 합니다.

트리거 사용 안함 설정 조치는 동기점 제어를 받지 않기 때문에 작업 단위 백아웃을 통해 트리거를 다시 사용으로 설정할 수 없습니다. 프로그램이 트리거 이벤트를 발생시킨 Put 요청을 백아웃하거나 프로그램이 이상종료되면 MQSET 호출 또는 ALTER QLOCAL 명령을 사용하여 트리거를 다시 사용으로 설정해야 합니다.

TriggerDepth

용량별 트리거를 사용할 때 트리거 이벤트를 발생시키는 큐의 메시지 수입니다.

큐 관리자가 트리거 메시지를 작성하기 위해 충족해야 할 조건은 [879 페이지의 『트리거 이벤트의 조건』](#)에서 설명합니다.

트리거 유형 EVERY 사용의 예

자동차 보험에 대한 요청을 생성하는 애플리케이션을 고려하십시오. 이 애플리케이션은 여러 보험 회사에 요청 메시지를 보내고 매번 동일한 응답 대상 큐를 지정합니다. 이 응답 대상 큐에 EVERY 유형의 트리거를 설정하면 응답이 도착할 때마다 응답이 서버의 인스턴스를 트리거하여 응답을 처리할 수 있습니다.

트리거 유형 FIRST 사용의 예

매일 비즈니스의 세부사항을 본사로 각각 전송하는 여러 지방 사무소를 둔 조직을 고려하십시오. 지방 사무소에서는 모두 근무일의 마지막에 이 작업을 동시에 수행하고, 본사에는 모든 지방 사무소에서 받은 세부사항을 처리하는 애플리케이션이 있습니다. 본사에 도착하는 첫 번째 메시지에 의해 이 애플리케이션을 시작하는 트리거 이벤트가 발생할 수 있습니다. 이 애플리케이션은 해당 큐에 메시지가 더 이상 없을 때까지 처리를 계속합니다.

트리거 유형 DEPTH 사용의 예

항공편 예약 확인, 호텔 객실 예약 확인, 자동차 렌트 및 여행자 수표 발행에 대한 단일 요청을 작성하는 여행사 애플리케이션을 고려하십시오. 이 애플리케이션은 이러한 항목을 4개의 요청 메시지로 분리하고 각각을 별도의 목적지로 보냅니다. 용량 값이 4로 설정된 응답 대상 큐에 DEPTH 유형의 트리거를 설정하면 4개의 응답이 모두 도착했을 때에만 애플리케이션이 재시작됩니다.

다른 요청에 의해 발행되었을 수 있는 또 하나의 메시지가 4개 응답 중 마지막 응답 이전에 응답 대상 큐에 도착하면 요청 애플리케이션이 미리 트리거됩니다. 이를 방지하려면 DEPTH 트리거를 사용하여 요청에 대한 다중 응답을 수집할 때는 항상 각 요청에 대해 새 응답 대상 큐를 사용하십시오.

트리거 유형 FIRST의 특수 사례

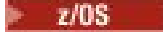
트리거 유형 FIRST를 사용하면 또 하나의 메시지가 도착했을 때 애플리케이션 큐에 이미 메시지가 있는 경우 큐 관리자는 일반적으로 다른 트리거 메시지를 작성하지 않습니다.

하지만 큐를 제공하는 애플리케이션이 실제로 큐를 열지 못할 수도 있습니다(예를 들어, 애플리케이션이 시스템 문제점으로 인해 종료될 수 있음). 올바르게 않은 애플리케이션 이름을 프로세스 정의 오브젝트에 넣으면 큐를 제공하는 애플리케이션이 메시지를 선택하지 않습니다. 이러한 상황에서 다른 메시지가 애플리케이션 큐에 도착하면 이 메시지(및 큐의 다른 메시지)를 처리할 서버가 실행되지 않습니다.

이를 해결하기 위해 큐 관리자는 다음 환경에서 추가 트리거 메시지를 작성합니다.

- 다른 메시지가 애플리케이션 큐에 도착하지만 큐 관리자가 해당 큐에 대한 마지막 트리거 메시지를 작성한 이후로 사전정의된 시간 간격이 경과하는 경우입니다. 이 시간 간격은 큐 관리자 속성 *TriggerInterval*에 정의됩니다. 기본값은 999 999 999밀리초입니다.
-  IBM MQ for z/OS에서 열린 이니시에이션 큐를 이름 지정하는 애플리케이션 큐는 주기적으로 스캔됩니다. 마지막 트리거 메시지가 송신된 이후로 *TRIGINT* 밀리초가 지났고 큐가 트리거 이벤트에 대한 조건을 충족하고 *CURDEPTH*가 0보다 큰 경우 트리거 메시지가 생성됩니다. 이 프로세스를 백스탑 트리거라고 합니다.

애플리케이션에서 사용할 트리거 간격의 값을 결정할 때 다음 사항을 고려하십시오.

- *TriggerInterval*이 낮은 값으로 설정되고 애플리케이션 큐를 제공하는 애플리케이션이 없는 경우 트리거 유형 FIRST가 트리거 유형 EVERY 같이 작동할 수 있습니다. 이는 애플리케이션 큐에 넣을 메시지의 비율에 따라 다르며, 결국 다른 시스템 활동에 따라 달라질 수 있습니다. 트리거 간격이 매우 작으면 트리거 유형이 EVERY가 아니고 FIRST일지라도 메시지가 애플리케이션 큐에 넣어질 때마다 또 다른 트리거 메시지가 생성되기 때문입니다. (트리거 간격이 0인 트리거 유형 FIRST는 트리거 유형 EVERY와 같습니다.)
-  IBM MQ for z/OS에서 *TRIGINT*를 낮은 값으로 설정하고 트리거 유형 FIRST 애플리케이션 큐를 제공하는 애플리케이션이 없는 경우 열린 이니시에이션 큐 이름을 지정하는 애플리케이션 큐의 주기적 스캔이 발생할 때마다 백스탑 트리거가 트리거 메시지를 생성합니다.
- 작업 단위가 백아웃되고(트리거 메시지 및 작업 단위) 트리거 간격이 높은 값(또는 기본값)으로 설정된 경우, 작업 단위가 백아웃될 때 하나의 트리거 메시지가 생성됩니다. 하지만 트리거 간격이 낮은 값 또는 0(트리거 유형 FIRST가 트리거 유형 EVERY 같이 작동하게 함)으로 설정된 경우 여러 트리거 메시지가 생성될 수 있습니다. 작업 단위가 백아웃되면 모든 트리거 메시지를 계속 사용할 수 있게 됩니다. 생성되는 트리거 메시지 수는 트리거 간격에 따라 달라집니다. 트리거 간격을 0으로 설정하면 최대 수의 메시지가 생성됩니다.

트리거된 큐를 사용하는 애플리케이션 설계

애플리케이션에 대한 설정과 제어 및 트리거하는 방법은 이미 살펴보았습니다. 다음은 애플리케이션을 설계할 때 고려해야 할 몇 가지 팁입니다.

트리거 메시지 및 작업 단위

작업 단위의 일부가 아닌 트리거 이벤트 때문에 작업 단위 외부에서 다른 메시지에 상관없이 작성된 트리거 메시지는 이니시에이션 큐에 넣는 즉시 트리거 모니터에서 검색할 수 있습니다.

작업 단위의 일부인 트리거 이벤트 때문에 작성된 트리거 메시지가 작업 단위가 커밋되든 백아웃되든 상관없이 UOW가 해석될 때 이니시에이션 큐에서 사용 가능하게 됩니다.

큐 관리자가 트리거 메시지를 이니시에이션 큐에 넣는 데 실패하면 데드-레터(미전달 메시지) 큐에 넣어집니다.

참고:

1. 큐 관리자는 트리거 이벤트에 대한 조건이 존재하는지 여부를 평가할 때 커밋된 메시지와 커밋되지 않은 메시지를 모두 계수합니다.

트리거 유형이 **FIRST** 또는 **DEPTH**인 경우, 작업 단위가 백아웃되더라도 트리거 메시지를 사용 가능하게 하여 필수 조건이 충족될 때 트리거 메시지를 항상 사용할 수 있습니다. 예를 들어, 트리거 유형 **FIRST**로 트리거되는 큐에 대해 작업 단위 내에서 Put 요청을 고려하십시오. 이 조건이 발생하면 큐 관리자가 트리거 메시지를 작성합니다. 다른 작업 단위에서 다른 Put 요청이 발생한 경우, 애플리케이션 큐의 메시지 수가 이제 1에서 2로 변경되었기 때문에 트리거 이벤트에 대한 조건이 충족되지 않으므로 다른 트리거 이벤트를 일으키지 않습니다. 이제 첫 번째 작업 단위가 백아웃되었지만 두 번째가 커밋된 경우 트리거 메시지가 계속 작성됩니다.

그러나 이것은 트리거 이벤트에 대한 조건이 충족되지 않았을 때 트리거 메시지가 때때로 작성되는 것을 의미합니다. 트리거를 사용하는 애플리케이션은 항상 이 상황을 핸들링하기 위한 준비가 되어 있어야 합니다.

*WaitInterval*을 적당한 값으로 설정하고 MQGET 호출에서 대기 옵션을 사용할 것을 권장합니다.

작성된 트리거 메시지는 작업 단위가 백아웃되든 커밋되든 상관없이 항상 사용 가능하게 됩니다.

2. 로컬 공유 큐의 경우(즉, 공유 큐가 큐 공유 그룹에 있음) 큐 관리자가 커밋된 메시지만 계수합니다.

트리거된 큐에서 메시지 가져오기

트리거를 사용하는 애플리케이션을 설계할 때, 트리거 모니터가 프로그램을 시작하는 것과 다른 메시지가 애플리케이션 큐에서 사용 가능하게 되는 것 사이에 지연이 생길 수 있습니다. 이는 트리거 이벤트를 일으키는 메시지가 다른 메시지보다 먼저 커밋될 때 발생할 수 있습니다.

메시지가 도착하는 시간을 허용하려면 트리거 조건이 설정된 큐에서 메시지를 제거하기 위해 MQGET 호출을 사용할 때 항상 대기 옵션을 사용하십시오. *WaitInterval*은 메시지를 넣은 시간과 해당 Put 호출이 커밋된 시간 사이의 타당한 최장 시간을 충분히 허용해야 합니다. 리모트 큐 관리자에서 메시지가 도착할 경우 이 시간은 다음의 영향을 받습니다.

- 커밋되기 전에 넣어진 메시지 수
- 통신 링크의 속도 및 가용성
- 메시지의 크기

대기 옵션을 포함하여 MQGET 호출을 사용해야 하는 상황의 예로, 작업 단위를 설명할 때 사용한 것과 동일한 예를 고려하십시오. 이것은 트리거 유형 **FIRST**로 트리거된 큐에 대한 작업 단위 내의 Put 요청이었습니다. 이 이벤트가 발생하면 트리거 관리자가 트리거 메시지를 작성합니다. 다른 작업 단위에서 다른 Put 요청이 발생한 경우, 애플리케이션 큐의 메시지 수가 0에서 1로 변경되지 않았기 때문에 다른 트리거 이벤트를 일으키지 않습니다. 이제 첫 번째 작업 단위가 백아웃되었지만 두 번째가 커밋된 경우 트리거 메시지가 계속 작성됩니다. 따라서 첫 번째 작업 단위가 백아웃되는 시점에 트리거 메시지가 작성됩니다. 두 번째 메시지가 커밋되기 전에 상당한 시간이 지연되는 경우 트리거된 애플리케이션이 대기해야 할 수도 있습니다.

트리거 유형이 **DEPTH**인 경우, 관련된 모든 메시지가 마침내 커밋되더라도 지연이 발생할 수 있습니다.

TriggerDepth 큐 속성의 값이 2라고 가정하십시오. 두 개의 메시지가 큐에 도착할 때 두 번째 메시지로 인해 트리거 메시지가 작성됩니다. 그러나 두 번째 메시지가 먼저 커밋되면 그 때 트리거 메시지가 사용 가능하게 됩니다. 트리거 모니터는 서버 프로그램을 시작하지만 프로그램은 첫 번째 메시지가 커밋될 때까지 두 번째 메시지만 검색할 수 있습니다. 따라서 프로그램은 첫 번째 메시지가 사용 가능하게 될 때까지 대기해야 할 수도 있습니다.

대기 간격이 만료될 때 검색할 수 있는 메시지가 없으면 종료하도록 애플리케이션을 설계하십시오. 하나 이상의 메시지가 나중에 도착하는 경우 애플리케이션을 다시 트리거하여 해당 메시지를 처리하십시오. 이 방법은 애플리케이션이 유휴 상태가 되는 것과 불필요하게 자원을 사용하는 것을 방지합니다.

트리거 모니터에 의한 이니시에이션 큐 처리

큐 관리자에 대한 트리거 모니터는 큐를 제공하는 다른 애플리케이션과 유사합니다. 하지만 트리거 모니터는 이니시에이션 큐를 제공합니다.

트리거 모니터는 대개 지속적으로 실행되는 프로그램입니다. 트리거 메시지가 이니시에이션 큐에 도착하면 트리거 모니터가 해당 메시지를 검색합니다. 트리거 모니터는 메시지의 정보를 사용하여 애플리케이션 큐의 메시지를 처리하는 애플리케이션 시작 명령을 실행합니다.

트리거 모니터는 프로그램이 올바른 애플리케이션 큐에서 올바른 조치를 수행할 수 있도록, 시작할 프로그램에 충분한 정보를 전달해야 합니다.

채널 시작기는 메시지 채널 에이전트에 대한 특수 유형의 트리거 모니터 예입니다. 하지만 이 상황에서는 트리거 유형 FIRST 또는 DEPTH를 사용해야 합니다.

ALW AIX, Linux, and Windows 시스템의 트리거 모니터

이 주제에는 AIX, Linux, and Windows 시스템에서 제공된 트리거 모니터에 대한 정보가 들어 있습니다.

다음 트리거 모니터가 서버 환경에 제공됩니다.

amqstrg0

이는 **runmqtrm**에서 제공하는 함수 서브세트를 제공하는 샘플 트리거 모니터입니다. **amqstrg0**에 대한 자세한 정보는 [971 페이지의 『멀티플랫폼용 샘플 절차적 프로그램 사용』](#)의 내용을 참조하십시오.

runmqtrm

이 명령의 구문은 **runmqtrm**['-m QMgrName']['-q InitQ']이며, 여기서 QMgrName 큐 관리자이고 InitQ 시작 큐입니다. 기본 큐는 기본 큐 관리자의 SYSTEM.DEFAULT.INITIATION.QUEUE입니다. 이는 해당 트리거 메시지에 대한 프로그램을 호출합니다. 이 트리거 모니터는 기본 애플리케이션 유형을 지원합니다.

트리거 모니터에서 운영 체제로 전달된 명령 문자열은 다음과 같이 빌드됩니다.

1. 관련 PROCESS 정의의 *ApplId*(작성된 경우)
2. 큰따옴표로 묶인 MQTMC2 구조
3. 관련 PROCESS 정의의 *EnvData*(작성된 경우)

여기서 *ApplId*는 명령행에 입력된 바와 같이 실행할 프로그램의 이름입니다.

전달된 매개변수는 MQTMC2 문자 구조입니다. 명령 문자열은 시스템 명령이 하나의 매개변수로 승인할 수 있도록 하기 위해 제공된 그대로 큰따옴표로 묶여 호출됩니다.

트리거 모니터는 방금 시작한 애플리케이션이 완료될 때까지 이니시에이션 큐에 다른 메시지가 있는지 확인해보지 않습니다. 애플리케이션이 수행해야 할 처리가 많을 경우, 트리거 모니터는 도착하는 트리거 메시지 수를 유지하지 못할 수도 있습니다. 다음 두 가지 옵션이 있습니다.

- 추가 트리거 모니터 실행
- 백그라운드에서 시작된 애플리케이션 실행

보다 많은 트리거 모니터를 실행하도록 선택하면 어느 때든지 실행할 수 있는 최대 애플리케이션 수를 제어할 수 있습니다. 백그라운드에서 애플리케이션을 실행하는 경우, IBM MQ에서 실행할 애플리케이션 수에 부과하는 제한은 없습니다.

 **Linux**  **AIX** 시작된 애플리케이션을 'AIX and Linux의 백그라운드에서 실행하려면 PROCESS 정의의 '*EnvData*' 끝에 &를 넣습니다.

Windows의 백그라운드에서 시작된 애플리케이션을 실행하려면 *ApplId* 필드 내에서 애플리케이션의 이름 앞에 START 명령을 붙이십시오. 예:

```
START ?B AMQSECHA
```


참고:  Windows 경로에 공백이 경로 이름의 일부로 포함된 경우, 이 공백을 따옴표(")로 묶어서 해당 경로가 단일 인수로 핸들링되도록 해야 합니다. 예를 들어, "C:\Program Files\Application Directory\Application.exe"입니다.

다음은 파일 이름에 공백이 경로의 일부로 포함된 APPLICID 문자열의 예입니다.

```
START "" /B "C:\Program Files\Application Directory\Application.exe"
```

예제에서 Windows START 명령의 구문은 큰따옴표로 묶인 빈 문자열을 포함합니다. START는 따옴표로 묶인 첫 번째 인수가 새 명령의 제목으로 처리되도록 지정합니다. Windows가 애플리케이션 경로를 'title' 인수로 오인하지 않도록 하려면 큰따옴표로 묶인 제목 문자열을 애플리케이션 이름 앞에 있는 명령에 추가하십시오.

다음 트리거 모니터가 IBM MQ 클라이언트에 제공됩니다.

runmqtmcc

이는 IBM MQ MQI client 라이브러리와 링크되는 점을 제외하고는 runmqtrm과 같습니다.

ALW CICS에 대한 트리거 모니터

CICS에 대해 amqltmcc0 트리거 모니터가 제공됩니다. 이 트리거 모니터는 표준 트리거 모니터 runmqtrm과 같은 방식으로 작동하지만, 다른 방식으로 실행하면 CICS 트랜잭션을 트리거합니다.

이 항목은 AIX and Linux x86-64 시스템에만 적용됩니다 Windows.

트리거 모니터는 CICS 프로그램으로 제공됩니다. 이를 4문자 트랜잭션 이름으로 정의하십시오. 트리거 모니터를 시작하려면 4문자 이름을 입력하십시오. 트리거 모니터는 기본 큐 관리자(qm.ini 파일 또는 IBM MQ for Windows의 레지스트리에 이름 지정된 대로) 및 SYSTEM.CICS.INITIATION.QUEUE를 사용합니다.

다른 큐 관리자 또는 큐를 사용하려면 트리거 모니터 MQTMC2 구조를 빌드하십시오. 이 경우, 구조가 너무 길어서 매개변수로서 추가할 수 없기 때문에 EXEC CICS START 호출을 사용하여 프로그램을 작성해야 합니다. 그런 다음, MQTMC2 구조를 트리거 모니터의 START 요청에 데이터로 전달하십시오.

MQTMC2 구조를 사용하는 경우 트리거 모니터에는 *StrucId*, *Version*, *QName* 및 **QMGrName** 매개변수만 제공해야 합니다. 기타 필드는 트리거 모니터가 참조하지 않습니다.

이니시에이션 큐에서 읽은 메시지는 트리거 메시지의 APPL_TYPE이 MQAT_CICS라고 가정하여 EXEC CICS START를 통해 CICS 트랜잭션을 시작하는 데 사용됩니다. 이니시에이션 큐에서 메시지 읽기는 CICS 동기점 제어 하에 수행됩니다.

메시지는 모니터가 시작되고 중지될 때와 오류가 발생할 때에 생성됩니다. 이러한 메시지는 CSMT 임시 데이터 큐로 송신됩니다.

표 144. 사용 가능한 트리거 모니터의 버전.	
2개의 열이 있는 표입니다. 첫 번째 열은 트리거 모니터에 대한 사용 가능한 버전을 나열하고 두 번째 열은 각 버전이 사용되고 있는 플랫폼에 대해 표시합니다.	
버전	다음을 사용하십시오.
amqltmcc0	TXSeries의 경우:  AIX  Linux x86-64 시스템
amqltmcc4	 TXSeries ~을 위해 Windows 5.1
amqltmcc	CICS 트리거 모니터의 클라이언트 바인드 버전
 V9.4.0  V9.4.0 amqltmcc064	x86-64Linux 용 64비트 TXSeries

표 144. 사용 가능한 트리거 모니터의 버전.

2개의 열이 있는 표입니다. 첫 번째 열은 트리거 모니터에 대한 사용 가능한 버전을 나열하고 두 번째 열은 각 버전이 사용되고 있는 플랫폼에 대해 표시합니다.

(계속)

버전	다음을 사용하십시오.
 V9.4.0  V9.4.0 amqltmcc64	amqltmc064 클라이언트 버전

다른 환경에 대한 트리거 모니터가 필요한 경우, 큐 관리자가 이니시에이션 큐에 넣는 트리거 메시지를 처리할 수 있는 프로그램을 작성하십시오. 해당 프로그램은 다음 조치를 수행해야 합니다.

1. MQGET 호출을 사용하여 메시지가 이니시에이션 큐에 도착할 때까지 대기하십시오.
2. 트리거 메시지의 MQTM 구조에 대한 필드를 조사하여 시작할 애플리케이션의 이름과 애플리케이션이 실행되는 환경을 찾으십시오.
3. 환경 특성의 시작 명령을 실행하십시오.

 예를 들어, z/OS 배치에서, 작업을 내부 리더에 제출하십시오.

4. 필요한 경우 MQTM 구조를 MQTMC2 구조로 변환하십시오.
5. 시작된 애플리케이션에 MQTMC2 또는 MQTM 구조를 전달하십시오. 이 구조는 사용자 데이터를 포함할 수 있습니다.
6. 해당 큐를 제공하는 애플리케이션을 애플리케이션 큐와 연관시키십시오. 큐의 **ProcessName** 속성에 프로세스 정의 오브젝트(작성된 경우) 이름을 지정하여 이 작업을 수행합니다. 프로세스 정의 객체에 이름을 지정하려면 또는 **ALTER QLOCAL DEFINE QLOCAL** 명령을 사용할 수 있습니다.

 IBM i에서는 CRTMQMQ 또는 CHGMQMQ를 사용할 수 있습니다.

트리거 모니터 인터페이스에 대한 자세한 정보는 [MQTMC2](#)를 참조하십시오.

 IBM i에서의 트리거 모니터

IBM i에서는 **runmqtrm** 제어 명령 대신 IBM MQ for IBM i CL 명령인 **STRMQMTRM**을 사용하십시오.

다음과 같이 STRMQMTRM 명령을 사용하십시오.

```
STRMQMTRM INITQNAME(InitQ) MQMNAME(QMgrName)
```

세부사항은 runmqtrm과 같습니다.

다음 샘플 프로그램도 제공되며, 이는 고유 트리거 모니터를 작성하는 데 모델로 사용할 수 있습니다.

AMQSTRG4

시작되는 프로세스에 대한 IBM i 작업을 제출하는 트리거 모니터이지만, 이는 각 트리거 메시지와 연관된 추가적인 처리가 있음을 의미합니다.

AMQSERV4

트리거 서버입니다. 각 트리거 메시지에 대해 이 서버는 자체 작업의 프로세스에 대한 명령을 실행하고 CICS 트랜잭션을 호출할 수 있습니다.

트리거 모니터와 트리거 서버 모두 시작한 프로그램에 MQTMC2 구조를 전달합니다. 이 구조에 대한 설명은 [MQTMC2](#)를 참조하십시오. 이 두 샘플 모두 소스 및 실행 파일 양식으로 전달됩니다.

이러한 트리거 모니터는 고유 IBM i 프로그램을 호출할 수 있기 때문에, Java 클래스가 IFS에 있다고 해서 직접적으로 Java 프로그램을 트리거할 수는 없습니다. 하지만 Java 프로그램을 호출하고 TMC2 구조를 통해 전달하는 CL 프로그램을 트리거하여 간접적으로 Java 프로그램을 트리거할 수 있습니다. TMC2 구조의 최소 크기는 732바이트입니다.

다음은 샘플 CLP의 소스입니다.

```
PGM PARM(&TMC2)
DCL &TMC2 *CHAR LEN(800)
```

```
ADDENVVAR ENVVAR(TM) VALUE(&TMC2)
QSH CMD('java_pgmname $TM')
RMVENVVAR ENVVAR(TM)
ENDPGM
```

다음 트리거 모니터 프로그램이 IBM MQ MQI client에 제공됩니다. RUNMQTMC

다음과 같이 RUNMQTMC를 호출하십시오.

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QMgrName '-q' InitQ)
```

▶ z/OS 트리거 모니터를 위한 z/OS

이 항목에는 에서 제공되는 트리거 모니터에 대한 z/OS 정보가 포함되어 있습니다.

IBM MQ for z/OS 다음과 같은 트리거 모니터를 제공합니다:

IMS 환경에서 사용되는 CSQQTRMN

배치 지향 IMS 프로그램을 설정하여 시작 IBM MQ 대기열을 모니터링할 수 있습니다. 자세한 내용은 [트리거 IMS 모니터 설정을 참조하십시오](#).

CICS 환경에서 사용되는 CKTI

환경에서 CICS CKTI 인스턴스를 설정하여 시작 대기열을 모니터링하고 도착하는 트리거 메시지를 해당 대기열에서 가져올 수 있습니다. 자세한 내용은 [작업 시작자 또는 트리거 모니터\(CKTI\)를 참조하십시오](#).

트리거 메시지의 특성

다음 주제에서는 트리거 메시지의 몇 가지 다른 특성에 대해 설명합니다.

- [889 페이지의 『트리거 메시지의 우선순위 및 지속성』](#)
- [889 페이지의 『큐 관리자 재시작 및 트리거 메시지』](#)
- [889 페이지의 『트리거 메시지와 오브젝트 속성 변경』](#)
- [889 페이지의 『트리거 메시지의 형식』](#)

트리거 메시지의 우선순위 및 지속성

트리거 메시지는 지속적이어야 할 필요가 없기 때문에 지속적이지 않습니다.

그러나 트리거 이벤트를 생성하기 위한 조건은 지속되므로, 이러한 조건이 충족될 때마다 트리거 메시지가 생성됩니다. 트리거 메시지가 손실되면, 애플리케이션 큐에 애플리케이션 메시지가 계속 존재하는 경우, 모든 조건이 충족되는 즉시 큐 관리자가 트리거 메시지를 생성하도록 보장합니다.

작업의 단위가 롤백되면 생성된 모든 트리거 메시지도 항상 전달됩니다.

트리거 메시지는 이니시에이션 큐의 기본 우선순위를 가집니다.

큐 관리자 재시작 및 트리거 메시지

큐 관리자 재시작 후 이니시에이션 큐가 입력을 위해 다음에 열릴 때, 연관된 애플리케이션 큐에 관련 메시지가 있고 트리거에 대해 정의된 경우 트리거 메시지를 이 이니시에이션 큐에 넣을 수 있습니다.

트리거 메시지와 오브젝트 속성 변경

트리거 메시지는 트리거 이벤트 시점에 강제 실행 중인 트리거 속성의 값에 따라 작성됩니다.

(작업 단위 안에 트리거 메시지 생성을 야기한 메시지를 넣었기 때문에) 그 이후까지 트리거 모니터에서 트리거 메시지 사용이 불가능하면 그 동안은 트리거 속성을 변경해도 트리거 메시지에 영향을 미치지 않습니다. 특히, 트리거를 사용 안함으로 설정해도 트리거 메시지는 한 번 작성되면 사용 불가능하게 되지 않습니다. 또한 트리거 메시지를 사용할 수 있게 되었을 때 애플리케이션 큐가 더 이상 존재하지 않을 수도 있습니다.

트리거 메시지의 형식

트리거 메시지의 형식은 MQTM 구조에 의해 정의됩니다.

이 형식은 다음과 같은 필드가 있습니다. 큐 관리자는 애플리케이션 큐 및 해당 큐와 연관된 프로세스의 오브젝트 정의에 있는 정보를 사용하여 트리거 메시지를 작성할 때 이 필드를 채웁니다.

StrucId

구조 ID입니다.

Version

구조의 버전입니다.

QName

트리거 이벤트가 발생한 애플리케이션 큐의 이름입니다. 큐 관리자는 트리거 메시지를 작성할 때 애플리케이션 큐의 **QName** 속성을 사용하여 이 필드를 채웁니다.

ProcessName

애플리케이션 큐와 연관된 프로세스 정의 오브젝트의 이름입니다. 큐 관리자는 트리거 메시지를 작성할 때 애플리케이션 큐의 **ProcessName** 속성을 사용하여 이 필드를 채웁니다.

TriggerData

트리거 모니터에서 사용되는 자유 형식 필드입니다. 큐 관리자는 트리거 메시지를 작성할 때 애플리케이션 큐의 **TriggerData** 속성을 사용하여 이 필드를 채웁니다. 이 IBM MQ for Multiplatforms 필드는 트리거될 채널의 이름을 지정하는 데 사용할 수 있습니다.

ApplType

트리거 모니터가 시작하는 애플리케이션의 유형입니다. 큐 관리자가 트리거 메시지를 생성할 때, 이 필드는 에서 식별된 프로세스 정의 객체의 **ProcessName** 속성을 **ApplType** 사용하여 채워집니다.

ApplId

트리거 모니터가 시작하는 애플리케이션을 식별하는 문자열입니다. 큐 관리자가 트리거 메시지를 생성할 때, 이 필드는 에서 식별된 프로세스 정의 객체의 **ProcessName** 속성을 **ApplId** 사용하여 채워집니다.

CICS이(가) 제공하는 트리거 모니터 CKTI를 사용하는 경우 프로세스 정의 오브젝트의 **ApplId** 속성은 CICS 트랜잭션 ID입니다.

 IBM MQ for z/OS이(가) 제공하는 CSQQTRMN을 사용하는 경우 프로세스 정의 오브젝트의 **ApplId** 속성은 IMS 트랜잭션 ID입니다.

EnvData

트리거 모니터에서 사용되는 환경 관련 데이터를 포함하는 문자 필드입니다. 큐 관리자가 트리거 메시지를 생성할 때, 이 필드는 에서 식별된 프로세스 정의 객체의 **ProcessName** 속성을 **EnvData** 사용하여 채워집니다. 제공된 트리거 모니터 CICS(CKTI)는 이 또는 제공된 IBM MQ for z/OS 트리거 모니터(CSQQTRMN) 필드를 사용하지 않지만, 다른 트리거 모니터는 이를 사용할 수 있습니다.

UserData

트리거 모니터에서 사용되는 사용자 데이터를 포함하는 문자 필드입니다. 큐 관리자가 트리거 메시지를 생성할 때, 이 필드는 에서 식별된 프로세스 정의 객체의 **ProcessName** 속성을 **UserData** 사용하여 채워집니다. 이 필드는 트리거할 채널의 이름을 지정할 때 사용할 수 있습니다.

[MQTM](#)에 트리거 메시지 구조에 대한 전체 설명이 있습니다.

트리거가 작동하지 않는 경우

트리거 모니터가 프로그램을 시작할 수 없거나 큐 관리자가 트리거 메시지를 전달할 수 없는 경우 프로그램이 트리거되지 않습니다. 예를 들어, 프로세스 오브젝트의 **applid**는 프로그램이 백그라운드에서 시작되도록 지정해야 합니다. 그렇지 않으면, 트리거 모니터가 프로그램을 시작할 수 없습니다.

트리거 메시지를 작성했지만 이니시에이션 큐에 넣을 수 없는 경우(예를 들어, 큐가 가득 찼거나 트리거 메시지 길이가 이니시에이션 큐에 대해 지정된 최대 메시지 길이보다 크기 때문)에는 해당 트리거 메시지를 데드-레터(미전달 메시지) 큐에 대신 넣습니다.

데드-레터 큐에 대한 넣기 조작을 완료할 수 없는 경우 트리거 메시지가 제거되고 경고 메시지가  z/OS 콘솔 또는 시스템 운영자에게 송신되거나 오류 로그에 넣기됩니다.

데드-레터 큐에 트리거 메시지를 넣으면 해당 큐에 대한 트리거 메시지가 생성될 수 있습니다. 데드-레터 큐에 메시지를 추가하면 이 두 번째 트리거 메시지는 제거됩니다.

프로그램이 성공적으로 트리거되었지만 큐에서 메시지를 수신하기 전에 이상종료되는 경우, 추적 유틸리티(예: 프로그램이 CICS에서 실행 중인 경우 CICS AUXTRACE)를 사용하여 실패의 원인을 찾으십시오.

MQI 및 클러스터에 대한 작업

클러스터와 관련된 호출 및 리턴 코드에 대한 특수 옵션이 있습니다.

다음 링크를 사용하면 클러스터에서 사용할 호출 및 리턴 코드에 사용 가능한 옵션에 대해 자세히 알아볼 수 있습니다.

- [891 페이지의 『MQOPEN 및 클러스터』](#)
- [893 페이지의 『MQPUT, MQPUT1 및 클러스터』](#)
- [893 페이지의 『MQINQ 및 클러스터』](#)
- [894 페이지의 『MQSET 및 클러스터』](#)
- [894 페이지의 『리턴 코드』](#)

관련 개념

[744 페이지의 『MQI\(Message Queue Interface\) 개요』](#)
MQI(Message Queue Interface) 컴포넌트에 대해 학습합니다.

[756 페이지의 『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』](#)
IBM MQ 프로그래밍 서비스를 사용하려면 프로그램을 큐 관리자에 연결해야 합니다. 이 정보를 사용하여 큐 관리자에 연결하고 큐 관리자에서 연결을 끊는 방법에 대해 알아보십시오.

[763 페이지의 『오브젝트 열기 및 닫기』](#)
이 정보는 IBM MQ 오브젝트의 열기 및 닫기 조작에 대한 통찰력을 제공합니다.

[772 페이지의 『큐에 메시지 넣기』](#)
이 정보를 사용하여 메시지를 큐에 넣는 방법에 대해 알아보십시오.

[786 페이지의 『큐에서 메시지 가져오기』](#)
이 정보를 사용하여 큐에서 메시지를 가져오는 방법에 대해 알아봅니다.

[861 페이지의 『오브젝트 속성 조회 및 설정』](#)
속성은 IBM MQ 오브젝트의 특성을 정의하는 특성입니다.

[863 페이지의 『작업 단위 커밋 및 백아웃』](#)
이 정보는 작업 단위에서 발생한 복구 가능한 가져오기 및 넣기 조작을 커밋하고 백아웃하는 방법을 설명합니다.

[873 페이지의 『트리거를 사용한 IBM MQ 애플리케이션 시작』](#)
트리거와 트리거를 사용하여 IBM MQ 애플리케이션을 시작하는 방법에 대해 학습하십시오.

[895 페이지의 『IBM MQ for z/OS의 애플리케이션 사용 및 작성』](#)
IBM MQ for z/OS 애플리케이션은 여러 다른 환경에서 실행되는 프로그램으로 구성할 수 있습니다. 이는 둘 이상의 환경에서 사용 가능한 기능을 활용할 수 있음을 의미합니다.

[71 페이지의 『IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션』](#)
이 정보는 IBM MQ을(를) 사용하여 IMS 애플리케이션을 작성하는 데 도움이 됩니다.

MQOPEN 및 클러스터

클러스터 큐가 열릴 때 메시지를 넣고 읽을 큐는 MQOPEN 호출에 따라 달라집니다.

대상 큐 선택

오브젝트 디스크립터 MQOD에 큐 관리자 이름을 제공하지 않으면, 큐 관리자는 메시지를 송신할 큐 관리자를 선택합니다. 오브젝트 디스크립터에 큐 관리자 이름을 제공하면, 메시지는 항상 사용자가 선택한 큐 관리자에 송신됩니다.

큐 관리자가 대상 큐 관리자를 선택할 경우, 바인딩 옵션(MQOO_BIND_*) 및 로컬 큐의 존재 여부에 따라 선택이 달라집니다. 큐의 로컬 인스턴스가 있는 경우, CLWLUSEQ 속성이 ANY로 설정되지 않으면 로컬 인스턴스는 항상 리모트 인스턴스에 우선하여 열립니다. 그렇지 않은 경우 선택은 바인딩 옵션에 따라 다릅니다. 클러스터에서 [메](#)

시지 그룹을 사용할 때는 그룹 내의 모든 메시지가 동일한 대상에서 처리되도록 하기 위해 또는 MQ00_BIND_ON_GROUP 중 MQ00_BIND_ON_OPEN 하나를 지정해야 합니다.

큐 관리자는 대상 큐 관리자를 선택할 경우 워크로드 관리 알고리즘을 사용하여 라운드 로빈 방식으로 해당 작업을 수행합니다. 클러스터의 워크로드 밸런싱을 참조하십시오.

워크로드 밸런싱 알고리즘이 사용되는 시기는 클러스터 큐가 열리는 방식에 따라 달라집니다.

- MQ00_BIND_ON_OPEN - 애플리케이션이 큐를 열 때 한 번만 알고리즘이 사용됩니다.
- MQ00_BIND_NOT_FIXED - 큐에 넣어진 모든 메시지에 알고리즘이 사용됩니다.
- MQ00_BIND_ON_GROUP - 각 메시지 그룹을 시작할 때 한 번만 알고리즘이 사용됩니다.

MQ00_BIND_ON_OPEN

MQOPEN 호출의 MQ00_BIND_ON_OPEN 옵션은 대상 큐 관리자가 수정되도록 지정합니다. 클러스터 내에 동일한 큐의 인스턴스가 여럿 있는 경우 MQ00_BIND_ON_OPEN 옵션을 사용하십시오. MQOPEN 호출에서 리턴된 오브젝트 핸들을 지정하는 큐에 넣어진 모든 메시지가 동일한 큐 관리자에 전달됩니다.

- 메시지에 연관관계가 있는 경우 MQ00_BIND_ON_OPEN 옵션을 사용하십시오. 예를 들어, 일괄된 메시지를 모두 동일한 큐 관리자에서 처리하려면 큐를 열 때 MQ00_BIND_ON_OPEN을 지정하십시오. IBM MQ는 큐 관리자와, 해당 큐에 넣어진 모든 메시지에 사용되는 라우트를 수정합니다.
- MQ00_BIND_ON_OPEN 옵션을 지정한 경우 큐의 새 인스턴스를 선택하려면 큐를 다시 열어야 합니다.

MQ00_BIND_NOT_FIXED

MQOPEN 호출의 MQ00_BIND_NOT_FIXED 옵션은 대상 큐 관리자가 수정되지 않도록 지정합니다. MQOPEN 호출에서 리턴된 오브젝트 핸들을 지정하는 큐에 기록된 메시지는 메시지별로 MQPUT 시간에 큐 관리자로 라우팅됩니다. 모든 메시지가 동일한 목적지에 기록되도록 강제 실행하지 않으려면 MQ00_BIND_NOT_FIXED 옵션을 사용하십시오.

- MQ00_BIND_NOT_FIXED 및 MQMF_SEGMENTATION_ALLOWED를 동시에 지정하지 마십시오. 이를 동시에 지정할 경우 메시지의 세그먼트가 각기 다른 큐 관리자에 전달되고 클러스터 전체에 분산됩니다.

MQ00_BIND_ON_GROUP

메시지 그룹이 동일한 목적지 인스턴스에 할당되도록 애플리케이션에서 요청할 수 있게 합니다. 이 옵션은 큐에 대해서만 유효하며 클러스터 큐에만 영향을 미칩니다. 클러스터 큐가 아닌 큐에 대해 지정된 경우에는 옵션이 무시됩니다.

- MQPMO_LOGICAL_ORDER가 MQPUT에 지정된 경우, 그룹이 단일 목적지로만 라우팅됩니다. MQ00_BIND_ON_GROUP이 지정되었으나 메시지가 논리 그룹의 일부가 아닌 경우, BIND_NOT_FIXED 작동이 대신 사용됩니다.

참고: BIND_ON_GROUP을 사용하여 연결이 열리면 그룹화된 메시지가 클러스터 큐로 전송됩니다. 그 사이에, 클러스터 수신자 큐가 MAXDEPTH에 도달한 후 PUT 작업이 비활성화되면, 수신자 큐 관리자는 이후 그룹 내 잔여 부분 메시지에 대해 부하 분산을 강제로 수행하게 됩니다. 기본적으로 이 할당은 라운드 로빈 패턴을 따릅니다. 수신자 큐 관리자는 발신 측에서 이루어진 원래의 부하 분산 결정을 알지 못하기 때문입니다.

MQ00_BIND_AS_Q_DEF

MQ00_BIND_ON_OPEN, MQ00_BIND_NOT_FIXED 또는 MQ00_BIND_ON_GROUP을 지정하지 않은 경우 기본 옵션은 MQ00_BIND_AS_Q_DEF입니다. MQ00_BIND_AS_Q_DEF를 사용하면 큐 핸들에 사용되는 바 인딩이 DefBind 큐 속성으로 수행됩니다.

MQOPEN 옵션의 관련성

MQOPEN 옵션 MQ00_BROWSE, MQ00_INPUT_* 또는 MQ00_SET을 사용할 경우 MQOPEN이 성공하려면 클러스터 큐의 로컬 인스턴스가 필요합니다.

MQOPEN 옵션 MQ00_OUTPUT, MQ00_BIND_* 또는 MQ00_INQUIRE의 경우에는 성공하는 데 클러스터 큐의 로컬 인스턴스가 필요하지 않습니다.

해석된 큐 관리자 이름

MQOPEN 시간에 큐 관리자 이름이 해석되면 해석된 이름이 애플리케이션으로 리턴됩니다. 애플리케이션이 후속 MQOPEN 호출에서 이 이름을 사용하려고 해도 해당 이름에 액세스할 수 있는 권한이 부여되지 않습니다.

MQPUT, MQPUT1 및 클러스터

MQOO_BIND_NOT_FIXED가 MQOPEN에 지정되면, 워크로드 관리 루틴은 목적지 MQPUT 또는 MQPUT1이 선택하는 목적지를 선택합니다.

MQOO_BIND_NOT_FIXED가 MQOPEN 호출에 지정되면, 각 후속 MQPUT 호출은 워크로드 관리 루틴을 호출하여 메시지를 송신할 큐 관리자를 판별합니다. 사용할 목적지 및 라우트는 메시지별로 선택됩니다. 네트워크에서 조건이 변경되면, 메시지를 넣은 후에 목적지 및 라우트가 변경될 수 있습니다. MQPUT1 호출은 항상 MQOO_BIND_NOT_FIXED가 적용된 것처럼 작동합니다. 즉, 워크로드 관리 루틴을 호출합니다.

워크로드 관리 루틴이 큐 관리자를 선택하면 로컬 큐 관리자는 Put 조작을 완료합니다. 메시지는 다양한 큐에 배치할 수 있습니다.

1. 목적지가 큐의 로컬 인스턴스이면 메시지는 로컬 큐에 배치됩니다.
2. 목적지가 클러스터의 큐 관리자이면 메시지는 클러스터 전송 큐에 배치됩니다.
3. 목적지가 클러스터 외부의 큐 관리자이면, 메시지는 대상 큐 관리자와 동일한 이름으로 전송 큐에 배치됩니다.

MQOO_BIND_ON_OPEN이 MQOPEN 호출에 지정되면, 목적지와 라우트가 이미 선택되었기 때문에 MQPUT 호출은 워크로드 관리 루틴을 호출하지 않습니다.

MQINQ 및 클러스터

조회되는 클러스터 큐는 MQOO_INQUIRE와 결합하는 옵션에 따라 다릅니다.

큐를 조회하려면 먼저 MQOPEN 호출을 사용하여 열고 MQOO_INQUIRE를 지정하십시오.

클러스터 큐를 조회하려면 MQOPEN 호출을 사용하고 다른 옵션을 MQOO_INQUIRE와 결합하십시오. 조회할 수 있는 속성은 클러스터 큐의 로컬 인스턴스가 있는지 여부와 큐를 여는 방법에 따라 달라집니다.

- MQOO_BROWSE, MQOO_INPUT_* 또는 MQOO_SET을 MQOO_INQUIRE와 결합할 경우 열기에 성공하려면 클러스터 큐의 로컬 인스턴스가 필요합니다. 이 경우에 로컬 큐에 유효한 모든 속성을 조회할 수 있습니다.
- MQOO_OUTPUT을 MQOO_INQUIRE와 결합하고 선행 옵션을 지정하지 않을 경우에 열리는 인스턴스는 다음 중 하나입니다.
 - 로컬 큐 관리자의 인스턴스(있는 경우). 이 경우에 로컬 큐에 유효한 모든 속성을 조회할 수 있습니다.
 - 클러스터의 다른 위치에 있는 인스턴스(로컬 큐 관리자 인스턴스가 없는 경우). 이 경우에 다음 속성만 조회할 수 있습니다. 이 경우 QType 속성의 값은 MQQT_CLUSTER입니다.
 - DefBind
 - DefPersistence
 - DefPriority
 - InhibitPut
 - QDesc
 - QName
 - QType

클러스터 큐의 DefBind 속성을 조회하려면 선택자 MQIA_DEF_BIND와 함께 MQINQ 호출을 사용하십시오. 리턴되는 값은 MQBND_BIND_ON_OPEN이나 MQBND_BIND_NOT_FIXED 또는 MQBND_BIND_ON_GROUP입니다. 클러스터와 함께 그룹을 사용할 때는 또는 MQBND_BIND_ON_GROUP 중 하나를 MQBND_BIND_ON_OPEN 반드시 지정해야 합니다.

큐의 로컬 인스턴스의 CLUSTER 및 CLUSNL 속성을 조회하려면, 선택자 MQCA_CLUSTER_NAME 또는 선택자 MQCA_CLUSTER_NAMELIST와 함께 MQINQ 호출을 사용하십시오.

참고: MQOPEN이 바인드된 큐를 수정하지 않고 클러스터 큐를 열면, 연속 MQINQ 호출이 클러스터 큐의 다른 인스턴스를 조회할 수 있습니다.

관련 정보

769 페이지의 『클러스터 큐에 대한 MQOPEN 옵션』

MQSET 및 클러스터

MQOPEN 옵션의 MQOO_SET 옵션에는 MQSET이 성공할 수 있도록 클러스터 큐의 로컬 인스턴스가 있어야 합니다.

MQSET 호출은 클러스터의 임의의 위치에서 큐의 속성을 설정하는 데 사용할 수 없습니다.

클러스터 속성으로 정의된 로컬 알리어스 또는 리모트 큐를 열고 MQSET 호출을 사용할 수 있습니다. 로컬 알리어스 또는 리모트 큐의 속성을 설정할 수 있습니다. 대상 큐가 다른 큐 관리자에 정의된 클러스터 큐인지 여부는 중요하지 않습니다.

리턴 코드

클러스터에 특정한 리턴 코드

MQRC_CLUSTER_EXIT_ERROR(2266 X'8DA')

MQOPEN, MQPUT 또는 MQPUT1 호출을 발행하여 클러스터 큐를 열거나 해당 큐에 메시지를 넣습니다. 큐 관리자의 ClusterWorkloadExit 속성에 의해 정의된 클러스터 워크로드 엑시트가 예상치 못하게 실패하거나 제시간에 응답하지 않습니다.

z/OS IBM MQ for z/OS의 시스템 로그에 이 오류에 대한 자세한 정보를 제공하는 메시지가 기록됩니다.

이 큐 핸들에 대한 후속 MQOPEN, MQPUT 및 MQPUT1 호출은 ClusterWorkloadExit 속성이 공백인 것처럼 처리됩니다.

MQRC_CLUSTER_EXIT_LOAD_ERROR(2267 X'8DB')

z/OS z/OS에서 클러스터 워크로드 엑시트를 로드할 수 없습니다.

메시지가 시스템 로그에 기록되고 ClusterWorkloadExit 속성이 공백인 것처럼 처리가 계속됩니다.

Multi 다중 플랫폼에서는 MQCONN 또는 MQCONNx 호출을 실행하여 큐 관리자에 연결합니다. 큐 관리자의 큐 관리자 ClusterWorkloadExit 속성에 의해 정의된 클러스터 워크로드 엑시트를 로드할 수 없기 때문에 호출은 실패합니다.

MQRC_CLUSTER_PUT_INHIBITED(2268 X'8DC')

클러스터 큐에 대해 MQOO_OUTPUT 및 MQOO_BIND_ON_OPEN 옵션이 적용된 MQOPEN 호출이 발행됩니다. 클러스터에서 큐의 모든 인스턴스는 InhibitPut 속성이 MQQA_PUT_INHIBITED로 설정되어 있으므로 현재 넣기 금지되어 있습니다. 메시지를 수신할 수 있는 큐 인스턴스가 없기 때문에 MQOPEN 호출은 실패합니다.

다음 두 명령문이 모두 true일 때만 이 이유 코드가 발생합니다.

- 큐의 로컬 인스턴스가 없습니다. 로컬 인스턴스가 있으면, 로컬 인스턴스가 넣기 금지된 경우에도 MQOPEN 호출은 성공합니다.
- 큐에 대한 클러스터 워크로드 엑시트가 없거나, 클러스터 워크로드 엑시트는 있지만 해당 엑시트가 큐 인스턴스를 선택하지 않습니다. (클러스터 워크로드 엑시트가 큐 인스턴스를 선택하면, 해당 인스턴스가 넣기 금지된 경우에도 MQOPEN 호출은 성공합니다.)

MQOO_BIND_NOT_FIXED 옵션이 MQOPEN 호출에 지정되면, 클러스터에서 모든 큐가 넣기 금지된 경우에도 호출은 성공할 수 있습니다. 그러나 모든 큐가 해당 호출 시점에 여전히 넣기 금지되어 있으면 후속 MQPUT 호출은 실패할 수 있습니다.

MQRC_CLUSTER_RESOLUTION_ERROR(2189 X'88D')

1. MQOPEN, MQPUT 또는 MQPUT1 호출을 발행하여 클러스터 큐를 열거나 해당 큐에 메시지를 넣습니다. 전체 저장소 큐 관리자의 응답이 필요하지만 사용 불가능하기 때문에 큐 정의를 올바르게 해석할 수 없습니다.
2. PUBSCOPE (ALL) 또는 SUBSCOPE (ALL)를 지정하는 토픽 객체에 대해, MQPUTMQOPEN, MQPUT1 또는 MQSUB 호출이 발행됩니다. 전체 리포지토리 큐 관리자의 응답이 필요하지만 사용할 수 없기 때문에 클러스터 토픽 정의를 올바르게 해결할 수 없습니다.

MQRC_CLUSTER_RESOURCE_ERROR(2269 X'8DD')

클러스터 큐에 대해 MQOPEN, MQPUT 또는 MQPUT1 호출이 발행됩니다. 클러스터링에 필요한 자원을 사용하려고 하는 동안 오류가 발생합니다.

MQRC_NO_DESTINATIONS_AVAILABLE(2270 X'8DE')

클러스터 큐에 메시지를 넣도록 MQPUT 또는 MQPUT1 호출이 발행됩니다. 호출 시점에는 클러스터에 더 이상 큐의 인스턴스가 없습니다. MQPUT은 실패하고 메시지가 송신되지 않습니다.

큐를 여는 MQOPEN 호출에 MQOO_BIND_NOT_FIXED가 지정되어 있거나 메시지를 넣기 위해 MQPUT1이 사용되는 경우 오류가 발생할 수 있습니다.

MQRC_STOPPED_BY_CLUSTER_EXIT(2188 X'88C')

클러스터 큐를 열거나 해당 큐에 메시지를 넣도록 MQOPEN, MQPUT 또는 MQPUT1 호출이 발행됩니다. 클러스터 워크로드 엑시트가 호출을 거부합니다.

z/OS IBM MQ for z/OS의 애플리케이션 사용 및 작성

IBM MQ for z/OS 애플리케이션은 여러 다른 환경에서 실행되는 프로그램으로 구성할 수 있습니다. 이는 둘 이상의 환경에서 사용 가능한 기능을 활용할 수 있음을 의미합니다.

이 정보는 지원되는 각 환경에서 실행 중인 프로그램에서 사용 가능한 IBM MQ 기능에 대해 설명합니다. 또한,

- IBM MQ-CICS bridge사용에 대한 정보는 [CICS와 함께 IBM MQ 사용](#)을 참조하십시오.
- IMS 및 IMS 브릿지 사용에 대한 정보는 [71 페이지의 『IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션』](#)의 내용을 참조하십시오.

IBM MQ for z/OS의 애플리케이션 사용 및 작성에 대해 자세히 알아보려면 다음 링크를 사용하십시오.

- [896 페이지의 『환경 종속적 IBM MQ for z/OS 기능』](#)
- [896 페이지의 『디버깅 기능, 동기점 지원 및 복구 지원』](#)
- [897 페이지의 『애플리케이션 환경과의 IBM MQ for z/OS 인터페이스』](#)
- [899 페이지의 『z/OS UNIX System Services 애플리케이션 작성』](#)
- [902 페이지의 『공유 큐를 사용한 애플리케이션 프로그래밍』](#)

관련 개념

[744 페이지의 『MQI\(Message Queue Interface\) 개요』](#)
MQI(Message Queue Interface) 컴포넌트에 대해 학습합니다.

[756 페이지의 『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』](#)

IBM MQ 프로그래밍 서비스를 사용하려면 프로그램을 큐 관리자에 연결해야 합니다. 이 정보를 사용하여 큐 관리자에 연결하고 큐 관리자에서 연결을 끊는 방법에 대해 알아보십시오.

[763 페이지의 『오브젝트 열기 및 닫기』](#)

이 정보는 IBM MQ 오브젝트의 열기 및 닫기 조작에 대한 통찰력을 제공합니다.

[772 페이지의 『큐에 메시지 넣기』](#)

이 정보를 사용하여 메시지를 큐에 넣는 방법에 대해 알아보십시오.

[786 페이지의 『큐에서 메시지 가져오기』](#)

이 정보를 사용하여 큐에서 메시지를 가져오는 방법에 대해 알아보십시오.

[861 페이지의 『오브젝트 속성 조회 및 설정』](#)

속성은 IBM MQ 오브젝트의 특성을 정의하는 특성입니다.

863 페이지의 『작업 단위 커밋 및 백아웃』

이 정보는 작업 단위에서 발생한 복구 가능한 가져오기 및 넣기 조작을 커밋하고 백아웃하는 방법을 설명합니다.

873 페이지의 『트리거를 사용한 IBM MQ 애플리케이션 시작』

트리거와 트리거를 사용하여 IBM MQ 애플리케이션을 시작하는 방법에 대해 학습하십시오.

891 페이지의 『MQI 및 클러스터에 대한 작업』

클러스터와 관련된 호출 및 리턴 코드에 대한 특수 옵션이 있습니다.

71 페이지의 『IBM MQ for z/OS 의 IMS 및 IMS 브릿지 애플리케이션』

이 정보는 IBM MQ(를) 사용하여 IMS 애플리케이션을 작성하는 데 도움이 됩니다.

▶ **z/OS** 환경 종속적 **IBM MQ for z/OS** 기능

IBM MQ for z/OS 기능을 고려할 경우 이 정보를 사용합니다.

다음과 같은 환경에서 IBM MQ for z/OS 실행되는 함수 IBM MQ 간 주요 차이점은 다음과 같습니다:

- IBM MQ for z/OS는 다음 트리거 모니터를 제공합니다.

- CICS 환경에서 사용되는 CKTI
- IMS 환경에서 사용되는 CSQQTRMN

다른 환경에서 애플리케이션을 시작하려면 사용자 자체 모듈을 작성해야 합니다.

- 2단계 커밋을 사용한 동기점 조정은 CICS 및 IMS 환경에서 지원됩니다. 이는 트랜잭션 관리와 복구 가능한 자원 관리자 서비스(RRS)를 사용하여 z/OS 배치 환경에서도 지원됩니다. 단일 단계 커밋은 환경 z/OS 자체에서 IBM MQ 지원됩니다.
- 배치 및 IMS 환경의 경우, MQI는 프로그램을 큐 관리자에 연결하는 호출과 해당 연결을 끊는 호출을 제공합니다. 프로그램은 둘 이상의 큐 관리자에 연결할 수 있습니다.
- CICS 시스템은 하나의 큐 관리자에만 연결할 수 있습니다. 이러한 연결은 CICS 시스템 시동 작업에서 서브시스템 이름이 정의된 경우 CICS를 시작할 때 설정될 수 있습니다. MQI 연결 및 연결 해제 호출은 CICS 환경에서 허용되지만 유효하지 않습니다.
- API 교차 엑시트를 사용하면 프로그램이 모든 MQI 호출의 처리에 개입할 수 있습니다. 이 엑시트는 CICS 환경에서만 사용 가능합니다.
- 멀티프로세서 시스템의 CICS에서는 MQI 호출이 여러 z/OS TCB에서 실행될 수 있기 때문에 성능상의 이점이 있습니다. 자세한 정보는 [~할 계획입니다 z/OS IBM MQ for z/OS](#) 개념 및 계획 안내서를 참조하십시오.

이러한 기능은 896 페이지의 표 145에 요약되어 있습니다.

표 145. z/OS 환경 기능			
	CICS	IMS	배치/TSO
제공된 트리거 모니터	예	예	아니오
2단계 커밋	예	예	예
1단계 커밋	예	아니오	예
MQI 연결/연결 해제 호출	허용	예	예
API 교차 엑시트(API-crossing exit)	예	아니오	아니오

참고: 2단계 커밋은 RRS를 사용하여 배치/TSO 환경에서 지원됩니다.

▶ **z/OS** 디버깅 기능, 동기점 지원 및 복구 지원

이 정보를 사용하여 프로그램 디버깅 기능, 동기점 지원 및 복구 지원에 대해 알아봅니다.

프로그램 디버깅 기능

IBM MQ for z/OS는 모든 환경에서 프로그램을 디버깅하기 위해 사용할 수 있는 추적 기능을 제공합니다.

또한 CICS 환경에서 다음을 사용할 수 있습니다.

- CEDF(CICS Execution Diagnostic Facility)
- CETR(CICS Trace Control Transaction)
- IBM MQ for z/OS API 교차 엑시트

z/OS 플랫폼에서 사용 중인 프로그래밍 언어가 지원하는 사용 가능한 모든 대화식 디버깅 도구를 사용할 수 있습니다.

동기점 지원

트랜잭션 처리 환경에서 트랜잭션 처리를 안전하게 사용할 수 있도록 하려면 작업 단위의 시작과 끝을 동기화해야 합니다.

이는 및 IMS CICS 환경에서 IBM MQ for z/OS 완전히 지원됩니다. 완전한 지원은 CICS 또는 IMS의 제어 하에 작업 단위를 한꺼번에 커밋하거나 백아웃할 수 있도록 하는 자원 관리자 간 협업을 의미합니다. 자원 관리자의 예는 Db2, CICS 파일 제어, IMS 및 IBM MQ for z/OS입니다.

z/OS 배치 애플리케이션은 IBM MQ for z/OS 호출을 사용하여 1단계 커밋 기능을 제공할 수 있습니다. 이는 다른 자원 관리자에 관계없이 애플리케이션에 정의된 큐 조작 세트를 커밋하거나 백아웃할 수 있다는 의미입니다.

2단계 커밋은 트랜잭션 관리와 복구 가능한 자원 관리자 서비스(RRS)를 사용하여 z/OS 배치 환경에서도 지원됩니다. 추가 정보는 [z/OS 배치 애플리케이션의 동기점을 참조하십시오](#).

복구 지원

큐 관리자와 CICS 또는 IMS 시스템 사이의 연결이 트랜잭션 중에 중단되면 일부 작업 단위가 백아웃되지 않을 수 있습니다.

하지만 이러한 작업 단위는 CICS 또는 IMS 시스템과의 연결이 재설정될 때 (동기점 관리자의 제어 하에) 큐 관리자에서 해결됩니다.

▶ **z/OS** 애플리케이션 환경과의 IBM MQ for z/OS 인터페이스

서로 다른 환경에서 실행하는 애플리케이션이 메시지 큐잉 네트워크를 통해 메시지를 보내고 받을 수 있도록 IBM MQ for z/OS은(는) 지원하는 각 환경에 대한 어댑터를 제공합니다.

이러한 어댑터는 애플리케이션 프로그램과 IBM MQ for z/OS 서브시스템 사이의 인터페이스입니다. 해당 어댑터를 통해 프로그램에서 MQI를 사용할 수 있습니다.

▶ **z/OS** 배치 어댑터

이 정보를 사용하여 배치 어댑터 및 배치 어댑터가 지원하는 커밋 프로토콜에 대해 알아봅니다.

배치 어댑터는 다음 상태에서 실행 중인 프로그램의 IBM MQ for z/OS 자원에 대한 액세스를 제공합니다.

- 태스크(TCB) 모드
- 문제점 또는 수퍼바이저 상태
- 1차 주소 공간 제어 모드

프로그램은 교차 메모리 모드가 아니어야 합니다.

애플리케이션 프로그램과 IBM MQ for z/OS 간의 연결은 태스크 레벨에 존재합니다. 어댑터는 애플리케이션 TCB(Task Control Block)에서 IBM MQ for z/OS로의 단일 연결 스레드를 제공합니다.

어댑터는 IBM MQ for z/OS에 소유된 자원의 변경을 위한 1단계 커밋 프로토콜을 지원하지만 다단계 커밋 프로토콜은 지원하지 않습니다.

RRS 배치 어댑터

이 정보를 사용하여 RRS 배치 어댑터 및 IBM MQ가 제공하는 두 가지 RRS 배치 어댑터에 대해 알아봅니다.

트랜잭션 관리 및 복구 가능한 자원 관리자 서비스(RRS) 어댑터:

- 커밋 제어에 z/OS RRS를 사용합니다.
- 단일 태스크에서 단일 z/OS 인스턴스 상에서 실행 중인 여러 IBM MQ 하위 시스템에 대한 동시 연결을 지원합니다.
- 다음과 관련하여 z/OS RRS를 준수하는 복구 가능한 관리자를 통해 액세스된 복구 가능한 자원에 대해 z/OS 전체의 통합 커밋 제어(z/OS RRS 사용)를 제공합니다.
 - RRS 배치 어댑터를 사용하여 IBM MQ에 연결하는 애플리케이션
 - Db2- 저장 프로시저가 실행되는 저장 프로시저 Db2주소 공간은 워크로드 관리자(WLM)에 의해 관리됩니다 z/OS.
- TCB 간에 IBM MQ 배치 스레드를 전환시키는 기능을 지원합니다.

IBM MQ for z/OS는 두 가지 RRS 배치 어댑터를 제공합니다.

CSQBRSTB

이 어댑터를 사용하려면 IBM MQ 애플리케이션에서 MQCMIT문을 SRRCMIT로 변경하고 MQBACK문을 SRRBACK으로 변경해야 합니다. (CSQBRSTB와 링크된 애플리케이션에서 MQCMIT 또는 MQBACK을 코딩하는 경우 MQRC_ENVIRONMENT_ERROR를 수신합니다.)

CSQBRRSI

이 어댑터를 사용하면 IBM MQ 애플리케이션이 MQCMIT 및 MQBACK 또는 SRRCMIT 및 SRRBACK을 사용할 수 있습니다.

참고: CSQBRSTB 및 CSQBRRSI는 연계 속성인 AMODE(31) RMODE(ANY)와 함께 제공됩니다. 애플리케이션이 각각의 스텝을 16MB 행 아래에 로드하는 경우 먼저 해당 스텝을 RMODE(24)에 다시 링크하십시오.

마이그레이션

기존 배치/TSO IBM MQ 애플리케이션을 마이그레이션하여 변경이 거의 없거나 전혀 없는 상태로 RRS 조정을 사용할 수 있습니다.

CSQBRRSI 어댑터로 IBM MQ 애플리케이션을 링크 편집하는 경우, MQCMIT 및 MQBACK은 IBM MQ 및 다른 모든 RRS 사용 자원 관리자에서 작업 단위의 동기점을 조정합니다. CSQBRSTB 어댑터로 IBM MQ 애플리케이션을 링크 편집하는 경우, MQCMIT를 SRRCMIT로 변경하고 MQBACK을 SRRBACK으로 변경하십시오. 동기점이 IBM MQ 자원으로만 제한되지 않음을 분명하게 나타낸다는 점에서 후자의 방법이 더 바람직합니다.

IMS 어댑터

IBM MQ for z/OS 시스템에서 IMS 어댑터를 사용하는 경우 IMS이(가) 최대 100MB의 메시지를 수용할 수 있는 충분한 스토리지를 확보할 수 있는지 확인하십시오.

사용자 참고사항

IMS 어댑터는 IBM MQ for z/OS 자원에 대한 액세스를 제공합니다.

- 온라인 메시지 처리 프로그램(MPP)
- 대화식 단축 경로 프로그램(IFP)
- 배치 메시지 처리 프로그램(BMP)

이러한 자원을 사용하려면, 프로그램은 태스크(TCB) 모드 및 문제점 상태에서 실행 중이어야 하고 교차 메모리 모드 또는 액세스 등록 모드여서는 안 됩니다.

어댑터는 애플리케이션 TCB(Task Control Block)에서 IBM MQ로의 연결 스레드를 제공합니다. 이 어댑터는 가 소유한 리소스에 대한 변경 사항에 대해 IBM MQ for z/OS 2단계 커밋 프로토콜을 지원하며, 가 동기화점 IMS 조정자 역할을 수행합니다.

또한 어댑터는 큐의 특정 트리거 조건이 충족될 때 자동으로 프로그램을 시작할 수 있는 트리거 모니터 프로그램을 제공합니다. 자세한 정보는 873 페이지의 『트리거를 사용한 IBM MQ 애플리케이션 시작』의 내용을 참조하십시오.

배치 DL/I 프로그램을 작성할 경우, 이 주제에서 z/OS 배치 프로그램에 대해 제공된 자세한 내용을 따르십시오.

▶ z/OS z/OS UNIX System Services 애플리케이션 작성

배치 어댑터는 배치 및 TSO 주소 공간에서 큐 관리자 연결을 지원합니다.

배치 주소 공간의 경우, 어댑터는 해당 주소 공간 내의 여러 TCB로부터 다음과 같이 연결을 지원합니다:

- 각 TCB는 MQCONN 또는 MQCONNEX 호출을 사용하여 여러 큐 관리자에 연결할 수 있습니다. (단, TCB에는 어느 때든지 특정 큐 관리자에 대한 연결 인스턴스가 하나만 있을 수 있습니다.)
- 다중 TCB는 동일한 큐 관리자에 연결할 수 있습니다. (단, MQCONN 또는 MQCONNEX 호출에서 리턴된 큐 관리자 핸들은 발행 TCB에 바인드되어 있으므로 다른 TCB에서 사용할 수 없습니다.)

z/OS UNIX System Services는 두 가지 유형의 pthread_create 호출을 지원합니다.

1. 중량 스레드로, 각 TCB에 대해 하나만 실행되고 z/OS에서 스레드 시작 및 종료 시 ATTACH 및 DETACH됩니다.
2. 중간 가중치 스레드로, 각 TCB에 대해 하나만 실행되지만 해당 TCB가 장기 실행 TCB 풀에 속할 수 있습니다. 서버에 연결되면 태스크(TCB) 종료 시 서버가 제공하는 기본 스레드 종료가 반드시 구동되지는 않기 때문에, 애플리케이션은 필요한 모든 애플리케이션 정리를 수행해야 합니다.

경량 스레드는 지원되지 않습니다. (애플리케이션은 자체 작업 요청을 디스패치하는 영구 스레드를 작성하는 경우 다음 작업 요청을 시작하기 전에 모든 자원을 정리해야 합니다.)

IBM MQ for z/OS는 다음과 같이 배치 어댑터를 사용하여 z/OS UNIX System Services 스레드를 지원합니다.

1. 중량 스레드는 완전히 배치 연결로 지원됩니다. 각 스레드는 자체 TCB에서 실행되며, 이 TCB는 스레드 시작 및 종료 시 첨부 및 분리됩니다. MQDISC 호출을 발행하기 전에 스레드가 종료되면, IBM MQ for z/OS는 해당 표준 태스크 정리를 수행합니다. 여기에는 스레드가 정상 종료한 경우에는 미해결 작업 단위를 커밋하고 스레드가 비정상 종료한 경우에는 백아웃하는 작업이 포함됩니다.
2. 중간 가중치 스레드는 완전히 지원되지만 TCB가 다른 스레드에서 재사용될 경우, 애플리케이션은 다음 스레드가 시작되기 전에 MQCMIT 또는 MQBACK에 뒤이어 MQDISC 호출이 발행되는지 확인해야 합니다. 이는 애플리케이션이 프로그램 인터럽트 핸들러를 설정한 후 애플리케이션이 이상종료되는 경우 인터럽트 핸들러가 다른 스레드에서 TCB를 재사용하기 전에 MQCMIT 및 MQDISC 호출을 발행해야 한다는 것을 의미합니다.

참고: 스레딩 모델은 여러 스레드에서 공통 IBM MQ 리소스에 대한 접근을 지원하지 않습니다.

▶ z/OS z/OS의 API 교차 엑시트

이 주제에는 제품별 프로그래밍 인터페이스 정보가 포함되어 있습니다.

엑시트는 IBM 제공 코드에서 고유 코드를 실행할 수 있는 지점입니다. IBM MQ for z/OS는 MQI에 대한 호출을 인터셉트하고 MQI 호출의 함수를 모니터링하거나 수정하기 위해 사용할 수 있는 API 교차 엑시트를 제공합니다. 이 절에서는 API 교차 엑시트를 사용하는 방법에 대해 설명하고 IBM MQ for z/OS와 함께 제공되는 샘플 엑시트 프로그램에 대해 설명합니다.

이 절은 CICS TS V3.1 및 서버의 사용자에게만 적용 가능합니다. CICS TS V3.2 이상의 사용자는 CICS 제품 문서에서 IBM MQ와 CICS의 통합 절을 참조하십시오.

참고

API 교차 출구는 오직 의 CICS 어댑터에 IBM MQ for z/OS의해서만 호출됩니다. 엑시트 프로그램은 CICS 주소 공간에서 실행됩니다.

▶ z/OS 고유 엑시트 프로그램 작성

IBM MQ for z/OS와 함께 제공되는 샘플 API 교차 엑시트 프로그램을 고유 프로그램의 프레임워크로 사용할 수 있습니다.

이 프로그램은 900 페이지의 『샘플 API 교차 엑시트 프로그램, CSQCAPX』에서 설명합니다.

엑시트 프로그램을 작성할 때, 애플리케이션에서 발행한 MQI 호출의 이름을 찾으려면 MQXP 구조의 *ExitCommand* 필드를 조사하십시오. 호출에서 매개변수의 수를 찾으려면 *ExitParmCount* 필드를 조사하십시오. 16바이트 *ExitUserArea* 필드를 사용하여 애플리케이션이 확보한 동적 스토리지의 주소를 저장할 수 있습니다. 이 필드는 엑시트의 호출 사이에 보유하고, CICS 태스크와 동일한 활성 시간을 가지고 있습니다.

CICS Transaction Server V3.2를 사용할 경우, 엑시트 프로그램을 스레드에 안전하게 작성하고 엑시트 프로그램을 스레드 안전으로 선언해야 합니다. 초기 CICS 릴리스를 사용할 경우에도 엑시트 프로그램을 작성하고 스레드 안전으로 선언하여 CICS Transaction Server V3.2로의 마이그레이션을 준비하는 것이 좋습니다.

엑시트 프로그램은 *ExitResponse* 필드에서 MQXCC_SUPPRESS_FUNCTION 또는 MQXCC_SKIP_FUNCTION을 리턴하여 MQI 호출의 실행을 억제할 수 있습니다. 호출이 실행되고 호출이 완료된 후 엑시트 프로그램이 다시 호출되도록 하려면, 엑시트 프로그램이 MQXCC_OK를 리턴해야 합니다.

MQI 호출 이후에 호출되는 엑시트 프로그램은 호출로 설정된 완료 및 이유 코드를 검사하고 수정할 수 있습니다.

사용법 참고

엑시트 프로그램을 작성할 때 고려해야 할 몇 가지 일반 사항은 다음과 같습니다.

- 성능상의 이유 때문에 어셈블러 언어로 프로그램을 작성하십시오. IBM MQ for z/OS에서 지원되는 다른 언어로 프로그램을 작성하는 경우 고유 데이터 정의 파일을 제공해야 합니다.
- 프로그램을 AMODE(31) 및 RMODE(ANY)로 링크 편집하십시오.
- 프로그램에 엑시트 매개변수 블록을 정의하려면 어셈블러 언어 매크로 CMQXPA를 사용하십시오.
- 엑시트 프로그램이 호출하는 모든 프로그램과 엑시트 프로그램을 정의할 때 CONCURRENCY(THREADSAFE)를 지정하십시오.
- CICS Transaction Server for z/OS 스토리지 보호 기능을 사용할 경우 CICS 실행 키로 프로그램을 실행해야 합니다. 즉, 제어를 전달하는 모든 프로그램과 엑시트 프로그램을 모두 정의할 때 EXECKEY(CICS)를 지정해야 합니다. CICS 엑시트 프로그램 및 CICS 스토리지 보호 기능에 대한 정보는 CICS 사용자 정의 안내서를 참조하십시오.
- 귀하의 프로그램은 작업 관련 사용자 종료 CICS 프로그램이 사용할 수 있는 모든 API(예: IMS, Db2, 및 CICS)를 사용할 수 있습니다. 또한 MQCONN, MQCONNX 및 MQDISC를 제외한 모든 MQI 호출을 사용할 수 있습니다. 그러나 엑시트 프로그램 안에 있는 MQI 호출은 다시 엑시트 프로그램을 호출하지 않습니다.
- 프로그램은 EXEC CICS SYNCPOINT 또는 EXEC CICS SYNCPOINT ROLLBACK 명령을 실행할 수 있습니다. 그러나 이러한 명령은 태스크에서 엑시트가 사용된 지점까지 완료된 모든 업데이트를 커밋하거나 롤백하므로 사용하지 않는 것이 좋습니다.
- 프로그램은 EXEC CICS RETURN 명령을 실행하여 종료해야 합니다. XCTL 명령으로 제어를 전송해서는 안 됩니다.
- 엑시트는 IBM MQ for z/OS 코드의 확장으로 기록됩니다. 엑시트가 MQI를 사용하는 IBM MQ for z/OS 프로그램 또는 트랜잭션을 중단시키지 않는지 확인하십시오. 이러한 트랜잭션이나 프로그램은 일반적으로 CSQ 또는 CK의 접두부로 표시됩니다.
- CSQCAPX가 CICS에 정의되면, CICS 시스템은 CICS가 IBM MQ for z/OS에 연결할 때 엑시트 프로그램을 로드하려고 시도합니다. 이 시도가 성공하면 메시지 CSQC301I가 CKQC 패널 또는 시스템 콘솔로 송신됩니다. 로드가 실패하면(예를 들어, 로드 모듈이 DFHRPL 병합된 라이브러리에 없는 경우) 메시지 CSQC315가 CKQC 패널 또는 시스템 콘솔로 송신됩니다.
- 통신 영역의 매개변수가 주소이기 때문에 엑시트 프로그램은 CICS 시스템에 로컬로 정의되어야 합니다(즉, 원격 프로그램이 아님).

샘플 API 교차 엑시트 프로그램, CSQCAPX

샘플 엑시트 프로그램은 어셈블러 언어 프로그램으로서 제공됩니다. 소스 파일(CSQCAPX)은 라이브러리 **thlqual.SCSQASMS**에서 제공됩니다. (여기서 **thlqual**은 설치 시 사용되는 상위 레벨 규정자입니다.) 이 소스 파일에는 프로그램 논리에 대해 설명하는 의사 코드(pseudocode)가 포함되어 있습니다.

샘플 프로그램에는 고유 엑시트 프로그램을 작성할 때 사용할 수 있는 레이아웃 및 초기화 코드가 포함되어 있습니다.

샘플은 다음을 수행하는 방법을 보여줍니다.

- 엑시트 매개변수 블록 설정
- 호출 및 엑시트 매개변수 블록 주소 지정
- 엑시트가 호출될 MQI 호출 판별
- MQI 호출 처리 이전 또는 이후에 엑시트가 호출되는지 여부 판별
- CICS 임시 스토리지 큐에 메시지 넣기
- 재입력 유지보수를 위한 동적 스토리지 확보에 매크로 DFHEIENT 사용
- CICS exec 인터페이스 제어 블록에 DFHEIBLK 사용
- 오류 조건 트랩
- 호출자에 대한 제어 리턴

샘플 엑시트 프로그램의 디자인

샘플 엑시트 프로그램은 CICS 임시 스토리지 큐(CSQ1EXIT)에 엑시트의 조작을 표시하는 메시지를 씁니다.

메시지는 MQI 호출 이전 또는 이후에 엑시트가 호출되는지 여부를 표시합니다. 호출 이후에 엑시트를 호출하면, 호출에 의해 리턴되는 완료 코드 및 이유 코드가 메시지에 포함됩니다. 샘플은 CMQXPA 매크로에서 이름 지정된 상수를 사용하여 입력 항목 유형(즉, 호출 이전 또는 이후)을 검사합니다.

샘플은 모니터링 기능을 수행하지 않지만, 처리 중인 호출의 유형을 표시하는 시간소인이 있는 메시지를 CICS 큐에 배치합니다. 이는 MQI 성능과 엑시트 프로그램 작동이 올바른지 여부를 표시합니다.

참고: 샘플 엑시트 프로그램은 프로그램 실행 중에 수행되는 각 MQI 호출에 대해 6가지 EXEC CICS 호출을 발행합니다. 이 엑시트 프로그램을 사용하는 경우 IBM MQ for z/OS 성능이 저하됩니다.

API 교차 엑시트 준비 및 사용

샘플 엑시트는 소스 양식으로만 제공됩니다.

샘플 종료 프로그램이나 직접 작성한 종료 프로그램을 사용하려면, 다른 CICS 프로그램과 마찬가지로 로드 라이브러리를 생성하십시오. 자세한 내용은 961 페이지의 『z/OS에서 CICS 애플리케이션 빌드』에서 설명한 대로입니다.

- Transaction CICS Server for z/OS 및 CICS 을 MVS/ESA위한 경우, 시스템 정의(CSD) CICS 데이터 세트를 업데이트할 때 필요한 정의는 멤버 **thlqual.SCSQPROC**(CSQ4B100)에 있습니다.

참고: 정의는 MQ의 접미부를 사용합니다. 이 접미부가 이미 엔터프라이즈에서 사용된 경우 어셈블리 스테이지 이전에 이를 변경해야 합니다.

제공된 기본 CICS 프로그램 정의를 사용하는 경우, 엑시트 프로그램 CSQCAPX은 **사용 안함** 상태에서 설치됩니다. 그 이유는 엑시트 프로그램을 사용하면 성능이 크게 저하될 수 있기 때문입니다.

API 교차 엑시트를 일시적으로 활성화하려면 다음을 수행하십시오.

1. 마스터 **CEMT S PROGRAM(CSQCAPX) ENABLED**CICS 터미널에서 명령을 실행하십시오.
2. CKQC 트랜잭션을 실행하고, 연결 풀다운에서 옵션 3을 사용하여 API 교차 엑시트의 상태를 **사용**으로 대체하십시오.

API-crossing exit을 영구적으로 활성화하고, Transaction Server for z/OSCICS 및 CICS for MVS/ESA를 IBM MQ for z/OS 사용하려면 다음 중 하나를 수행하십시오:

- STATUS(DISABLED)를 STATUS(ENABLED)로 변경하여 멤버 CSQ4B100의 CSQCAPX 정의를 대체하십시오. CICS 제공 배치 프로그램 DFHCSDUP를 사용하여 CICS CSD 정의를 업데이트할 수 있습니다.
- 상태를 DISABLED에서 ENABLED로 변경하여 CSQCAT1 그룹의 CSQCAPX 정의를 대체하십시오.

두 경우 모두 그룹을 재설치해야 합니다. CICS 시스템을 콜드 스타트하거나, CICS가 실행 중일 때 CICS CEDA 트랜잭션을 사용해서 그룹을 재설치하는 방법으로 이 작업을 수행할 수 있습니다.

참고: CEDA를 사용하면 그룹의 모든 입력 항목이 현재 사용 중인 경우 오류가 발생할 수 있습니다.

제품별 프로그래밍 인터페이스 정보를 종료합니다.

▶ z/OS 공유 큐를 사용한 애플리케이션 프로그래밍

이 주제에서는 공유 큐를 사용하도록 새 애플리케이션을 설계하고 기존 애플리케이션을 공유 큐 환경으로 마이그레이션할 때 고려해야 할 몇 가지 요인에 대한 정보를 제공합니다.

▶ z/OS 애플리케이션 직렬화

특정 유형의 애플리케이션은 메시지가 큐에 도착한 순서와 정확히 동일한 순서로 큐에서 검색되어야 할 수도 있습니다.

예를 들어, IBM MQ를 원격 시스템에 대해 데이터베이스 업데이트를 새도우 처리하는 데 사용할 경우 레코드의 삽입에 대해 설명하는 메시지 다음에 해당 레코드에 대한 업데이트를 설명하는 메시지를 처리해야 합니다. 로컬 큐잉 환경에서는 메시지를 가져와서 MQOO_INPUT_EXCLUSIVE 옵션으로 큐를 여는 애플리케이션을 통해 이 작업이 수행되기도 하는데, 이 경우 다른 가져오기 애플리케이션이 동시에 큐를 처리하지 못하도록 합니다.

IBM MQ는 애플리케이션이 같은 방법으로 공유 큐를 독점하여 열 수 있게 허용합니다. 그러나 애플리케이션이 큐의 파티션에서 작동 중이고(예를 들어, 모든 데이터베이스 업데이트는 동일한 큐에 있지만, 테이블 A에 대한 업데이트의 상관 ID는 A이고 테이블 B에 대한 업데이트의 상관 ID는 B임) 여러 애플리케이션이 테이블 A 업데이트 및 테이블 B 업데이트에 대한 메시지를 동시에 가져올 경우, 독점적으로 큐를 여는 단순 메커니즘은 불가능합니다.

이러한 유형의 애플리케이션이 고가용성의 공유 큐를 이용할 경우, 1차 가져오기 애플리케이션 또는 큐 관리자가 실패하면 동일한 공유 큐에 액세스하고 2차 큐 관리자에서 실행 중인 애플리케이션의 다른 인스턴스가 인계받도록 결정할 수 있습니다.

1차 큐 관리자가 실패하면 다음 두 가지 결과가 발생합니다.

- 공유 큐 피어 복구가 1차 애플리케이션에서 완료되지 않은 업데이트가 완료되거나 백아웃되었는지 확인합니다.
- 2차 애플리케이션이 큐 처리를 인계받습니다.

완료되지 않은 모든 작업 단위가 처리되기 전에 2차 애플리케이션이 시작될 수 있고, 이를 통해 2차 애플리케이션이 순서 상관없이 메시지를 검색할 수 있습니다. 이런 유형의 문제점을 해결하기 위해 애플리케이션은 직렬화된 애플리케이션이 될 수 있습니다.

직렬화된 애플리케이션은 MQCONNX 호출을 사용하여 큐 관리자에 연결하며, 연결 시 해당 애플리케이션에 고유한 연결 태그를 지정합니다. 애플리케이션에 의해 수행된 모든 작업 단위에는 연결 태그가 표시됩니다. IBM MQ는 큐 공유 그룹 내에서 동일한 연결 태그가 있는 작업 단위가 (MQCONNX 호출의 직렬화 옵션에 따라) 직렬화되었는지 확인합니다.

이는 주 애플리케이션이 연결 태그로 MQCONNX 호출을 사용하는 경우, 보조 인수 애플리케이션이 동일한 연결 태그로 Database shadow retriever MQCONNX 호출을 시도하면 보조 애플리케이션이 주 작업 단위가 완료될 때까지(이 경우 피어 IBM MQ 복구를 통해) 두 번째에 연결할 수 없음을 의미합니다.

큐의 정확한 메시지 순서를 사용하는 애플리케이션에 대해서는 직렬화 애플리케이션 기술을 사용해 보십시오. 특히, 다음과 같습니다.

- 애플리케이션 또는 큐 관리자 실패 후 애플리케이션에 대해 이전에 실행된 모든 커밋 및 백아웃 조작이 완료될 때까지 재시작해서는 안 되는 애플리케이션

이 사례에서 애플리케이션이 동기점으로 작동하는 경우에만 직렬화 애플리케이션 기술이 적용 가능합니다.

- 동일한 애플리케이션의 다른 인스턴스가 이미 실행 중일 때 시작해서는 안 되는 애플리케이션

이 사례에서 애플리케이션이 독점 입력 큐를 열 수 없는 경우에만 직렬화 애플리케이션 기술이 필요합니다.

참고: IBM MQ는 특정 기준이 충족될 때만 메시지 순서를 보존하도록 보장합니다. 이에 대해서는 [MQGET](#)에서 설명합니다.

▶ z/OS 공유 큐에서 사용하기에 적당하지 않은 애플리케이션

공유 큐를 사용하는 경우 IBM MQ의 일부 기능이 지원되지 않으므로 이러한 기능을 사용하는 애플리케이션은 공유 큐 환경에 적합하지 않습니다.

공유 큐 애플리케이션을 설계할 때 고려해야 할 사항은 다음과 같습니다.

- 공유 큐에 대해서는 큐 색인화를 제한합니다. 메시지 ID 또는 상관 ID를 사용하여 큐에서 가져올 메시지를 선택하려면 올바른 값으로 큐를 색인화해야 합니다. 메시지 ID로만 메시지를 선택할 경우 (MQIT_NONE을 사용할 수도 있지만) 큐의 색인 유형은 MQIT_MSG_ID여야 합니다. 상관 ID로만 메시지를 선택할 경우 큐의 색인 유형은 MQIT_CORREL_ID여야 합니다.
- 임시 동적 큐는 공유 큐로 사용할 수 없습니다. 하지만 영구적 동적 큐는 사용할 수 있습니다. 공유 동적 큐 모델은 PERMDYN(영구적 동적) 큐와 같은 방식으로 작성되고 영구 삭제되지만 DEFTYPE이 SHAREDYN(공유 동적)입니다.

z/OS 비애플리케이션 큐 공유 여부 결정
비애플리케이션 큐 공유를 고려할 때 이 정보를 사용합니다.

공유하는 것을 고려해볼 수 있는 애플리케이션 큐 이외의 큐가 있습니다.

이니시에이션 큐

공유 이니시에이션 큐를 정의하는 경우, 실행 중인 트리거 모니터가 하나 이상 있다면 공유 그룹의 모든 큐 관리자에서 실행 중인 트리거 모니터가 없어도 됩니다. (큐 공유 그룹의 각 큐 관리자에서 실행 중인 트리거 모니터가 있는 경우에도 공유 이니시에이션 큐를 사용할 수 있습니다.)

공유 애플리케이션 큐가 있고 트리거 유형 EVERY(또는 트리거 유형 EVERY 같이 작동하며 트리거 간격이 작은 트리거 유형 FIRST)를 사용하는 경우, 사용자의 이니시에이션 큐는 항상 공유 큐여야 합니다. 공유 이니시에이션 큐를 사용할 시기에 대한 자세한 정보는 [904 페이지의 표 146](#)의 내용을 참조하십시오.

SYSTEM.* 큐

SYSTEM.ADMIN 을 정의할 수 있습니다.* 이벤트 메시지를 공유 큐로 보관하는 데 사용되는 큐들. 이는 예외가 발생할 경우 로드 밸런싱을 확인하는 데 유용할 수 있습니다. IBM MQ가 작성한 각 이벤트 메시지에는 해당 메시지를 생성한 큐 관리자를 나타내는 상관 ID가 포함되어 있습니다.

SYSTEM.QSG 를 반드시 정의해야 합니다.* 공유 채널 및 그룹 내 대기열에 사용되는 대기열을 공유 대기열로 지정합니다.

또한 공유할 SYSTEM.DEFAULT.LOCAL.QUEUE의 정의를 변경하거나 사용자 자체의 기본 공유 큐 정의를 정의할 수 있습니다. 자세한 내용은 [시스템 객체 정의하기](#)를 IBM MQ for z/OS 참조하십시오.

다른 SYSTEM.*을 정의할 수 없습니다 큐를 공유 큐로.

z/OS 공유 큐를 사용할 기존 애플리케이션 마이그레이션
이유 코드, 트리거 및 MQINQ API 호출은 공유 큐 환경에서 다르게 작동할 수 있습니다.

기존 큐를 공유 큐로 마이그레이션하는 방법에 대한 정보는 '비공유 큐를 공유 큐로 마이그레이션하기'를 참조하십시오.

기존 애플리케이션을 마이그레이션할 경우에는 공유 큐 환경에서 다른 방식으로 작동할 수 있으므로 다음 사항을 고려하십시오.

이유 코드

공유 큐를 사용하기 위해 기존 애플리케이션을 마이그레이션할 때 발행될 수 있는 새 이유 코드를 확인하십시오.

트리거

공유 애플리케이션 큐를 사용할 경우 커밋된 메시지에서만 트리거가 작동합니다. (비공유 애플리케이션 큐에서는 모든 메시지에 대해 트리거가 작동합니다.)

트리거를 사용하여 애플리케이션을 시작하는 경우 공유 이니시에이션 큐를 사용할 수도 있습니다. [904 페이지의 표 146](#)에서는 사용할 이니시에이션 큐의 유형을 결정할 때 고려해야 할 사항에 대해 설명합니다.

표 146. 공유 이니시에이션 큐 사용 시기		
	비공유 애플리케이션 큐	공유 애플리케이션 큐
비공유 이니시에이션 큐	이전 릴리스의 경우	FIRST 또는 DEPTH의 트리거 유형을 사용하는 경우 비공유 이니시에이션 큐를 공유 애플리케이션 큐로 사용할 수 있습니다. 추가 트리거 메시지가 생성될 수 있지만, 이 설정은 장기 실행 애플리케이션(예: CICS bridge)을 트리거 하기에 적합하며 고가용성을 제공합니다. 트리거 유형 FIRST 또는 DEPTH의 경우, 트리거 메시지는 트리거 모니터를 실행 중이고 입력을 위해 애플리케이션 큐가 아직 열리지 않은 모든 큐 관리자에서 애플리케이션의 인스턴스를 트리거합니다. 모든 큐 관리자에 대해 하나의 트리거 메시지가 생성됩니다. 특정 큐 관리자에서 비공유 로컬 이니시에이션 큐에 대해 둘 이상의 트리거 모니터가 실행 중인 경우 이들 트리거 모니터는 메시지 처리를 위해 경쟁합니다.
공유 이니시에이션 큐	비공유 애플리케이션 큐와 함께 공유 이니시에이션 큐를 사용하지 마십시오.	트리거 유형 EVERY의 경우, 애플리케이션이 공유 애플리케이션 큐에 메시지를 넣을 때, 메시지를 넣는 큐 관리자가 트리거-에브리 이벤트에 관심이 있는 큐 관리자를 확인하고 해당 큐 관리자 중 하나에 알림을 보냅니다. 통지된 큐 관리자에 대해, 결과적인 작업은 시작 큐로 트리거 메시지를 생성하는 것입니다 참고: 트리거 유형이 EVERY인 공유 애플리케이션 큐가 있는 경우 공유 이니시에이션 큐를 사용하십시오. 그렇지 않으면 특정 상황(예: 큐 관리자 실패)에서 트리거 메시지가 손실될 수 있습니다. 트리거 유형 FIRST 또는 DEPTH의 경우, 입력을 위해 이름 지정된 이니시에이션 큐가 열려 있는 각 큐 관리자에서 하나의 트리거 메시지가 생성됩니다. 참고: 트리거 유형 FIRST 또는 DEPTH의 경우, 하나의 트리거 모니터 인스턴스가 사용 중이면 사용이 많지 않은 트리거 모니터가 공유 이니시에이션 큐에서 둘 이상의 트리거 메시지를 처리할 가능성이 있습니다. 따라서 지정된 큐 관리자에 대하여 서버 애플리케이션의 여러 인스턴스를 시작할 수 있습니다. 이러한 다중 인스턴스는 다중 트리거 메시지 처리의 결과로 시작된다는 점에 유의하십시오. 보통은 트리거 유형 FIRST 또는 DEPTH의 경우, 애플리케이션 인스턴스가 이미 애플리케이션 큐를 제공하고 있으면 애플리케이션이 연결된 큐 관리자에서 다른 트리거 메시지가 생성되지 않습니다.

MQINQ

MQINQ 호출을 사용하여 공유 큐에 대한 정보를 표시하는 경우, 입력 및 출력을 위해 큐가 열려 있는 MQOPEN 호출 수의 값은 호출을 발행한 큐 관리자에만 관련됩니다. 큐 공유 그룹에서 큐가 열려 있는 다른 큐 관리자에 대한 정보는 생성되지 않습니다.

z/OS IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션

이 정보는 IBM MQ(를) 사용하여 IMS 애플리케이션을 작성하는 데 도움이 됩니다.

- IMS 애플리케이션에서 동기점 및 MQI 호출을 사용하려면, [71 페이지의 『사용하여 애플리케이션 IMS 작성 IBM MQ』](#)의 내용을 참조하십시오.
- IBM MQ - IMS 브릿지를 사용하는 애플리케이션을 쓰려면, [75 페이지의 『IMS 브릿지 애플리케이션 작성』](#)의 내용을 참조하십시오.

다음 링크를 사용하여 IBM MQ for z/OS의 IMS 및 IMS 브릿지 애플리케이션에 대해 자세히 알아보십시오.

- [71 페이지의 『사용하여 애플리케이션 IMS 작성 IBM MQ』](#)
- [75 페이지의 『IMS 브릿지 애플리케이션 작성』](#)

관련 개념

[744 페이지의 『MQI\(Message Queue Interface\) 개요』](#)

MQI(Message Queue Interface) 컴포넌트에 대해 학습합니다.

[756 페이지의 『큐 관리자에 연결 및 큐 관리자에서 연결 끊기』](#)

IBM MQ 프로그래밍 서비스를 사용하려면 프로그램을 큐 관리자에 연결해야 합니다. 이 정보를 사용하여 큐 관리자에 연결하고 큐 관리자에서 연결을 끊는 방법에 대해 알아보십시오.

[763 페이지의 『오브젝트 열기 및 닫기』](#)

이 정보는 IBM MQ 오브젝트의 열기 및 닫기 조작에 대한 통찰력을 제공합니다.

[772 페이지의 『큐에 메시지 넣기』](#)

이 정보를 사용하여 메시지를 큐에 넣는 방법에 대해 알아보십시오.

[786 페이지의 『큐에서 메시지 가져오기』](#)

이 정보를 사용하여 큐에서 메시지를 가져오는 방법에 대해 알아봅니다.

[861 페이지의 『오브젝트 속성 조회 및 설정』](#)

속성은 IBM MQ 오브젝트의 특성을 정의하는 특성입니다.

[863 페이지의 『작업 단위 커밋 및 백아웃』](#)

이 정보는 작업 단위에서 발생한 복구 가능한 가져오기 및 넣기 조작을 커밋하고 백아웃하는 방법을 설명합니다.

[873 페이지의 『트리거를 사용한 IBM MQ 애플리케이션 시작』](#)

트리거와 트리거를 사용하여 IBM MQ 애플리케이션을 시작하는 방법에 대해 학습하십시오.

[891 페이지의 『MQI 및 클러스터에 대한 작업』](#)

클러스터와 관련된 호출 및 리턴 코드에 대한 특수 옵션이 있습니다.

[895 페이지의 『IBM MQ for z/OS의 애플리케이션 사용 및 작성』](#)

IBM MQ for z/OS 애플리케이션은 여러 다른 환경에서 실행되는 프로그램으로 구성할 수 있습니다. 이는 둘 이상의 환경에서 사용 가능한 기능을 활용할 수 있음을 의미합니다.

사용하여 애플리케이션 IMS 작성 IBM MQ

애플리케이션에서 IMS를 IBM MQ 사용할 때 고려해야 할 추가 사항이 있습니다. 여기에는 어떤 MQ API 호출을 사용할 수 있는지, 그리고 동기화 지점(syncpoint)을 위한 메커니즘이 포함됩니다.

다음 링크를 통해 애플리케이션 IMS 작성에 대해 IBM MQ for z/OS 자세히 알아보세요:

- [72 페이지의 『IMS 애플리케이션의 동기점』](#)
- [72 페이지의 『IMS 애플리케이션에서 MQI 호출』](#)

제한사항

어댑터를 IMS 사용하는 애플리케이션이 호출할 수 있는 IBM MQ API에는 제한 사항이 있습니다.

다음 IBM MQ API 호출은 IMS 어댑터를 사용하는 애플리케이션 내에서 지원되지 않습니다.

- MQCB
- MQCB_FUNCTION
- MQCTL

관련 개념

[75 페이지의 『IMS 브릿지 애플리케이션 작성』](#)

이 주제에는 IBM MQ - IMS 브릿지를 사용하기 위해 애플리케이션을 작성하는 데 대한 정보가 포함되어 있습니다.

▶ z/OS IMS 애플리케이션의 동기점

IMS 애플리케이션에서 GU(get unique)부터 IOPCB 및 CHKP(checkpoint)와 같은 IMS 호출을 사용하여 동기점을 설정합니다.

이전 체크포인트 이후로 모든 변경을 백아웃하려면 IMS ROLB(롤백) 호출을 사용할 수 있습니다. 자세한 내용은 문서의 [IMS ROLB 호출](#)을 참조하십시오.

큐 관리자는 2단계 커밋 프로토콜의 참가자입니다. IMS 동기점 관리자는 코디네이터입니다.

열려 있는 모든 핸들은 (배치 또는 비메시지 구동 BMP 환경을 제외한) 동기점에서 IMS 어댑터에 의해 처리완료됩니다. 이는 다른 사용자가 다음 작업 단위를 시작할 수 있고 MQPUT 또는 MQGET 호출 시가 아니라 MQCONN, MQCONNX 및 MQOPEN 호출 시에 IBM MQ 보안 검사가 수행되기 때문입니다.

그러나 입력 대기(WFI) 또는 의사 입력 대기(PWFI) 환경에서 IMS는 다음 메시지가 도착하거나 QC 상태 코드가 애플리케이션으로 리턴될 때까지 핸들을 닫도록 IBM MQ에 알리지 않습니다. 애플리케이션이 IMS 리전에서 대기 중이고 이러한 핸들 중 하나라도 트리거된 큐에 속하면 해당 큐가 열려 있기 때문에 트리거가 발생하지 않습니다. 이러한 이유로 WFI 또는 PWFI 환경에서 실행 중인 애플리케이션은 다음 메시지에 대해 GU부터 IOPCB를 수행하기 전에 큐 핸들을 명확하게 MQCLOSE해야 합니다.

IMS 애플리케이션(BMP 또는 MPP)이 MQDISC 호출을 발행하면, 열린 큐는 처리완료되지만 암시적 동기점이 사용되지 않습니다. 애플리케이션이 정상적으로 종료되면 열린 큐는 모두 닫히고 암시적 커밋이 발생합니다. 애플리케이션이 비정상적으로 종료되는 경우에는 열린 큐가 모두 닫히고 암시적 백아웃이 발생합니다.

▶ z/OS IMS 애플리케이션에서 MQI 호출

이 정보를 사용하여 서버 애플리케이션 및 조회 애플리케이션에서 MQI 호출의 사용에 대해 알아볼 수 있습니다.

이 절에서는 다음 유형의 IMS 애플리케이션에서 MQI 호출의 사용에 대해 설명합니다.

- 906 페이지의 『서버 애플리케이션』
- 908 페이지의 『조회 애플리케이션』

서버 애플리케이션

다음은 MQI 서버 애플리케이션 모델에 대한 개요입니다.

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

샘플 프로그램 CSQ4ICB3은 C/370에서 이 모델을 사용하는 BMP의 구현을 표시합니다. 프로그램은 먼저 IMS 그리고 IBM MQ와의 통신을 설정합니다.

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

```

Call EndMQM
End-if
End-if

Return

```

IMS 초기화는 프로그램이 메시지 구동 또는 배치 지향 BMP로 호출되었는지 판별하고 IBM MQ 큐 관리자 연결 및 큐 핸들을 적절하게 제어합니다.

```

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

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

Return to calling function

```

IBM MQ 초기화는 큐 관리자에 연결하여 큐를 엽니다. 메시지 구동 BMP에서 각 IMS 동기점이 사용된 후에 초기화가 호출됩니다. 배치 지향 BMP의 경우 프로그램 시동 중에만 호출됩니다.

```

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

Return to calling function

```

MPP에서 서버 모델의 구현은 MPP에서 호출당 단일 작업 단위를 처리한다는 사실에 의해 영향을 받습니다. 그 이유는 동기점(GU)이 사용되면 연결 및 큐 핸들이 닫히고 다음 IMS 메시지가 전달되기 때문입니다. 이 제한사항은 다음 중 하나로 부분적으로 해결할 수 있습니다.

- 단일 작업 단위 내에서 여러 메시지 처리

다음에 포함됩니다.

- 메시지 읽기

- 필수 업데이트 처리
- 응답 넣기

모든 메시지가 처리되거나 메시지 최대 세트 수가 처리될 때까지 동기점이 사용되는 시간에 반복됩니다.

애플리케이션의 특정 유형만(예를 들어, 단순 데이터베이스 업데이트 또는 조회) 이 방법으로 접근할 수 있습니다. MQI 응답 메시지를 핸들링되는 MQI 메시지의 진원지 권한을 사용하여 넣을 수 있더라도 IMS 자원 업데이트의 보안 관련 내용을 주의깊게 고려해야 합니다.

- **MPP의 호출당 하나의 메시지를 처리하고 사용 가능한 모든 메시지를 처리하기 위해 MPP의 다중 스케줄링을 보장합니다.**

IBM MQ IMS 트리거 모니터 프로그램(CSQQTRMN)을 사용하여 IBM MQ 큐에 메시지가 있고 이를 제공하는 애플리케이션이 없는 경우 MPP 트랜잭션을 스케줄링하십시오.

트리거 모니터가 MPP를 시작하는 경우 다음 COBOL 코드 추출에 표시된 대로 큐 관리자 이름 및 큐 이름이 프로그램에 전달됩니다.

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

CSQQTRMN을 사용하여 BMP를 트리거할 수 없더라도 장기 실행 태스크가 예상되는 서버 모델은 배치 처리 지역에서 제대로 지원됩니다.

조회 애플리케이션

조회 또는 업데이트를 시작하는 일반적인 IBM MQ 애플리케이션은 다음과 같이 작동합니다.

- 사용자로부터 데이터 수집
- 하나 이상의 IBM MQ 메시지 넣기
- 응답 메시지 가져오기(대기해야 할 수 있음)
- 사용자에게 응답 제공

IBM MQ 큐로 넣어지는 메시지는 커밋될 때까지 다른 IBM MQ 애플리케이션에서 사용할 수 없기 때문에 동기점 밖으로 넣거나 IMS 애플리케이션을 두 개의 트랜잭션으로 분할해야 합니다.

조회에 단일 메시지 넣기가 포함된 경우 동기점 없음 옵션을 사용할 수 있습니다. 그러나 조회가 좀 더 복잡하거나 자원 업데이트가 관련된 경우에는 실패가 발생하거나 동기점 조정을 사용하지 않으면 일관성 문제점이 발생할 수 있습니다.

이 문제를 해결하려면 프로그램 간 메시지 스위치를 사용하여 MQI MPP 트랜잭션을 호출로 분할할 IMS 수 있습니다. 이에 대한 자세한 내용은 [여기]를 참조하십시오 [IMS 시스템 간 통신 \(ISC\)](#). 조회 프로그램이 MPP에서 구현될 수 있습니다.

```
Initialize first program/Connect
.
Open queue for output
.
```

```

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

```

▶ z/OS IMS 브릿지 애플리케이션 작성

이 주제에는 IBM MQ - IMS 브릿지를 사용하기 위해 애플리케이션을 작성하는 데 대한 정보가 포함되어 있습니다.

IBM MQ - IMS 브릿지에 대한 정보는 [IMS 브릿지](#)를 참조하십시오.

다음 링크를 사용하여 IBM MQ for z/OS에서 IMS 브릿지 애플리케이션 작성에 대해 자세히 알아보십시오.

- [75 페이지의 『IMS 브릿지에서 메시지를 처리하는 방법』](#)
- [915 페이지의 『트랜잭션 프로그램을 IMS 작성하여 IBM MQ』](#)

관련 개념

[71 페이지의 『사용하여 애플리케이션 IMS 작성 IBM MQ』](#)

애플리케이션에서 IMS 를 IBM MQ 사용할 때 고려해야 할 추가 사항이 있습니다. 여기에는 어떤 MQ API 호출을 사용할 수 있는지, 그리고 동기화 지점(syncpoint)을 위한 메커니즘이 포함됩니다.

▶ z/OS IMS 브릿지에서 메시지를 처리하는 방법

IBM MQ - IMS 브릿지를 사용하여 IMS 애플리케이션으로 메시지를 전송하는 경우 메시지를 특별한 형식으로 생성해야 합니다.

또한 대상 IMS 시스템의 XCF 그룹 및 멤버 이름을 지정하는 저장 클래스로 정의된 큐에 IBM MQ 메시지를 배치해야 합니다. MQ-IMS 브릿지 큐 또는 단순히 **브릿지** 큐로 알려져 있습니다.

IBM MQ-IMS 브릿지는 QSGDISP(QMGR)로 정의된 경우 또는 QSGDISP(SHARED)를 NOSHARE 옵션과 함께 사용하여 정의된 경우 브릿지 큐에 대한 독점 입력 액세스(MQOO_INPUT_EXCLUSIVE)가 필요합니다.

사용자는 메시지를 IMS 애플리케이션에 보내기 전에 IMS에 사인온할 필요가 없습니다. MQMD 구조의 *UserIdentifier* 필드에서 사용자 ID는 보안 검사에 사용됩니다. 검사 레벨은 IBM MQ가 IMS에 연결할 때 판별되고 [IMS 브릿지에 대한 애플리케이션 액세스 제어](#)에서 설명됩니다. 의사 사인온을 구현되도록 할 수 있습니다.

IBM MQ - IMS 브릿지는 다음 유형의 메시지를 허용합니다.

- IMS 트랜잭션 데이터 및 MQIIH 구조를 포함하는 메시지(MQIIH에서 설명됨):

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

참고:

1. 각괄호 []는 선택적 다중 세그먼트를 나타냅니다.
 2. MQMD 구조의 *Format* 필드를 MQFMT_IMS로 설정하여 MQIIH 구조를 사용합니다.
- IMS 트랜잭션 데이터를 포함하지만 MQIIH 구조는 없는 메시지:

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

IBM MQ는 메시지 데이터의 유효성을 검증하여 LL 바이트의 합에 MQIIH(있는 경우)의 길이를 더하면 메시지 길이와 동일한지 확인합니다.

IBM MQ - IMS 브릿지가 브릿지 큐에서 메시지를 가져오는 경우 다음과 같이 메시지를 처리합니다.

- 메시지에 MQIIH 구조가 포함되어 있는 경우 브릿지는 MQIIH를 확인하고(MQIIH 참조) OTMA 헤더를 빌드하고 메시지를 IMS에 송신합니다. 트랜잭션 코드가 입력 메시지에 지정됩니다. 코드가 LTERM인 경우 IMS는 DFS1288E 메시지로 응답합니다. 트랜잭션 코드가 명령을 나타내는 경우 IMS는 명령을 실행합니다. 그렇지 않은 경우 메시지는 트랜잭션을 위해 IMS의 큐에 넣어집니다.
- 메시지에 IMS 트랜잭션 데이터가 포함되어 있지만 MQIIH 구조는 없는 경우 IMS 브릿지는 다음을 가정합니다.
 - 트랜잭션 코드는 5-12바이트의 사용자 데이터입니다.
 - 트랜잭션은 비대화식 모드입니다.
 - 트랜잭션은 커밋 모드 0(commit-then-send)입니다.
 - MQMD에서 *Format*은 *MFSMapName*(입력에서)으로 사용됩니다.
 - 보안 모드는 MQISS_CHECK입니다.

응답 메시지도 MQIIH 구조체 없이 생성되며, MQMD의 값은 *Format* 출력 IMS 메시지의 *MFSMapName* MQMD에서 가져옵니다.

IBM MQ - IMS 브릿지는 각 IBM MQ 큐에 대해 하나 이상의 Tpipe를 사용합니다.

- 동기화된 Tpipe는 커밋 모드 0(COMMIT_THEN_SEND)을 사용하는 모든 메시지에 사용됩니다(IMS /DIS TMEMBER client TPIPE xxxx 명령의 상태 필드에서 SYN과 함께 표시).
- 비동기화된 Tpipe는 커밋 모드 1(SEND_THEN_COMMIT)을 사용하는 모든 메시지에 사용됩니다.

Tpipe는 처음 사용될 때 IBM MQ에 의해 작성됩니다. 비동기화된 Tpipe는 IMS가 재시작될 때까지 존재합니다. 동기화된 Tpipe는 IMS가 콜드 스타트될 때까지 존재합니다. 이 Tpipe를 사용자가 삭제할 수 없습니다.

IBM MQ - IMS 브릿지에서 메시지를 처리하는 방법에 대한 자세한 정보는 다음 주제를 참조하십시오.

- 77 페이지의 『IBM MQ 메시지를 IMS 트랜잭션 유형에 매핑』
- 77 페이지의 『메시지를 IMS 큐에 넣을 수 없는 경우』
- 77 페이지의 『IMS 브릿지 피드백 코드』
- 78 페이지의 『IMS 브릿지의 메시지에 있는 MQMD 필드』
- 79 페이지의 『IMS 브릿지의 메시지에 있는 MQIIH 필드』
- 80 페이지의 『IMS의 응답 메시지』
- 80 페이지의 『IMS 트랜잭션에서 대체 응답 PCB 사용』
- 80 페이지의 『IMS의 요청되지 않은 메시지 송신』
- 80 페이지의 『메시지 세그먼트화』
- 81 페이지의 『교량과의 통신 IMS 메시지 데이터 변환』

관련 개념

915 페이지의 『트랜잭션 프로그램을 IMS 작성하여 IBM MQ』

트랜잭션을 IMS 처리하기 위해 필요한 코딩은 해당 IMS 트랜잭션이 요구하는 IBM MQ 메시지 형식과 반환 가능한 응답 범위에 따라 달라집니다. 그러나 애플리케이션이 IMS 화면 형식 정보를 핸들링할 때 몇 가지 고려할 사항이 있습니다.

 **z/OS** IBM MQ 메시지를 IMS 트랜잭션 유형에 매핑
메시지와 IBM MQ 트랜잭션 IMS 유형 간의 매핑을 설명하는 테이블.

표 147. 메시지가 IBM MQ 트랜잭션 IMS 유형에 어떻게 매핑되는가		
IBM MQ 메시지 유형	Commit-then-send(모드 0) - 동기화된 IMS Tpipe 사용	Send-then-commit(모드 1) - 동기화되지 않은 IMS Tpipe 사용
지속적 IBM MQ 메시지	- 복구 가능한 전체 기능 트랜잭션 - 복구 불가능한 트랜잭션은 IMS에 의해 거부됩니다.	- 빠른 경로 트랜잭션 - 대화식 트랜잭션 - 전체 기능 트랜잭션
비지속적 IBM MQ 메시지	- 복구할 수 없는 전체 기능 트랜잭션 - 복구 가능한 트랜잭션은 IMS V8, APAR PQ61404 및 IMS의 모든 이후 버전에서 허용됩니다.	- 빠른 경로 트랜잭션 - 대화식 트랜잭션 - 전체 기능 트랜잭션

참고: IMS 명령은 커밋 모드 0의 지속적 IBM MQ 메시지를 사용할 수 없습니다. 자세한 내용은 [커밋 모드 \(commitMode\)](#) 를 참조하십시오.

z/OS 메시지를 IMS 큐에 넣을 수 없는 경우
 메시지를 IMS 큐에 넣을 수 없는 경우 수행할 조치에 대해 알아보십시오.
 메시지를 IMS 큐에 넣을 수 없는 경우 IBM MQ에서 다음 조치를 수행합니다.

- 메시지가 올바르지 않기 때문에 IMS에 메시지를 넣을 수 없는 경우 메시지는 데드-레터 큐에 놓여지고 메시지가 시스템 콘솔로 송신됩니다.
- 메시지가 올바르지만 IMS에 의해 거부되고, IBM MQ에서 오류 메시지를 시스템 콘솔로 송신하는 경우 메시지는 IMS 감지 코드를 포함하며 IBM MQ 메시지가 데드-레터 큐에 놓여집니다. IMS 감지 코드가 001A인 경우 IMS는 실패의 이유가 포함되어 있는 IBM MQ 메시지를 응답 큐에 송신합니다.

참고: 앞서 나열된 환경에서 IBM MQ가 어떠한 이유로 인해 메시지를 데드-레터 큐에 넣을 수 없는 경우 메시지는 원래 IBM MQ 큐로 리턴됩니다. 오류 메시지가 시스템 콘솔에 송신되고 해당 큐로부터 더 이상 추가 메시지는 송신되지 않습니다.

메시지를 다시 송신하려면 다음 중 **하나**를 수행하십시오.

- 큐에 해당하는 IMS에서 Tpipe를 중지하고 다시 시작합니다.
- 큐를 GET(DISABLED)로 변경한 후 다시 GET(ENABLED)로 변경합니다.
- IMS 또는 OTMA를 중지하고 다시 시작합니다.
- IBM MQ 서브시스템을 중지하고 다시 시작합니다.
- 메시지 오류 이외의 다른 이유로 IMS에서 메시지를 거부하는 경우 IBM MQ 메시지가 원래 큐로 리턴되고 IBM MQ는 큐 처리를 중지하고 오류 메시지가 시스템 콘솔에 송신됩니다.

예외 보고 메시지가 필요한 경우 브릿지는 진원지 권한을 사용하여 응답 대상 큐에 메시지를 넣습니다. 메시지를 큐에 넣을 수 없는 경우에는 브릿지 권한을 사용하여 데드-레터 큐에 보고 메시지를 넣습니다. DLQ에 넣을 수 없다면 제거됩니다.

z/OS IMS 브릿지 피드백 코드

IMS 감지 코드는 일반적으로 CSQ2001I와 같은 IBM MQ 콘솔 메시지(예: 감지 코드 0x001F)에서 16진 형식의 출력입니다. 데드-레터 큐로 넣기되는 메시지의 데드 레터 헤더에 표시된 대로 IBM MQ 피드백 코드는 10진수입니다.

IMS 브릿지 피드백 코드는 NACK 감지 코드 0x001A에 대해 301에서 399 또는 600에서 855 범위에 있습니다. 다음과 같이 IMS-OTMA 감지 코드에서 매핑됩니다.

1. IMS-OTMA 감지 코드는 16진수에서 10진수로 변환됩니다.
2. 1번 계산 결과에 300을 더하여 코드를 IBM MQ *Feedback* 얻습니다.
3. IMS-OTMA 감지 코드 0x001A, 소수점 26은 특수한 경우입니다. 600-855 범위의 *Feedback* 코드가 생성됩니다.

- a. IMS-OTMA 이유 코드가 16진수에서 10진수로 변환됩니다.
- b. a 의 계산 결과에 600을 더하여 피드백 IBM MQ 코드를 얻습니다.

IMS-OTMA 감지 코드에 대한 정보는 [NAK 메시지의 OTMA 감지 코드](#)를 참조하십시오.

▶ z/OS IMS 브릿지의 메시지에 있는 MQMD 필드

IMS 브릿지의 메시지에 있는 MQMD 필드에 대해 알아보십시오.

원래 메시지의 MQMD는 OTMA 헤더의 사용자 데이터 섹션에 있는 IMS에 의해 이동됩니다. 메시지가 IMS에서 생성되는 경우 IMS 목적지 해석 엑시트에 의해 빌드됩니다. IMS에서 수신된 메시지의 MQMD는 다음과 같이 빌드됩니다.

StrucID

"MD"

버전

MQMD_VERSION_1

보고서

MQRO_NONE

MsgType

MQMT_REPLY

만기

MQIIH_PASS_EXPIRATION이 MQIIH의 플래그 필드에 설정되어 있는 경우 이 필드에는 남은 만기 시간이 포함됩니다. 그 밖에 MQEI_UNLIMITED로 설정됩니다.

Feedback

MQFB_NONE

Encoding

MQENC.Native(z/OS 시스템의 인코딩)

CodedCharSetId

MQCCSI_Q_MGR(z/OS 시스템의 CodedCharSetID)

형식

MQFMT_IMS(입력 메시지의 MQMD.Format이 MQFMT_IMS인 경우), 그렇지 않으면 IOPCB.MODNAME

Priority

입력 메시지의 MQMD.Priority

Persistence

커미트 모드에 따라 달라집니다. CM-1인 경우 입력 메시지의 MQMD.Persistence이며 CM-0인 경우 지속성은 IMS 메시지의 복구 가능성과 일치합니다.

MsgId

MQMD.MsgId(MQRO_PASS_MSG_ID인 경우), 그렇지 않으면 새 MsgId(기본값)

CorrelId

MQRO_PASS_CORREL_ID인 경우 입력 메시지의 MQMD.CorrelId. 그렇지 않으면 입력 메시지의 MQMD.MsgId(기본값)

BackoutCount

0

ReplyToQ

공백

ReplyToQMGr

공백(MQPUT 동안 큐 관리자에 의해 로컬 qmgr 이름으로 설정)

UserIdentifier

입력 메시지의 MQMD.UserIdentifier

AccountingToken

입력 메시지의 MQMD.AccountingToken

ApplIdentityData

입력 메시지의 MQMD.ApplIdentityData

PutApplType

오류가 없는 경우 MQAT_XCF이며 그렇지 않으면 MQAT_BRIDGE

PutApplName

오류가 없는 경우 <XCFgroupName><XCFmemberName>이며 그렇지 않으면 QMGR 이름입니다.

PutDate

메시지를 넣은 날짜

PutTime

메시지를 넣은 시간

ApplOriginData

공백

 *IMS* 브릿지의 메시지에 있는 MQIIH 필드
IMS 브릿지의 메시지에 있는 MQIIH 필드에 대해 알아보십시오.
IMS에서 수신된 메시지의 MQIIH는 다음과 같이 빌드됩니다.

StrucId

"IIH"

버전

1

StrucLength

84

Encoding

MQENC_NATIVE

CodedCharSetId

MQCCSI_Q_MGR

형식

MQIIH.ReplyToFormat이 공백이 아닌 경우 입력 메시지의 MQIIH.ReplyToFormat이며 그렇지 않으면 IOPCB.MODNAME.

플래그

0

LTermOverride

OTMA 헤더의 LTERM 이름(Tpipe)

MFSMapName

OTMA 헤더의 맵 이름

ReplyToFormat

공백

Authenticator

응답 메시지가 MQ-IMS 브릿지 큐로 놓여지는 경우 입력 메시지의 MQIIH.Authenticator이며 그렇지 않으면 공백입니다.

TranInstanceId

대화 중인 경우 OTMA 헤더의 대화 ID/서버 토큰입니다. V14 이전의 IMS 버전에서는 대화 중이 아닌 경우 이 필드가 항상 널입니다. IMS V14 이상에서는 IMS가 대화 중이 아니더라도 이 필드를 설정할 수 있습니다.

TranState

대화 중인 경우 "C"이며 그렇지 않은 경우 공백입니다.

CommitMode

OTMA 헤더의 커밋 모드("0" 또는 "1")

SecurityScope

공백

▶ z/OS IMS의 응답 메시지

IMS 트랜잭션이 IOPCB에 ISRT되면 원래 LTERM 또는 TPIPE로 메시지가 다시 라우팅됩니다.

응답 메시지로 IBM MQ에 표시됩니다. IMS로부터의 응답 메시지가 원래 메시지에 지정된 응답 대상 큐에 놓여집니다. 메시지를 응답 대상 큐에 놓을 수 없는 경우 브릿지 권한을 사용하여 데드-레터 큐에 메시지를 넣습니다. 메시지를 데드-레터 큐에 넣을 수 없는 경우에는 부정 수신확인이 IMS로 송신되어 메시지를 수신할 수 없다고 전달합니다. 메시지에 대한 책임은 IMS로 리턴됩니다. 커밋 모드 0을 사용 중인 경우 해당 Tpipe의 메시지가 브릿지로 송신되지 않으며 IMS 큐에 남게 됩니다. 즉, 다시 시작할 때까지 추가적인 메시지가 송신되지 않습니다. 커밋 모드 1을 사용 중인 경우에는 다른 작업을 계속할 수 있습니다.

응답에 MQIIH 구조가 있는 경우 해당 형식 유형은 MQFMT_IMS입니다. 그렇지 않은 경우 형식 유형은 메시지를 삽입할 때 사용된 IMS MOD 이름에 의해 지정됩니다.

▶ z/OS IMS 트랜잭션에서 대체 응답 PCB 사용

IMS 트랜잭션이 대체 응답 PCB를 사용하면(ALTPCB에 ISRTR하거나 수정 가능한 PCB에 대한 CHNG 호출을 발행) 메시지를 다시 라우팅해야 하는지 여부를 판별하기 위해 사전 라우팅 엑시트(DFSYPX0)가 호출됩니다.

메시지가 다시 라우팅되는 경우 목적지를 확인하기 위해 목적지 해석 엑시트(DFSYDRU0)가 호출되며 헤더 정보를 준비합니다. 해당 엑시트 프로그램에 대한 정보는 [IMS에서 OTMA 엑시트 사용 및 사전 라우팅 엑시트 DFSYPX0](#)를 참조하십시오.

출력에서 조치가 취해지지 않는 한, IOPCB 또는 ALTPCB로 전송되는 모든 트랜잭션 IMS IBM MQ 출력은 동일한 큐 매니저로 반환됩니다.

▶ z/OS IMS의 요청되지 않은 메시지 송신

IMS에서 IBM MQ 큐로 메시지를 전송하려면 ALTPCB에 ISRT하는 IMS 트랜잭션을 호출해야 합니다.

메시지의 MQMD를 올바르게 빌드할 수 있도록 IMS의 요청되지 않은 메시지를 라우팅하고 OTMA 사용자 데이터를 빌드하기 위해 사전 라우팅 및 목적지 해석 엑시트를 작성해야 합니다. 이러한 엑시트 프로그램에 대한 정보는 [사전 라우팅 엑시트 DFSYPX0 및 목적지 해석 엑시트의 내용을 참조하십시오](#).

참고: IBM MQ - IMS 브릿지는 수신되는 메시지가 응답인지 또는 요청되지 않은 메시지인지 알지 못합니다. 메시지와 함께 도착한 OTMA UserData를 기반으로 응답의 MQMD 및 MQIIH를 빌드하면서 각각의 경우에서 동일한 방법으로 메시지를 처리합니다.

요청되지 않은 메시지는 새 Tpipe를 작성할 수 있습니다. 예를 들어, 기존의 IMS 트랜잭션이 새 LTERM(예: PRINT01)으로 전환되었지만 구현에서 출력이 OTMA를 통해 전달되어야 하는 경우 새 Tpipe(이 예에서는 PRINT01이라고 함)가 작성됩니다. 기본적으로, 이는 비동기화된 Tpipe입니다. 구현에서 메시지가 복귀 가능해야 하는 경우 목적지 해석 엑시트 출력 플래그를 설정하십시오. 자세한 정보는 [IMS 사용자 정의 안내서](#)를 참조하십시오.

▶ z/OS 메시지 세그먼트화

단일 또는 다중 세그먼트 입력을 예상하는 것으로 IMS 트랜잭션을 정의할 수 있습니다.

원래 IBM MQ 애플리케이션은 하나 이상의 LLZZ-데이터 세그먼트로 다음 MQIIH 구조 뒤에 사용자 입력을 구성해야 합니다. IMS 메시지의 모든 세그먼트는 단일 MQPUT으로 송신된 단일 IBM MQ 메시지에 포함되어야 합니다.

LLZZ-데이터 세그먼트의 최대 길이는 IMS/OTMA(32,767바이트)에 의해 정의됩니다. 전체 IBM MQ 메시지 길이는 LL 바이트의 합에 MQIIH 구조의 길이를 더한 값입니다.

응답의 모든 세그먼트가 단일 IBM MQ 메시지에 포함됩니다.

MQFMT_IMS_VAR_STRING 형식의 메시지에 32KB 한계의 추가적인 제한사항이 있습니다. ASCII 혼합 CCSID 메시지의 데이터가 EBCDIC 혼합 CCSID 메시지로 변환되는 경우 SBCS와 DBCS 문자 간에 상태 전이가 발생할 때마다 시프트인 바이트 또는 시프트아웃 바이트가 추가됩니다. 32KB 제한사항은 메시지의 최대 크기에 적용됩니다. 즉, 메시지의 LL 필드는 32KB를 초과할 수 없기 때문에 메시지는 모든 시프트인 및 시프트아웃 문자를 포함하여 32KB를 초과하지 않아야 합니다. 메시지를 빌드하는 애플리케이션은 이를 허용해야 합니다.

교량과의 통신 IMS 메시지 데이터 변환

데이터 변환은 스토리지 클래스에 대해 정의된 XCF 정보를 가지고 있는 목적지 큐에 메시지를 넣는 경우 분배 큐잉 기능(필요한 경우 엑시트를 호출할 수 있는) 또는 인트라 그룹 큐잉 에이전트(엑시트의 사용을 지원하지 않는)에 의해 수행됩니다. 메시지가 발행/구독에 의해 큐로 전달되면 데이터 변환이 수행되지 않습니다.

필요한 엑시트의 경우 CSQXLIB DD 명령문에서 참조하는 데이터 세트의 분배 큐잉 기능에 사용 가능해야 합니다. 이는 모든 IBM MQ 플랫폼에서 - IMSIBM MQ 브릿지를 사용하여 애플리케이션에 IMS 메시지를 보낼 수 있음을 의미합니다.

변환 오류가 있는 경우 메시지는 변환되지 않은 큐에 놓입니다. 브릿지는 헤더 형식을 인식할 수 없기 때문에 결국 IBM MQ - IMS 브릿지에 의해 오류로 처리됩니다. 변환 오류가 발생하면 오류 메시지가 z/OS 콘솔로 송신됩니다.

일반적으로 데이터 변환에 대한 자세한 정보는 729 페이지의 『데이터 변환 엑시트 작성』의 내용을 참조하십시오.

IBM MQ - IMS 브릿지에 메시지 송신

변환이 올바르게 수행되었는지 확인하려면 큐 관리자에게 메시지 형식을 알려야 합니다.

메시지에 MQIIH 구조가 있는 경우 MQMD의 *Format*을 내장 형식 MQFMT_IMS로 설정해야 하며 MQIIH의 *Format*은 메시지 데이터를 설명하는 형식의 이름으로 설정되어야 합니다. MQIIH가 없는 경우 MQMD의 *Format*을 형식 이름으로 설정하십시오.

(LLZZ 이외의) 데이터가 모두 문자 데이터(MQCHAR)인 경우 내장 형식 MQFMT_IMS_VAR_STRING을 (MQIIH 또는 MQMD에서 적절한) 형식 이름으로 사용하십시오. 그렇지 않으면, 사용자 형식에 대한 데이터 변환 엑시트를 제공해야 하는 경우 사용자 고유 형식 이름을 사용하십시오. 엑시트는 데이터 자체뿐만 아니라 메시지에서 LLZZ의 변환을 핸들링해야 합니다(그러나, 메시지 시작에서 MQIIH를 핸들링할 필요는 없습니다).

애플리케이션에서 *MFSMapName*을 사용하는 경우 대신 MQFMT_IMS으로 메시지를 사용할 수 있으며 MQIIH의 *MFSMapName* 필드에서 IMS 트랜잭션에 전달된 맵 이름을 정의할 수 있습니다.

IBM MQ - IMS 브릿지에서 메시지 수신

MQIIH 구조가 IMS에 보내는 중인 원래 메시지에 있는 경우 응답 메시지에도 해당 구조가 존재합니다.

응답이 올바르게 변환되었는지 확인하려면 다음을 수행하십시오.

- 원래 메시지에 MQIIH 구조가 있는 경우 원래 메시지의 MQIIH *ReplytoFormat* 필드에서 응답 메시지에 원하는 형식을 지정하십시오. 이 값은 응답 메시지의 MQIIH *Format* 필드에 사용됩니다. 모든 출력 데이터가 LLZZ<문자 데이터> 양식인 경우 특히 유용합니다.
- 원래 메시지에 MQIIH 구조가 없는 경우 응답 메시지에 원하는 형식을 IOPCB에 대한 IMS 애플리케이션 ISRT의 MFS MOD 이름으로 지정하십시오.

트랜잭션 프로그램을 IMS 작성하여 IBM MQ

트랜잭션을 IMS 처리하기 위해 필요한 코딩은 해당 IMS 트랜잭션이 요구하는 IBM MQ 메시지 형식과 반환 가능한 응답 범위에 따라 달라집니다. 그러나 애플리케이션이 IMS 화면 형식 정보를 핸들링할 때 몇 가지 고려할 사항이 있습니다.

IMS 트랜잭션이 3270 화면에서 시작되면 IMS 메시지 형식 서비스를 통해 메시지가 전달됩니다. 그러면 트랜잭션에 의해 표시되는 데이터 스트림에서 모든 터미널 종속 항목이 제거될 수 있습니다. 트랜잭션이 OTMA를 통해 시작될 때 MFS는 관련이 없습니다. 애플리케이션 논리가 MFS에서 구현된 경우 새 애플리케이션에서 이 논리를 다시 작성해야 합니다.

일부 IMS 트랜잭션에서 일반 사용자 애플리케이션은 특정 3270 화면 작동을 수정할 수 있습니다. 예를 들면, 올바르게 읽힌 데이터가 입력된 필드를 강조표시할 수 있습니다. 프로그램으로 수정해야 할 각 화면 필드에 대해 IMS 메시지에 2바이트 속성 필드를 추가하여 이런 유형의 정보를 전달합니다.

이와 같이 3270처럼 보이도록 애플리케이션을 코딩할 경우에는 메시지를 빌드하거나 수신할 때 이러한 필드를 고려해야 합니다.

프로그램의 정보를 코딩하여 다음을 처리해야 할 수도 있습니다.

- 누른 키(예: Enter 및 PF1)
- 메시지가 애플리케이션에 전달되었을 때의 커서 위치
- IMS 애플리케이션이 속성 필드를 설정했는지 여부
 - 높음, 보통 또는 영(0)의 강도
 - 색상
 - 다음에 Enter를 누를 때 IMS가 필드 복귀를 예상하는지 여부
- IMS 애플리케이션이 모든 필드에서 널 문자(X'3F')를 사용했는지 여부

IMS 메시지에 문자 데이터(LLZZ 데이터 세그먼트는 제외)만 포함되어 있고 MQIIH 구조를 사용 중인 경우, MQMD 형식을 MQFMT_IMS로 설정하고 MQIIH 형식을 MQFMT_IMS_VAR_STRING으로 설정하십시오.

IMS 메시지에 문자 데이터(LLZZ 데이터 세그먼트는 제외)만 포함되어 있고 MQIIH 구조를 사용 중이지 않은 경우, MQMD 형식을 MQFMT_IMS_VAR_STRING으로 설정하고 IMS 애플리케이션이 응답 시 MODName MQFMT_IMS_VAR_STRING을 지정하는지 확인하십시오. 문제가 발생하고(예: 사용자에게 트랜잭션을 사용할 권한이 없음) IMS이(가) 오류 메시지를 전송하는 경우 이 메시지에는 DFSMOx 양식의 MODName이 있습니다. 여기서, x는 1-5 범위의 숫자입니다. 이는 MQMD.Format 형식으로 저장됩니다.

IMS 메시지에 2진, 팩형 또는 부동 소수점 데이터(LLZZ 데이터 세그먼트는 제외)가 포함되어 있는 경우, 사용자 자체의 데이터 변환 루틴을 코딩하십시오. 화면 IMS 포매팅에 대한 정보는 응용 프로그램 프로그래밍: IMS/ESA 트랜잭션 매니저를 참조하십시오.

를 통해 트랜잭션을 IMS 처리하는 코드를 IBM MQ작성할 때 다음 사항을 고려하십시오.

- [916 페이지의 『대화형 트랜잭션을 IMS 호출하는 애플리케이션 IBM MQ 작성』](#)
- [916 페이지의 『IMS 명령을 포함하는 프로그램 작성』](#)
- [917 페이지의 『트리거』](#)

대화형 트랜잭션을 IMS 호출하는 애플리케이션 IBM MQ 작성

이 정보를 대화형 트랜잭션을 IMS 호출하는 애플리케이션을 IBM MQ 작성할 때 고려 사항의 지침으로 활용하십시오.

IMS 대화를 호출하는 애플리케이션을 작성할 때 다음을 고려하십시오.

- 애플리케이션 메시지에 MQIIH 구조를 포함하십시오.
- MQIIH의 *CommitMode*를 MQICM_SEND_THEN_COMMIT로 설정하십시오.
- 새 대화를 호출하려면 MQIIH의 *TranState*를 MQITS_NOT_IN_CONVERSATION으로 설정하십시오.
- 대화의 두 번째 및 후속 단계를 호출하려면, *TranState*를 MQITS_IN_CONVERSATION으로 설정하고 *TranInstanceId*를 대화의 이전 단계에서 리턴된 해당 필드 값으로 설정하십시오.
- 원본 메시지를 분실한 IMS 경우, *TranInstanceId*의 값을 찾는 쉬운 IMS 방법은 없습니다.
- 응용 프로그램은 에서 발신된 *TranState* 메시지의 를 확인하여 IMSIMS 트랜잭션이 대화를 종료했는지 여부를 검사해야 합니다.
- /EXIT를 사용하여 대화를 끝낼 수 있습니다. 또한 *TranInstanceId*를 인용하고 *TranState*를 MQITS_IN_CONVERSATION으로 설정하며 대화가 실행 중인 IBM MQ 큐를 사용해야 합니다.
- /HOLD 또는 /REL을 사용하여 대화를 보류하거나 해제할 수 없습니다.
- IBM MQ - IMS 브릿지를 통해 호출된 대화는 IMS가 재시작되면 종료됩니다.

IMS 명령을 포함하는 프로그램 작성

응용 프로그램은 트랜잭션 대신 LL ZZcommand 형식의 메시지를 IBM MQ 생성할 수 있으며, 여기서 *command*는 /DIS TRAN PART 또는 /DIS POOL ALL 형식이다.

대부분의 IMS 명령은 이런 방법으로 실행할 수 있습니다. 세부사항은 IMS V11 통신 및 연결을 참조하십시오. 명령 출력은 표시용 3270 터미널로 송신되는 텍스트 양식의 IBM MQ 응답 메시지로 수신됩니다.

OTMA는 출력의 구조화된 양식을 리턴하는 IMS 표시 트랜잭션 명령의 특수 양식을 구현했습니다. 정확한 형식은 *IMS V11* 통신 및 연결에 정의되어 있습니다. IBM MQ 메시지에서 이 양식을 호출하려면, 이전과 같이 메시지 데이터를 빌드하고(예: /DIS TRAN PART) MQIIH의 TranState를 MQITS_ARCHITECTED로 설정하십시오. IMS는 명령을 처리하고 구조화된 양식으로 응답을 리턴합니다. 구조화된 응답에는 출력의 텍스트 양식에서 찾을 수 있는 모든 정보와 한 가지 추가 정보(즉, 트랜잭션이 복구 가능 또는 복구 불가능으로 정의되었는지 여부)가 포함되어 있습니다.

트리거

IBM MQ - IMS 브릿지는 트리거 메시지를 지원하지 않습니다.

XCF 매개변수에 스토리지 클래스를 사용하는 이니시에이션 큐를 정의하는 경우, 해당 큐에 넣어진 메시지는 브릿지에 도착할 때 거부됩니다.

클라이언트 프로시저 애플리케이션 작성

프로시저 언어를 사용하여 IBM MQ에서 클라이언트 애플리케이션을 작성할 때 알아야 할 사항입니다.

애플리케이션은 IBM MQ 클라이언트 환경에서 빌드하고 실행할 수 있습니다. 애플리케이션을 빌드하고 사용되는 IBM MQ MQI client에 링크해야 합니다. 애플리케이션을 빌드하고 링크하는 방식은 사용되는 플랫폼 및 프로그래밍 언어에 따라 달라집니다. 클라이언트 애플리케이션을 빌드하는 방법에 대한 정보는 [922 페이지의 『IBM MQ MQI clients의 애플리케이션 빌드』](#)의 내용을 참조하십시오.

IBM MQ 애플리케이션은 특정 조건이 충족되면 코드를 변경하지 않고 전체 IBM MQ 환경 및 IBM MQ MQI client 환경 모두에서 실행할 수 있습니다. IBM MQ 클라이언트 환경에서 애플리케이션을 실행하는 방법에 대한 자세한 정보는 [923 페이지의 『IBM MQ MQI client 환경에서의 애플리케이션 실행』](#)의 내용을 참조하십시오.

메시지 큐 인터페이스(MQI)를 사용하여 IBM MQ MQI client 환경에서 실행할 애플리케이션을 작성하는 경우, IBM MQ 애플리케이션 처리가 중단되지 않도록 하기 위해 MQI 호출 중에 부과하는 몇 가지 추가 제어가 있습니다. 이러한 제어에 대한 자세한 정보는 [918 페이지의 『클라이언트 애플리케이션에서 MQI 사용』](#)의 내용을 참조하십시오.

다른 애플리케이션 유형을 클라이언트 애플리케이션으로 준비하고 실행하는 방법에 대한 정보는 다음 주제를 참조하십시오.

- [935 페이지의 『CICS 및 Tuxedo 애플리케이션 준비 및 실행』](#)
- [52 페이지의 『Microsoft Transaction Server 애플리케이션 준비 및 실행』](#)
- [937 페이지의 『IBM MQ JMS 애플리케이션 준비 및 실행』](#)

관련 개념

[6 페이지의 『애플리케이션 개발 개념』](#)

사용자는 원하는 절차적 또는 객체 지향 언어를 사용하여 IBM MQ 애플리케이션을 작성할 수 있습니다. IBM MQ 애플리케이션을 디자인하고 작성하기 전에 기본 IBM MQ 개념을 숙지하십시오.

[5 페이지의 『IBM MQ용 애플리케이션 개발』](#)

메시지를 송신하고 수신하며, 큐 관리자와 관련 자원을 관리하기 위한 애플리케이션을 개발할 수 있습니다. IBM MQ는 많은 다양한 언어와 프레임워크로 작성된 애플리케이션을 지원합니다.

[53 페이지의 『IBM MQ 애플리케이션에 대한 설계 고려사항』](#)

애플리케이션에서 사용 가능한 플랫폼과 환경을 이용할 수 있는 방법을 결정한 경우 IBM MQ에서 제공한 기능의 사용 방법을 결정해야 합니다.

[743 페이지의 『큐잉을 위한 프로시저 애플리케이션 작성』](#)

이 정보를 사용하여 큐잉 애플리케이션 작성, 큐 관리자에 연결 및 연결 끊기, 발행/구독 및 오브젝트 열기 및 닫기에 대해 알아보십시오.

[822 페이지의 『발행/구독 애플리케이션 작성』](#)

발행/구독 IBM MQ 애플리케이션 작성을 시작합니다.

[937 페이지의 『프로시저 애플리케이션 빌드』](#)

여러 프로시저 언어 중 하나로 IBM MQ 애플리케이션을 작성하고 여러 다른 플랫폼에서 애플리케이션을 실행할 수 있습니다.

[1133 페이지의 『절차에 따른 프로그램 오류 핸들링』](#)

이 정보는 호출할 때 또는 메시지를 최종 목적지에 전달할 때 애플리케이션 MQI 호출과 관련된 오류를 설명합니다.

관련 태스크

971 페이지의 『멀티플랫폼용 샘플 절차적 프로그램 사용』

이 샘플 프로시저 프로그램은 제품과 함께 전달됩니다. 샘플은 C 및 COBOL로 작성되며 MQI(Message Queue Interface)의 일반 사용을 보여줍니다.

1079 페이지의 『다음의 샘플 절차적 프로그램을 사용하여 z/OS』

IBM MQ for z/OS와 함께 전달된 샘플 프로시저 애플리케이션은 MQI(Message Queue Interface)의 일반적인 사용에 대해 설명합니다.

클라이언트 애플리케이션에서 MQI 사용

이 토픽 컬렉션에서는 MQI(message queue interface) 클라이언트 환경에서 실행는 애플리케이션 작성과 전체 IBM MQ 큐 관리자 환경에서 실행되는 IBM MQ 애플리케이션 작성 사이의 차이점을 고려합니다.

애플리케이션을 설계할 경우, IBM MQ 애플리케이션 처리가 중단되지 않도록 하기 위해 MQI 호출 중에 부과해야 할 제어를 고려하십시오.

MQI를 사용하는 애플리케이션을 실행하려면 먼저 특정 IBM MQ 오브젝트를 작성해야 합니다. 자세한 정보는 [MQI를 사용하는 애플리케이션 프로그램을 참조하십시오](#).

클라이언트 애플리케이션에서 메시지의 크기 제한

큐 관리자에는 최대 메시지 길이가 있지만, 클라이언트 애플리케이션에서 전송할 수 있는 최대 메시지 크기는 채널 정의에 의해 제한됩니다.

큐 관리자의 최대 메시지 길이(MaxMsgLength) 속성은 해당 큐 관리자가 핸들링할 수 있는 메시지의 최대 길이입니다.

 다중 플랫폼에서는 큐 관리자의 최대 메시지 길이 속성을 늘릴 수 있습니다. 자세한 정보는 [ALTER QMGR](#)을 참조하십시오.

MQINQ 호출을 사용하여 큐 관리자의 MaxMsgLength 값을 알아볼 수 있습니다.

MaxMsgLength 속성을 변경하면 새로운 값보다 긴 큐 및 메시지가 없는지 확인되지 않습니다. 이 속성을 변경한 후, 변경사항이 적용되었는지 확인하려면 애플리케이션 및 채널을 재시작하십시오. 그런 다음, (큐 관리자 세그먼트화가 허용되지 않으면) 큐 관리자 또는 큐의 MaxMsgLength를 초과하는 새 메시지를 생성할 수 없습니다.

채널 정의의 최대 메시지 길이는 클라이언트 연결을 따라 전송할 수 있는 메시지의 크기를 제한합니다. IBM MQ 애플리케이션이 이보다 큰 메시지에 MQPUT 호출 또는 MQGET 호출을 사용하려고 하면, 오류 코드가 애플리케이션으로 리턴됩니다. 채널 정의의 최대 메시지 크기 매개변수는 클라이언트 연결을 통해 MQCB를 사용하여 이용될 수 있는 최대 메시지 크기에 영향을 주지 않습니다.

관련 개념

922 페이지의 『MQCONNX 사용』

MQCONNX 호출을 사용하여 MQCNO 구조에 채널 정의(MQCD) 구조를 지정할 수 있습니다.

관련 참조

[최대 메시지 길이\(MAXMSGL\)](#)

[ALTER CHANNEL](#)

[2010 \(07DA\) \(RC2010\): MQRC_DATA_LENGTH_ERROR](#)

클라이언트 또는 서버 CCSID 선택

클라이언트에 대한 CCSID(local coded character set identifier)를 사용하십시오. 큐 관리자는 필요한 변환을 수행합니다. 환경 **MQCCSID** 변수를 사용하여 CCSID를 재정의할 수 있습니다. 애플리케이션이 PUT을 여러 개 수행할 경우, 첫 번째 PUT이 완료된 후 MQMD의 인코딩 필드 및 CCSID를 덮어쓸 수 있습니다.

애플리케이션에서 클라이언트 스텝으로 MQI(message queue interface)를 통해 전달되는 데이터는 로컬 CCSID에 있어야 하고, IBM MQ MQI client용으로 인코딩되어야 합니다. 연결된 큐 관리자에서 데이터 변환이 필요하면 큐 관리자의 클라이언트 지원 코드로 변환이 수행됩니다.

IBM WebSphere MQ 7.0 이후 버전에서 Java 클라이언트는 큐 관리자가 이를 수행할 수 없는 경우 변환을 수행할 수 있습니다. 343 페이지의 『IBM MQ classes for Java 클라이언트 연결』의 내용을 참조하십시오.

클라이언트 코드에서는 클라이언트의 MQI를 교차하는 문자 데이터가 해당 워크스테이션용으로 구성된 CCSID에 있다고 가정합니다. 이 CCSID가 지원되지 않는 CCSID이거나 요구되는 CCSID가 아닌 경우, 다음 명령어 중 하나를 사용하여 구성 IBM MQ MQI client 파일의 속성 **CCSID** 또는 환경 **MQCCSID** 변수로 재정의할 수 있습니다:

• **Windows**

```
SET MQCCSID=850
```

• **Linux** **AIX**

```
export MQCCSID=850
```

• **IBM i**

```
ADDENVVAR ENVVAR(MQCCSID) VALUE(37)
```

이 매개변수가 프로파일에 설정되면, 모든 MQI 데이터는 코드 페이지 850에 있다고 가정됩니다.

참고: 코드 페이지 850에 대한 가정은 메시지의 애플리케이션 데이터에는 적용되지 않습니다.

애플리케이션이 메시지 디스크립터(MQMD) 뒤에 IBM MQ 헤더를 포함하는 다중 PUT을 수행하는 경우 MQMD의 인코딩 필드 및 CCSID는 첫 번째 PUT이 완료된 후 덮어씌웁니다.

첫 번째 PUT 이후, 이러한 필드에는 연결된 큐 관리자에서 IBM MQ 헤더를 변환하는 데 사용된 값이 포함됩니다. 애플리케이션에서 해당 값을 필수 값으로 재설정하는지 확인하십시오.

클라이언트 애플리케이션의 MQINQ 사용

MQINQ를 사용하여 조회한 몇 가지 값은 클라이언트 코드로 수정됩니다.

CCSID

큐 관리자의 값이 아닌 클라이언트 CCSID로 설정됩니다.

MaxMsgLength

채널 정의에 의해 값이 제한될 경우에 줄어듭니다. 이 값은 다음 값 중 낮은 값입니다.

- 큐 정의에 정의된 값 또는
- 채널 정의에 정의된 값

자세한 정보는 [MQINQ](#)를 참조하십시오.

클라이언트 애플리케이션에서 동기점 조정 사용

기본 클라이언트에서 실행 중인 애플리케이션은 MQCMIT 및 MQBACK을 발행할 수 있지만 동기점 제어의 범위가 MQI 자원으로 제한됩니다. 확장 트랜잭션 클라이언트에서 외부 트랜잭션 관리자를 사용할 수 있습니다.

IBM MQ 내에서 큐 관리자의 역할 중 하나는 애플리케이션 내에서의 동기점 제어입니다. 애플리케이션은 IBM MQ 기본 클라이언트에서 실행되는 경우 MQCMIT 및 MQBACK을 발행할 수 있지만 동기점 제어의 범위가 MQI 자원으로 제한됩니다. IBM MQ verb MQBEGIN은 기본 클라이언트 환경에서 유효하지 않습니다.

서버의 전체 큐 관리자 환경에서 실행되는 애플리케이션은 트랜잭션 모니터를 통해 여러 자원(예: 데이터베이스)을 통합할 수 있습니다. 서버에서는 IBM MQ 제품과 함께 제공되는 트랜잭션 모니터 또는 다른 트랜잭션 모니터(예: CICS)를 사용할 수 있습니다. 기본 클라이언트 애플리케이션과 함께 트랜잭션 모니터를 사용할 수는 없습니다.

외부 트랜잭션 관리자는 IBM MQ 확장 트랜잭션 클라이언트에서 사용할 수 있습니다. 세부사항은 [확장 트랜잭션 클라이언트의 개념](#)을 참조하십시오.

클라이언트 애플리케이션에서의 미리 읽기 사용

클라이언트에서 미리 읽기를 사용하면 클라이언트 애플리케이션이 메시지를 요청하지 않아도 비지속 메시지를 클라이언트에 송신할 수 있습니다.

클라이언트는 서버의 메시지가 필요할 때 서버에 요청을 송신합니다. 이용하는 각 메시지에 대해 별도의 요청을 송신합니다. 이러한 요청 메시지 송신을 방지하여 클라이언트의 비지속 메시지 이용 성능을 향상시키려는 경우 클라이언트가 미리 읽기를 사용하도록 구성할 수 있습니다. 미리 읽기를 사용하면 애플리케이션이 요청하지 않아도 메시지를 클라이언트에 송신할 수 있습니다.

미리 읽기를 사용하면 클라이언트 애플리케이션에서 비지속 메시지를 이용할 때 성능을 향상시킬 수 있습니다. 이러한 성능 개선은 MQI 및 JMS 애플리케이션 모두에서 가능합니다. MQGET을 사용하거나 비동기로 이용하는 클라이언트 애플리케이션은 비지속 메시지 이용 시 성능 개선의 이점이 있습니다.

MQOO_READ_AHEAD와 함께 MQOPEN을 호출할 경우 특정 조건이 충족되면 IBM MQ 클라이언트에서는 미리 읽기만 가능합니다. 이러한 조건은 다음과 같습니다.

- 클라이언트 애플리케이션이 컴파일되고 스레드된 IBM MQ MQI 클라이언트 라이브러리에 대해 링크되어야 합니다.
- 클라이언트 채널이 TCP/IP 프로토콜을 사용해야 합니다.
- 채널이 클라이언트 및 서버 채널 정의 모두에서 0이 아닌 SharingConversations(SHARECNV) 설정을 사용해야 합니다.

미리 읽기가 설정되어 있으면, 메시지는 미리 읽기 버퍼라고 하는 클라이언트의 메모리 버퍼로 송신됩니다. 클라이언트에는 미리 읽기가 사용 가능한 상태로 열려 있는 각 큐에 대한 미리 읽기 버퍼가 있습니다. 미리 읽기 버퍼의 메시지는 지속되지 않습니다. 클라이언트는 이용한 데이터 용량에 대한 정보로 서버를 주기적으로 업데이트 합니다.

모든 옵션이 사용하도록 지원되는 것이 아니므로 모든 클라이언트 애플리케이션 디자인이 미리 읽기 사용에 적합한 것은 아닙니다. 일부 옵션은 미리 읽기가 사용 가능할 때 MQGET 호출 간에 일치해야 합니다. 클라이언트가 MQGET 호출 간에 해당 선택 기준을 대체하는 경우, 미리 읽기 버퍼에 저장될 메시지가 클라이언트 미리 읽기 버퍼에 스트랜드 상태로 유지됩니다. 자세한 정보는 [802 페이지의 『비지속 메시지의 성능 개선』](#)의 내용을 참조하십시오.

미리 읽기 구성은 세 가지 속성(MaximumSize, PurgeTime 및 UpdatePercentage)으로 제어되며, 이러한 속성은 IBM MQ 클라이언트 구성 파일의 MessageBuffer 스탠자에 지정되어 있습니다.

클라이언트 애플리케이션에서 비동기 넣기 사용

비동기 넣기를 사용하면 애플리케이션이 큐 관리자의 응답을 대기하지 않고 메시지를 큐에 넣을 수 있습니다. 이를 통해 일부 상황에서 메시징 성능을 개선할 수 있습니다.

일반적으로 애플리케이션은 MQPUT 또는 MQPUT1을 사용하여 메시지를 큐에 넣을 때, MQI 요청이 처리되었음을 큐 관리자가 확인하도록 대기해야 합니다. 특히 클라이언트 바인딩을 사용하는 애플리케이션 및 많은 소용량 메시지를 큐에 넣는 애플리케이션의 경우 그 대신에 비동기적으로 메시지를 넣도록 선택하여 메시징 성능을 향상시킬 수 있습니다. 애플리케이션이 메시지를 비동기로 넣는 경우, 큐 관리자는 각 호출의 성공 또는 실패를 리턴하지 않지만 사용자가 대신 주기적으로 오류를 확인할 수 있습니다.

메시지를 큐에 비동기적으로 넣으려면 MQPMO 구조의 Options 필드에 MQPMO_ASYNC_RESPONSE 옵션을 사용하십시오.

메시지가 비동기 넣기에 적합하지 않은 경우 동기적으로 큐에 넣어집니다.

MQPUT 또는 MQPUT1에 대한 비동기 넣기 응답을 요청하는 경우, CompCode와 MQCC_OK 및 MQRC_NONE의 이유가 반드시 큐에 메시지가 성공적으로 넣어졌음을 의미하지는 않습니다. 각 개별 MQPUT 또는 MQPUT1 호출의 성공 또는 실패가 바로 리턴되지 않더라도 비동기 호출에서 발생한 첫 번째 오류는 나중에 MQSTAT 호출을 통해 판별할 수 있습니다.

MQPMO_ASYNC_RESPONSE에 대한 자세한 정보는 [MQPMO 옵션](#)을 참조하십시오.

비동기 넣기 샘플 프로그램은 사용 가능한 일부 기능을 보여줍니다. 프로그램의 기능 및 디자인에 대한 세부사항은 [988 페이지의 『비동기 Put 샘플 프로그램』](#)의 내용을 참조하십시오.

클라이언트 애플리케이션의 공유 대화 사용

공유 대화가 허용되는 환경에서 대화는 MQI 채널 인스턴스를 공유할 수 있습니다.

공유 대화는 두 필드(둘 다 SharingConversations라고 함)로 제어되며, 하나는 채널 정의(MQCD) 구조 부분이고 다른 하나는 채널 엑시트 매개변수(MQCXP) 구조 부분입니다. MQCD의 SharingConversations 필드는 정수 값

이며, 채널과 연관된 채널 인스턴스를 공유할 수 있는 최대 대화 수를 판별합니다. MQCXP의 SharingConversations 필드는 부울 값이며, 채널 인스턴스가 현재 공유되는지 여부를 표시합니다.

공유 대화가 허용되지 않는 환경에서 동일한 MQCD를 지정하는 새 클라이언트 연결은 채널 인스턴스를 공유하지 않습니다.

새 클라이언트 애플리케이션 연결은 다음 조건이 참이면 채널 인스턴스를 공유합니다.

- 채널 인스턴스의 클라이언트 연결 및 서버 연결 끝 모두 공유 대화용으로 구성되며, 이러한 값은 채널 엑시트로 대체되지 않습니다.
- 클라이언트 연결 MQCD 값(클라이언트 MQCONNx 호출 또는 클라이언트 채널 정의 테이블(CCDT)에서 제공됨)은 기존 채널 인스턴스가 처음으로 설정될 때 CCDT 또는 클라이언트 MQCONNx 호출에서 제공된 클라이언트 연결 MQCD 값과 정확히 일치합니다. 원래 MQCD는 나중에 엑시트 또는 채널 조정으로 대체되었을 수 있지만, 이러한 변경이 일어나기 전에 클라이언트 시스템에 제공된 값에 대해 일치 항목이 작성된다는 점에 유의하십시오.
- 서버 측의 공유 대화 한계가 초과되지 않습니다.

새 클라이언트 애플리케이션 연결이 다른 대화와의 채널 인스턴스 공유를 실행하는 기준에 일치하는 경우 해당 대화에서 엑시트가 호출되기 전에 이 의사결정이 이루어집니다. 이러한 대화의 엑시트는 해당 대화가 다른 대화와 채널 인스턴스를 공유하고 있다는 사실을 대체할 수 없습니다. 새 채널 정의에 일치하는 기존 채널 인스턴스가 없으면 새 채널 인스턴스가 연결됩니다.

채널 조정은 채널 인스턴스의 첫 번째 대화에 대해서만 발생합니다. 채널 인스턴스의 조정된 값은 해당 스테이지에 고정되며 후속 대화가 시작될 때 이를 대체할 수 없습니다. 또한 TLS 인증은 첫 번째 대화에 대해서만 발행합니다.

채널 인스턴스의 클라이언트 연결 또는 서버 연결 끝에서 소켓의 첫 번째 대화에 대한 모든 보안, 송신 또는 수신 엑시트의 초기화 중에 MQCD SharingConversations 값이 대체되는 경우, 이 모든 엑시트가 초기화된 후의 새 값을 사용하여 채널 인스턴스의 공유 대화 값을 판별합니다(가장 낮은 값이 우선함).

공유 대화에 대한 조정 값이 0이면 채널 인스턴스가 공유되지 않습니다. 이 필드를 0으로 설정한 추가적인 엑시트 프로그램은 자체 채널 인스턴스에서 유사하게 실행됩니다.

공유 대화에 대한 조정 값이 0보다 크면 엑시트에 대한 후속 호출의 MQCXP SharingConversations가 TRUE로 설정됩니다. 이는 이 채널 인스턴스에 다른 엑시트 프로그램이 동시에 나타날 수 있는 것을 의미합니다.

채널 엑시트 프로그램을 작성할 때에는 공유 대화를 포함하는 채널 인스턴스에서 실행할지 여부를 고려하십시오. 채널 인스턴스가 공유 대화를 포함하는 경우, MQCD 필드를 변경하는 채널 엑시트의 다른 인스턴스에 미치는 영향을 고려하십시오. 모든 MQCD 필드는 모든 공유 대화에 걸쳐 공유 값을 가집니다. 채널 인스턴스가 설정된 후 엑시트 프로그램이 MQCD 필드를 대체하려 시도하는 경우, 채널 인스턴스에서 실행하는 엑시트 프로그램의 기타 인스턴스가 동시에 동일한 필드를 대체하려 시도 중일 수 있으므로 문제점이 발생할 수 있습니다. 엑시트 프로그램에 이러한 상황이 발생하면 엑시트 코드의 MQCD에 대한 액세스를 직렬화해야 합니다.

대화를 공유하도록 정의된 채널에 대해 작업 중이지만 특정 채널 인스턴스에서 공유하고 싶지 않으면 채널 인스턴스의 첫 번째 대화에서 채널 엑시트를 초기화할 때 SharingConversations의 MQCD 값을 1 또는 0으로 설정하십시오. SharingConversations의 값에 대한 설명은 [SharingConversations](#)를 참조하십시오.

예

공유 대화가 사용 가능합니다.

엑시트 프로그램을 지정하는 클라이언트 연결 채널 정의를 사용하고 있습니다.

처음 이 채널을 시작할 때 엑시트 프로그램은 초기화 시 일부 MQCD 매개변수를 대체합니다. 이러한 매개변수는 채널에 의해 작동되기 때문에, 현재 채널 실행과 관련되는 정의가 원래 제공된 정의와는 다릅니다. MQCXP SharingConversations 매개변수는 TRUE로 설정됩니다.

다음 번에 애플리케이션이 이 채널을 사용하여 연결될 때, 대화는 동일한 원래 채널 정의가 있기 때문에 이전에 시작된 채널 인스턴스에서 실행됩니다. 애플리케이션이 두 번째로 연결하는 채널 인스턴스는 첫 번째로 연결한 인스턴스와 같습니다. 따라서 엑시트 프로그램에 의해 대체된 정의를 사용합니다. 엑시트 프로그램은 두 번째 대화에 대해 초기화될 때 MQCD 필드를 대체할 수 없더라도 채널에 적용되지 않습니다. 이러한 동일한 특성은 채널 인스턴스를 공유하는 모든 후속 대화에 적용됩니다.

MQCONNx 사용

MQCONNx 호출을 사용하여 MQCNO 구조에 채널 정의(MQCD) 구조를 지정할 수 있습니다.

이를 통해 호출 클라이언트 애플리케이션이 런타임 시 클라이언트 연결 채널의 정의를 지정할 수 있습니다. 자세한 내용은 MQCNO를 IBM MQ MQI client 사용하여 클라이언트 연결 채널 생성하기를 참조하십시오.

MQCONNx를 사용할 경우, 서버에서 발행되는 호출은 서버 레벨 및 리스너 구성에 따라 다릅니다.

클라이언트에서 MQCONNx를 사용할 경우에는 다음 옵션이 무시됩니다.

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING

사용 가능한 MQCD 구조는 사용 중인 MQCD 버전 번호에 따라 다릅니다. MQCD 버전(MQCD_VERSION)에 대한 정보는 MQCD 버전을 참조하십시오. 예를 들어, MQCD 구조를 사용하여 채널 엑시트 프로그램을 서버에 전달할 수 있습니다. MQCD 버전 3 이상을 사용할 경우에는 이 구조를 사용하여 엑시트 배열을 서버에 전달할 수 있습니다. 이 함수를 사용하여 기존 엑시트를 수정하기 보다는 각 조작마다 엑시트를 추가함으로써 동일한 메시지에 대해 암호화 및 압축 등 둘 이상의 조작을 수행할 수 있습니다. MQCD 구조에서 배열을 지정하지 않으면 단일 엑시트 필드가 검사됩니다. 채널 엑시트 프로그램에 대한 자세한 정보는 710 페이지의 『메시지 채널에 대한 채널 엑시트 프로그램』의 내용을 참조하십시오.

MQCONNx의 공유 연결 핸들

공유 연결 핸들을 사용하여 동일한 프로세스 내의 서로 다른 스레드 간에 핸들을 공유할 수 있습니다.

공유 연결 핸들을 지정하면, 후속 MQI 호출 시 프로세스의 임의의 스레드에 MQCONNx 호출에서 리턴되는 연결 핸들을 전달할 수 있습니다.

참고: IBM MQ MQI client의 공유 연결 핸들을 사용하면 공유 연결 핸들을 지원하지 않는 서버 큐 관리자에 연결할 수 없습니다.

IBM MQ MQI clients의 애플리케이션 빌드

애플리케이션은 IBM MQ MQI client 환경에서 빌드하고 실행할 수 있습니다. 애플리케이션을 빌드하고 사용되는 IBM MQ MQI client에 링크해야 합니다. 애플리케이션을 빌드하고 링크하는 방식은 사용되는 플랫폼 및 프로그래밍 언어에 따라 달라집니다.

애플리케이션이 클라이언트 환경에서 실행되는 경우, 다음 표에 표시된 언어로 애플리케이션을 작성할 수 있습니다.

표 148. 클라이언트 환경에서 지원되는 프로그래밍 언어						
클라이언트 플랫폼	C	C++	COBOL	pTAL	RPG	Visual Basic
 AIX	예	예	예			
 IBM i	예		예		예	
 Linux	예	예	예			
 Windows	예	예	예			예

IBM MQ MQI client 코드와 C 애플리케이션 링크

애플리케이션을 작성한 후 IBM MQ, 해당 애플리케이션을 실행하려면 해당 애플리케이션을 IBM MQ MQI client 코드에 IBM MQ MQI client 연결해야 합니다.

다음 두 가지 방법으로 애플리케이션을 IBM MQ MQI client 코드에 링크할 수 있습니다.

1. 곧바로 큐 관리자에 애플리케이션을 연결하는 방법. 이 경우 큐 관리자는 애플리케이션과 같은 시스템에 있어야 합니다.
2. 같거나 다른 시스템에서 큐 관리자에 대한 액세스 권한을 부여하는 클라이언트 라이브러리 파일에 연결하는 방법

IBM MQ는 각 환경에 대해 클라이언트 라이브러리 파일을 제공합니다.

AIX

비스레드 애플리케이션용 libmqic.a 라이브러리 또는 스레드 애플리케이션용 libmqic_r.a 라이브러리

Linux

비스레드 애플리케이션용 libmqic.so 라이브러리 또는 스레드 애플리케이션용 libmqic_r.so 라이브러리

IBM i

비스레드 애플리케이션용 LIBMQIC 클라이언트 서비스 프로그램 또는 스레드 애플리케이션용 LIBMQIC_R 서비스 프로그램과 클라이언트 애플리케이션을 바인딩합니다.

Windows

MQIC32.LIB.

ALW **IBM MQ MQI client 코드와 C++ 애플리케이션 링크**

C++로 클라이언트에서 실행할 애플리케이션을 작성할 수 있습니다. 빌드 메소드는 환경에 따라 다릅니다.

C++ 애플리케이션을 링크하는 방법에 대한 정보는 [IBM MQ C++ 프로그램 빌드](#)를 참조하십시오.

C++ 사용의 모든 측면에 대한 전체 정보는 [C++ 사용](#)을 참조하십시오.

Multi **IBM MQ MQI client 코드와 COBOL 애플리케이션 링크**

IBM MQ MQI client에서 실행할 COBOL 애플리케이션을 작성했으면 적절한 라이브러리와 링크해야 합니다.

IBM MQ는 각 환경에 대해 클라이언트 라이브러리 파일을 제공합니다.

AIX

비스레드 COBOL 애플리케이션을 라이브러리 libmqicb.a와 링크하거나 스레드 COBOL 애플리케이션을 libmqicb_r.a와 링크합니다.

IBM i

COBOL 클라이언트 애플리케이션을 비스레드 애플리케이션용 AMQICSTUB 서비스 프로그램 또는 스레드 애플리케이션용 AMQICSTUB_R 서비스 프로그램과 바인딩합니다.

Windows

애플리케이션 코드를 32비트 COBOL용 MQICCB 라이브러리와 링크합니다. IBM MQ MQI client for Windows는 16비트 COBOL을 지원하지 않습니다.

IBM MQ MQI client 환경에서의 애플리케이션 실행

IBM MQ 애플리케이션은 특정 조건이 충족되면 코드를 변경하지 않고 전체 IBM MQ 환경 및 IBM MQ MQI client 환경 모두에서 실행할 수 있습니다.

이러한 조건은 다음과 같습니다.

- 동시에 둘 이상의 큐 관리자에 애플리케이션을 연결할 필요가 없습니다.
- MQCONN 또는 MQCONNX 호출 시 큐 관리자 이름 앞에 별표(*)를 지정하지 않았습니다.
- 애플리케이션은 [IBM MQ MQI client에서 실행하는 애플리케이션에](#) 나열된 예외를 사용할 필요가 없습니다.

참고: 링크 편집 시 사용하는 라이브러리는 애플리케이션을 실행해야 하는 환경을 판별합니다.

IBM MQ MQI client 환경에서 작업할 때 다음을 염두에 두십시오.

- IBM MQ MQI client 환경에서 실행 중인 각 애플리케이션에는 서버에 대한 자체 연결이 있습니다. 애플리케이션은 MQCONN 또는 MQCONNX 호출을 발행할 때마다 서버에 하나의 연결을 설정합니다.
- 애플리케이션은 메시지를 동시에 보내고 가져옵니다. 이는 클라이언트의 호출 발행 시간과 네트워크를 통한 완료 코드 및 이유 코드 리턴 사이의 대기를 의미합니다.
- 모든 데이터 변환은 서버에서 수행되지만, 시스템에서 구성된 CCSID를 대체하는 방법에 대한 정보는 [MQCCSID](#)도 참조하십시오.

큐 관리자에 IBM MQ MQI client 애플리케이션 연결

IBM MQ MQI client 환경에서 실행 중인 애플리케이션은 다양한 방법으로 큐 관리자에 연결할 수 있습니다. 환경 변수, MQCNO 구조 또는 클라이언트 정의 테이블을 사용할 수 있습니다.

IBM MQ 클라이언트 환경에서 실행 중인 애플리케이션이 MQCONN 또는 MQCONNX 호출을 발행하면 클라이언트는 연결 작성 방법을 식별합니다. IBM MQ 클라이언트의 애플리케이션에서 MQCONNX 호출을 발행하면, MQI 클라이언트 라이브러리는 다음과 같은 순서로 클라이언트 채널 정보를 검색합니다.

1. MQCNO 구조의 ClientConnOffset 또는 ClientConnPtr 필드 콘텐츠 사용(제공된 경우). 이러한 필드는 클라이언트 연결 채널의 정의로 사용될 채널 정의 구조(MQCD)를 식별합니다. 사전 연결 엑시트를 사용하여 연결 세부사항을 대체할 수 있습니다. 자세한 정보는 [737 페이지의 『저장소의 사전 연결 엑시트를 사용하여 연결 정의 참조』](#)의 내용을 참조하십시오.
2. 환경 **MQSERVER** 변수가 설정되어 있으면, 해당 변수가 정의하는 채널이 사용됩니다.
3. 파일이 mqclient.ini 정의되어 있고 채널 스탠자에 속성이 **ServerConnectionParms** 포함되어 있다면, 해당 파일이 정의하는 채널이 사용됩니다. 자세한 내용은 [IBM MQ MQI client 구성 파일, mqclient.ini](#) 클라이언트 구성 파일의 채널 스탠자 참조하십시오.
4. 환경 변수 **MQCHLIBMQCHLTAB** 와 가 설정된 경우, 이들이 가리키는 클라이언트 채널 정의 테이블이 사용됩니다. 또는 환경 **MQCCDTURL** 변수는 및 **MQCHLTAB** 환경 **MQCHLLIB** 변수를 조합하여 설정하는 것과 동등한 기능을 제공합니다. 가 **MQCCDTURL** 설정된 경우, 해당 설정이 가리키는 클라이언트 채널 정의 테이블이 사용됩니다. 자세한 내용은 [URL](#) 에서 CCDT 접근 방법을 참조하십시오.
5. 파일이 mqclient.ini 정의되어 있고 채널 스탠자에 및 **ChannelDefinitionFile** 속성이 **ChannelDefinitionDirectory** 포함된 경우, 이 속성들은 클라이언트 채널 정의 테이블을 찾는 데 사용됩니다. 자세한 내용은 [IBM MQ MQI client 구성 파일, mqclient.ini](#) 클라이언트 구성 파일의 채널 스탠자 참조하십시오.
6. 마지막으로, 환경 변수가 설정되지 않은 경우 클라이언트는 파일 mqs.ini 내의 AllQueueManagers 스탠자의 **DefaultPrefix** 속성에서 설정된 경로와 이름을 가진 클라이언트 채널 정의 테이블을 검색합니다. 자세한 내용은 [mqs.ini 파일의 AllQueueManagers 구절을 참조하십시오](#).

클라이언트 정의 테이블 검색에 실패하는 경우 클라이언트는 다음 경로를 사용합니다.

-   AIX and Linux: /var/mqm/AMQCLCHL.TAB
-  Windows: C:\Program Files\IBM\MQ\amqclchl.tab
-  IBM i: /QIBM/UserData/mqm/@ipcc
-  MQ Appliance: IBM MQ Appliance의 경우: QMname_AMQCLCHL.TAB. 이들은 mqbackup:// URI 아래에 표시됩니다.

이전 목록에서 설명된 옵션 중 첫 번째 옵션(MQCNO의 ClientConnOffset 또는 ClientConnPtr 필드 사용)은 MQCONNX 호출에 의해서만 지원됩니다. 애플리케이션이 MQCONNX 대신 MQCONN을 사용 중인 경우 목록에 표시되어 있는 순서대로 남아 있는 다섯 가지 방법으로 채널 정보를 검색합니다. 클라이언트가 채널 정보 찾기에 실패한 경우 MQCONN 또는 MQCONNX 호출은 실패합니다.

MQCONN 또는 MQCONNX 호출이 성공하려면 채널 이름(클라이언트 연결용)이 서버에 정의된 서버 연결 채널 이름과 일치해야 합니다.

관련 개념

[클라이언트 채널 정의 테이블에 대한 웹 주소 지정 가능 액세스](#)

관련 태스크

[서버와 클라이언트 간의 연결 구성](#)

관련 참조

[클라이언트 채널 정의 테이블](#)

[MQCNO - 연결 옵션](#)

환경 변수를 사용하여 큐 관리자에 클라이언트 애플리케이션 연결

클라이언트 환경에서 실행 중인 애플리케이션에 환경 변수를 사용하여 클라이언트 채널 정보를 제공할 수 있습니다.

IBM MQ MQI client 환경에서 실행 중인 애플리케이션은 다음 환경 변수를 사용하여 큐 관리자에 연결될 수 있습니다.

MQSERVER

환경 **MQSERVER** 변수는 최소 채널을 정의하는 데 사용됩니다. **MQSERVER** IBM MQ 서버의 위치와 사용할 통신 방법을 지정합니다.

MQCHLIB

환경 **MQCHLIB** 변수는 클라이언트 채널 정의 테이블(CCDT)이 포함된 파일의 디렉터리 경로를 지정합니다. 파일은 서버에서 작성되지만, IBM MQ MQI client 워크스테이션에서 복사될 수 있습니다.

MQCHLTAB

환경 **MQCHLTAB** 변수는 클라이언트 채널 정의 테이블(CCDT)을 포함하는 파일의 이름을 지정합니다.

환경 **MQCCDTURL** 변수는 및 **MQCHLTAB** 환경 **MQCHLIB** 변수를 조합하여 설정하는 것과 동등한 기능을 제공합니다. **MQCCDTURL** URL 를 단일 값으로 제공하여 클라이언트 채널 정의 테이블을 얻을 수 있도록 합니다. 자세한 내용은 [URL](#) 에서 CCDT 접근 방법을 참조하십시오.

MQCNO 구조를 사용하여 큐 관리자에 클라이언트 애플리케이션 연결

MQCONN 호출의 **MQCNO** 구조를 사용하여 제공되는 채널 정의 구조(MQCD)에 채널 정의를 지정할 수 있습니다.

자세한 내용은 [MQCNO](#)를 IBM MQ MQI client 사용하여 클라이언트 연결 채널 생성하기를 참조하십시오.

클라이언트 채널 정의 테이블을 사용하여 관리자에 클라이언트 애플리케이션 연결

MQSC DEFINE CHANNEL 명령을 사용하면 사용자가 제공하는 세부사항이 클라이언트 채널 정의 테이블(ccdt)에 배치됩니다. **MQCONN** 또는 **MQCONN** 호출의 **QMgrName** 매개변수의 콘텐츠에 따라 클라이언트가 연결하는 큐 관리자를 판별합니다.

클라이언트는 이 파일에 액세스하여 애플리케이션에서 사용할 채널을 판별합니다. 적당한 채널 정의가 둘 이상 있는 경우 채널 선택은 클라이언트 채널 가중치(CLNTWGHT)와 연결 연관관계(AFFINITY) 채널 속성의 영향을 받습니다.

자동 클라이언트 다시 연결 사용

여러 개의 컴포넌트를 구성하여 추가 코드를 작성하지 않고 클라이언트 애플리케이션이 자동으로 다시 연결되도록 할 수 있습니다.

자동 클라이언트 다시 연결은 인라인입니다. 클라이언트 애플리케이션 프로그램의 어느 지점에서든지 연결이 자동으로 복원되며, 오브젝트를 열기 위한 핸들이 모두 복원됩니다.

반대로, 수동 다시 연결의 경우 클라이언트 애플리케이션이 **MQCONN** 또는 **MQCONN**를 사용하여 연결을 다시 작성한 다음 오브젝트를 다시 열어야 합니다. 자동 클라이언트 다시 연결은 대부분의 클라이언트 애플리케이션에 적합하지만, 그렇다고 해서 모든 클라이언트 애플리케이션에 적합한 것은 아닙니다.

자세한 정보는 [자동 클라이언트 다시 연결](#)을 참조하십시오.

클라이언트 채널 정의 테이블의 역할

클라이언트 채널 정의 테이블(CCDT)은 클라이언트 연결 채널의 정의를 포함합니다. 클라이언트 애플리케이션에서 여러 대체 큐 관리자에 연결해야 할 경우에 특히 유용합니다.

클라이언트 채널 정의 테이블은 큐 관리자를 정의할 때 작성됩니다. 같은 파일이 둘 이상의 IBM MQ 클라이언트에서 사용될 수 있습니다.

클라이언트 애플리케이션에서 CCDT를 사용하는 여러 가지 방법이 있습니다. CCDT를 클라이언트 컴퓨터에 복사할 수 있습니다. CCDT를 둘 이상의 클라이언트가 공유하는 위치에 복사할 수 있습니다. CCDT가 서버에 있는 상태에서 클라이언트가 공유 파일로서 CCDT에 액세스할 수 있게 설정할 수 있습니다.

CCDT는 URI를 통해 접근 가능한 중앙 위치에 호스팅될 수 있어, 배포된 각 클라이언트마다 개별적으로 CCDT를 업데이트할 필요가 없습니다.

관련 개념

[클라이언트 채널 정의 테이블에 대한 웹 주소 지정 가능 액세스](#)

관련 태스크

[클라이언트 연결 채널 정의 액세스](#)

관련 참조

[클라이언트 채널 정의 테이블](#)

CCDT의 큐 관리자 그룹

클라이언트 채널 정의 테이블(CCDT)의 연결 세트를 큐 관리자 그룹으로 정의할 수 있습니다. 큐 관리자 그룹의 일부인 큐 관리자에 애플리케이션을 연결할 수 있습니다. MQCONN 또는 MQCONNX 호출에서 큐 관리자 이름 앞에 별표(*)를 붙여서 이를 수행할 수 있습니다.

다음과 같은 이유로 둘 이상의 서버 시스템에 대한 연결을 정의하기로 선택할 수 있습니다.

- 가용성을 개선하기 위해 실행 중인 큐 관리자 세트 중 하나에 클라이언트를 연결하려 합니다.
- 지난번에 성공적으로 연결된 동일한 큐 관리자에 클라이언트를 다시 연결하길 원하지만, 연결이 실패하면 다른 큐 관리자에 연결하려고 합니다.
- 연결이 실패하면, 클라이언트 프로그램에서 다시 MQCONN을 발행하여 다른 큐 관리자로서의 클라이언트 연결을 재시도하려 합니다.
- 연결이 실패하면, 클라이언트 코드를 작성하지 않고 다른 큐 관리자에 클라이언트 연결을 자동으로 다시 연결하려 합니다.
- 대기 인스턴스가 인계받는 경우 클라이언트 코드를 작성하지 않고 멀티 인스턴스 큐 관리자의 다른 인스턴스에 클라이언트 연결을 자동으로 다시 연결하려고 합니다.
- 일부 큐 관리자에 다른 큐 관리자보다 더 많은 클라이언트를 연결해서 여러 큐 관리자 사이에서 클라이언트 연결의 밸런스를 맞추려고 합니다.
- 많은 볼륨의 연결로 실패가 발생하는 경우, 다중 큐 관리자를 통해 장시간에 걸쳐 많은 클라이언트 연결의 재연결을 분산시키려고 합니다.
- 클라이언트 애플리케이션 코드를 변경하지 않고 큐 관리자를 이동할 수도 있습니다.
- 큐 관리자 이름을 알 필요가 없는 클라이언트 애플리케이션 프로그램을 작성하려 합니다.

다른 큐 관리자에 연결하는 것이 항상 적절한 것은 아닙니다. 예를 들어, WebSphere Application Server의 확장 트랜잭션 클라이언트 또는 Java 클라이언트는 예측 가능한 큐 관리자 인스턴스에 연결해야 할 수 있습니다. 자동 클라이언트 다시 연결은 IBM MQ classes for Java에서 지원되지 않습니다.

큐 관리자 그룹은 클라이언트 채널 정의 테이블(CCDT)에 정의된 연결 세트입니다. 이 세트는 채널 정의의 **QMNAME** 속성과 값이 동일한 구성원에 의해 정의됩니다.

927 페이지의 그림 127은 세 개의 큐 관리자 그룹 즉, CCDT에서 **QMNAME**(QM1) 및 **QMNAME**(QMGrp1)로 기록된 두 개의 이름 지정된 큐 관리자 그룹과 **QMNAME**(' ')으로 기록된 하나의 공백 또는 기본 그룹을 보여주는 클라이언트 연결 테이블의 그래픽 표현입니다.

1. 큐 관리자 그룹 QM1에는 큐 관리자 QM1 및 QM2에 연결하는 세 개의 클라이언트 연결 채널이 있습니다. QM1은 두 개의 다른 서버에 있는 멀티 인스턴스 큐 관리자일 수 있습니다.
2. 기본 큐 관리자 그룹에는 모든 큐 관리자에 연결하는 6개의 클라이언트 연결 채널이 있습니다.
3. QMGrp1에는 두 개의 큐 관리자 QM4 및 QM5에 대한 클라이언트 연결 채널이 있습니다.

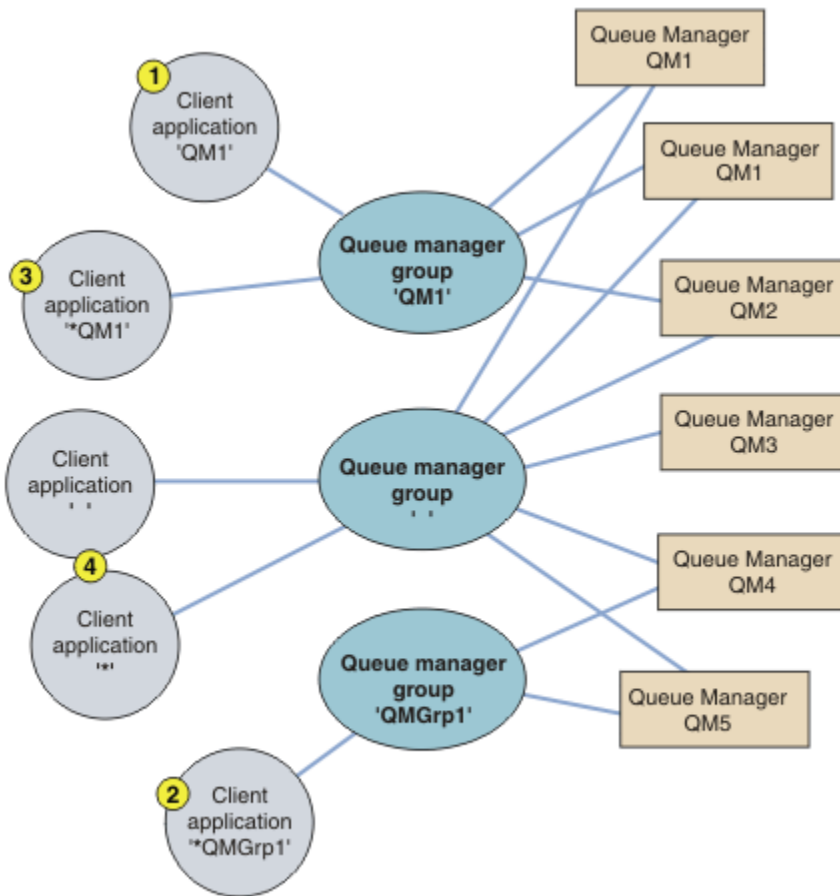


그림 127. 큐 관리자 그룹

927 페이지의 그림 127에는 번호가 매겨진 클라이언트 애플리케이션을 통해 이 클라이언트 연결 테이블을 사용하는 4가지 예제가 설명되어 있습니다.

1. 첫 번째 예제에서는 클라이언트 애플리케이션이 큐 관리자 이름 QM1을 **QmgrName** 매개변수로 MQCONN 또는 MQCONNx MQI 호출에 전달합니다. IBM MQ 클라이언트 코드는 일치하는 큐 관리자 그룹 QM1을 선택합니다. 그룹에는 세 개의 연결 채널이 포함되어 있으며, IBM MQ MQI client는 실행 중인 큐 관리자 QM1에 접속된 연결에 대한 IBM MQ 리스너를 찾을 때까지 이러한 각 채널을 차례로 사용하여 QM1에 연결하려고 시도합니다.

연결 시도의 순서는 클라이언트 연결 AFFINITY 속성의 값과 클라이언트 채널 가중치에 따라 달라집니다. 이러한 제한조건 내에서 연결 로드를 분산시키기 위해 세 가지 가능한 연결을 통해 장시간에 걸쳐 연결 시도의 순서가 랜덤화됩니다.

QM1의 실행 인스턴스에 대한 연결이 설정되면 클라이언트 애플리케이션이 발행한 MQCONN 또는 MQCONNx 호출은 성공합니다.

2. 두 번째 예제에서는 클라이언트 애플리케이션이 접두부에 별표(*)가 있는 큐 관리자 이름 *QMGrp1을 **QmgrName** 매개변수로 MQCONN 또는 MQCONNx MQI 호출에 전달합니다. IBM MQ 클라이언트는 일치하는 큐 관리자 그룹 QMGrp1을 선택합니다. 이 그룹에는 두 개의 클라이언트 연결 채널이 포함되어 있으며, IBM MQ MQI client는 각 채널을 차례로 사용하여 임의의 큐 관리자에 연결하려고 시도합니다. 이 예제에서 IBM MQ MQI client는 연결을 완료해야 하고, 연결되는 큐 관리자의 이름은 상관없습니다.

연결 시도 순서에 대한 규칙은 이전과 동일합니다. 유일한 차이점은 큐 관리자 이름의 접두부에 별표(*)를 붙여서 클라이언트가 큐 관리자의 이름이 무엇이든 상관하지 않음을 나타내고 있다는 점입니다.

QMGrp1 큐 관리자 그룹의 채널로 연결된 임의의 큐 관리자의 실행 인스턴스에 대한 연결이 설정되면 클라이언트 애플리케이션이 발행한 MQCONN 또는 MQCONNx 호출은 성공합니다.

3. 세 번째 예제는 **QmgrName** 매개변수의 접두부가 별표(*)이기 때문에(*QM1) 본질적으로 두 번째와 동일합니다. 이 예제는 하나의 채널 정의에서 **QMNAME** 속성을 자체적으로 검사하여 클라이언트 채널 연결이 어느 큐 관리자에 연결될지 판별할 수 없음을 보여줍니다. QM1이라는 큐 관리자에 연결하기에는 채널 정의의 **QMNAME** 속성이 QM1이라는 사실로는 충분하지 않습니다. 클라이언트 애플리케이션이 **QmgrName** 매개변수에 별표(*) 접두부를 붙이면 어느 큐 관리자든 연결 대상이 될 수 있습니다.

이 경우, QM1 또는 QM2의 실행 인스턴스에 대한 연결이 설정되면 클라이언트 애플리케이션이 발행한 MQCONN 또는 MQCONNEX 호출은 성공합니다.

4. 네 번째 예제에서는 기본 그룹 사용을 설명합니다. 이 경우 클라이언트 애플리케이션이 별표('*') 또는 공백 ' '을 **QmgrName** 매개변수로 MQCONN 또는 MQCONNEX MQI 호출에 전달합니다. 클라이언트 채널 정의의 규칙에 따라 공백 **QMNAME** 속성은 기본 큐 관리자 그룹을 나타내고 공백 또는 별표(*) **QmgrName** 매개변수는 공백 **QMNAME** 속성과 일치합니다.

이 예제에서는 기본 큐 관리자 그룹에 모든 큐 관리자에 대한 클라이언트 채널 연결이 있습니다. 기본 큐 관리자 그룹을 선택하여 그룹의 큐 관리자에 애플리케이션을 연결할 수 있습니다.

임의의 큐 관리자의 실행 인스턴스에 대한 연결이 설정되면 클라이언트 애플리케이션이 발행한 MQCONN 또는 MQCONNEX 호출은 성공합니다.

참고: 기본 그룹은 기본 큐 관리자와 다르지만, 애플리케이션은 공백 **QmgrName** 매개변수를 사용하여 기본 큐 관리자 그룹이나 기본 큐 관리자에 연결합니다. 기본 큐 관리자 그룹의 개념은 클라이언트 애플리케이션에만 관련되고 기본 큐 관리자는 서버 애플리케이션에 관련됩니다.

두 번째 또는 세 번째 큐 관리자에 연결하는 채널을 비롯하여 하나의 큐 관리자에만 클라이언트 연결 채널을 정의하십시오. 두 개의 큐 관리자에 해당 채널을 정의하지 말고 두 클라이언트 채널 정의 테이블을 병합하십시오. 클라이언트는 하나의 클라이언트 채널 정의 테이블에만 액세스할 수 있습니다.

예

토픽 시작 부분에 있는 큐 관리자 그룹 사용의 이유 목록을 다시 보십시오. 큐 관리자 그룹 사용 시 해당 기능이 어떻게 제공됩니까?

큐 관리자 세트 중 하나에 연결

세트에 있는 모든 큐 관리자에 연결하여 큐 관리자 그룹을 정의하고, 접두부에 별표(*)가 있는 **QmgrName** 매개변수를 사용하여 그룹에 연결합니다.

동일한 큐 관리자에 다시 연결하되, 지난번에 연결한 큐 관리자가 사용 불가능한 경우 다른 큐 관리자에 연결
큐 관리자 그룹을 이전과 같이 정의하되, 각 클라이언트 채널 정의에는 **AFFINITY**(PREFERRED) 속성을 설정합니다.

연결이 실패하는 경우 다른 큐 관리자에 연결 재시도

큐 관리자 그룹에 연결하고, 연결이 끊어지거나 큐 관리자가 실패하는 경우 MQCONN 또는 MQCONNEX MQI 호출을 다시 발행합니다.

연결이 실패하는 경우 다른 큐 관리자에 자동으로 다시 연결

MQCONNEX **MQCNO** 옵션 MQCNO_RECONNECT를 사용하여 큐 관리자 그룹에 연결합니다.

다중 인스턴스 큐 관리자의 다른 인스턴스에 자동으로 다시 연결

앞의 예제와 동일하게 수행합니다. 이 경우 특정 멀티 인스턴스 큐 관리자의 인스턴스에 연결하도록 큐 관리자 그룹을 제한하려면 멀티 인스턴스 큐 관리자 인스턴스에만 연결하여 그룹을 정의하십시오.

QmgrName 매개변수에 별표(*) 접두부를 붙이지 않고 MQCONN 또는 MQCONNEX MQI 호출을 발행하도록 클라이언트 애플리케이션에 요청할 수도 있습니다. 이 방식으로 클라이언트 애플리케이션은 이름 지정된 큐 관리자에만 연결할 수 있습니다. 마지막으로 **MQCNO** 옵션을 MQCNO_RECONNECT_Q_MGR로 설정할 수 있습니다. 이 옵션은 이전에 연결된 동일한 큐 관리자에 대한 재연결을 허용합니다. 이 값을 사용하여 일반 큐 관리자의 동일한 인스턴스에 대한 재연결을 제한할 수도 있습니다.

일부 큐 관리자에 더 많은 클라이언트를 연결하여 큐 관리자 사이에서 클라이언트 연결의 밸런스 유지

큐 관리자 그룹을 정의하고 각 클라이언트 채널 정의에 **CLNTWGT** 속성을 설정하여 연결을 고르지 않게 분배합니다.

클라이언트 재연결 로드를 고르지 않게 분산시키고 연결 또는 큐 관리자 실패 후에는 장시간에 걸쳐 분산

앞의 예제와 동일하게 수행합니다. IBM MQ MQI client는 큐 관리자 사이에 재연결을 랜덤화하고 장시간에 걸쳐 재연결을 분산시킵니다.

클라이언트 코드를 변경하지 않고 큐 관리자 이동

CCDT는 큐 관리자의 위치에서 클라이언트 애플리케이션을 격리합니다. CCDT는 클라이언트에서 정의할 수 있으며 공유 위치에서 읽거나 웹 서버에서 폐치할 수 있는 데이터 파일입니다. 자세한 정보는 [클라이언트 채널 정의 테이블](#)을 참조하십시오.

큐 관리자 이름을 알지 못하는 클라이언트 애플리케이션 작성

큐 관리자 그룹 이름을 사용해서 조직의 클라이언트 애플리케이션과 관련이 있는 큐 관리자 그룹 이름의 이름 지정 규칙을 설정하고, 큐 관리자 이름 지정이 아니라 사용자 솔루션의 아키텍처를 반영합니다.

큐 공유 그룹에 연결

큐 공유 그룹의 일부인 큐 관리자에 애플리케이션을 연결할 수 있습니다. MQCONN 또는 MQCONNX 호출에서 큐 관리자 이름 대신 큐 공유 그룹 이름을 사용하여 이 작업을 수행할 수 있습니다.

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

클라이언트 채널 정의는 큐 공유 그룹 일반 인터페이스를 사용하여 그룹에서 사용 가능한 큐 관리자에 연결해야 합니다. 자세한 정보는 [큐 공유 그룹에 클라이언트 연결](#)을 참조하십시오. 리스너와 연결될 큐 관리자가 큐 공유 그룹의 구성원인지 확인하는 검사가 수행됩니다.

공유 큐에 대한 자세한 정보는 [공유 큐 및 큐 공유 그룹](#)을 참조하십시오.

채널 가중치 및 연관관계의 예

이 예제는 0이 아닌 ClientChannelWeights를 사용할 때 클라이언트 연결 채널이 선택되는 방법을 보여줍니다.

ClientChannelWeight 및 ConnectionAffinity 채널 속성은 연결에 적당한 둘 이상의 채널이 사용 가능할 때 클라이언트 연결 채널이 선택되는 방법을 제어합니다. 이러한 채널은 각기 다른 큐 관리자에 연결하여 고가용성, 워크로드 밸런싱 또는 둘 다 제공할 수 있도록 구성되어 있습니다. 여러 큐 관리자 중 하나에 연결하는 MQCONN 호출은 다음에 설명된 대로 큐 관리자 이름 앞에 별표를 접두부로 추가해야 합니다. [MQCONN 호출 예: 예 1. 큐 관리자 이름에 별표\(*\)가 포함되어 있습니다.](#)

연결에 적용 가능한 후보 채널은 QMNAME 속성이 MQCONN 호출에 지정된 큐 관리자 이름과 일치하는 채널입니다. 연결에 적용 가능한 모든 채널에 값이 0(기본값)인 ClientChannelWeight가 있으면 다음 예제와 같이 채널이 알파벳순으로 선택됩니다. [MQCONN 호출 예: 예 1. 큐 관리자 이름에 별표\(*\)가 포함되어 있습니다.](#)

다음 예제는 0이 아닌 ClientChannelWeights가 사용될 때 발생하는 상황을 설명합니다. 이 기능은 의사 난수 채널 선택을 포함하기 때문에, 예제에서는 확인할 수는 없지만 발생할 가능성이 있는 일련의 조치를 보여줍니다.

예 1. ConnectionAffinity가 PREFERRED로 설정될 때 채널 선택

이 예제에서는 IBM MQ MQI client가 CCDT에서 ConnectionAffinity가 PREFERRED로 설정된 채널을 선택하는 방법을 설명합니다.

이 예제에서 여러 클라이언트 시스템은 큐 관리자가 제공하는 CCDT(Client Channel Definition Table)를 사용합니다. CCDT는 (DEFINE CHANNEL 명령의 구문을 사용하여 표시된) 다음 속성을 가진 클라이언트 연결 채널을 포함합니다.

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(PREFERRED)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(PREFERRED)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(PREFERRED)
```

애플리케이션은 MQCONN(*CORE)을 발행합니다.

채널 A는 QMNAME 속성이 일치하지 않으므로 이 연결에 대한 후보가 아닙니다. 채널 B, C 및 D가 후보로 식별되고, 해당 가중치에 근거하여 환경 설정의 순서대로 배치됩니다. 이 예에서 순서는 C, B, D일 수 있습니다. 클라이언트는 core2.ops.company.example에서 큐 관리자에 연결을 시도합니다. MQCONN 호출이 큐 관리자 이름에 별표를 포함하므로, 해당 주소의 큐 관리자 이름은 검사하지 않습니다.

AFFINITY(PREFERRED)를 사용하면 이 특정 클라이언트 시스템이 연결할 때마다 환경 설정의 동일한 초기 순서대로 채널을 배치한다는 것을 알고 있어야 합니다. 이는 다른 프로세스나 다른 시기의 연결에도 적용됩니다.

이 예제에서 core2.ops.company.example의 큐 관리자에는 이를 수 없습니다. 클라이언트는 채널 B가 환경 설정의 순서로 다음이기 때문에 core1.ops.company.example에 연결하려고 시도합니다. 또한 채널 C는 가장 덜 선호됨 상태로 강등됩니다.

동일한 애플리케이션에서 두 번째 MQCONN(*CORE) 호출을 발행합니다. 채널 C가 이전 연결에서 강등되었으므로 가장 선호하는 채널은 이제 B입니다. 이 연결은 core1.ops.company.example에 작성됩니다.

동일한 CCDT(Client Channel Definition Table)를 공유하는 두 번째 시스템은 환경 설정의 다른 초기 순서로 채널을 배치할 수 있습니다. 예를 들어, D, B, C입니다. 모든 채널이 작동하는 정상 상황에서는, 이 시스템의 애플리케이션이 core3.ops.company.example에 연결되는 반면 첫 번째 시스템의 애플리케이션은 core2.ops.company.example에 연결됩니다. 이를 통해 여러 큐 관리자에 걸쳐 많은 클라이언트의 워크로드를 밸런싱할 수 있고 각 개별 클라이언트가 동일한 큐 관리자에 연결(사용 가능한 경우)할 수 있습니다.

예 2. ConnectionAffinity가 NONE으로 설정될 때 채널 선택

이 예제에서는 IBM MQ MQI client가 CCDT에서 ConnectionAffinity가 NONE으로 설정된 채널을 선택하는 방법을 설명합니다.

이 예제에서 여러 클라이언트는 큐 관리자가 제공하는 CCDT(Client Channel Definition Table)를 사용합니다. CCDT는 (DEFINE CHANNEL 명령의 구문을 사용하여 표시된) 다음 속성을 가진 클라이언트 연결 채널을 포함합니다.

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(NONE)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(NONE)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(NONE)
```

애플리케이션은 MQCONN(*CORE)을 발행합니다. 앞의 예제에서와 같이 채널 A는 QMNAME이 일치하지 않으므로 고려되지 않습니다. 채널 B, C 또는 D는 해당 가중치에 따라 50%, 30% 또는 20%의 가능성으로 선택됩니다. 이 예제에서는 채널 B를 선택할 수 있습니다. 환경 설정의 지속적 순서는 작성되지 않습니다.

두 번째 MQCONN(*CORE) 호출이 수행됩니다. 여기서도 적용 가능한 세 채널 중 하나가 동일한 가능성으로 선택됩니다. 이 예제에서는 채널 C가 선택됩니다. 그러나 core2.ops.company.example이 응답하지 않으므로, 나머지 후보 채널 사이에서 또 하나가 선택됩니다. 채널 B가 선택되고 애플리케이션이 core1.ops.company.example에 연결됩니다.

AFFINITY(NONE)이면, 각 MQCONN 호출은 다른 호출과는 별개입니다. 따라서 이 예제 애플리케이션에서 세 번째 MQCONN(*CORE)을 작성할 때, B 또는 D 중 하나를 선택하기 전에 끊어진 채널 C를 통해 연결하려고 한 번 더 시도할 수 있습니다.

MQCONN 호출의 예

MQCONN을 사용하여 특정 큐 관리자에 연결하거나 그룹의 큐 관리자 중 하나에 연결하는 예입니다.

다음의 각 예제에서 네트워크는 동일합니다. 동일한 IBM MQ MQI client에서 두 개의 서버에 대한 연결이 정의되어 있습니다. (이러한 예제에서 MQCONN 호출 대신 MQCONNx 호출을 사용할 수 있습니다.)

서버 시스템에서 실행 중인 두 개의 큐 관리자가 있으며 하나의 이름은 SALE이고, 다른 하나의 이름은 SALE_BACKUP입니다.

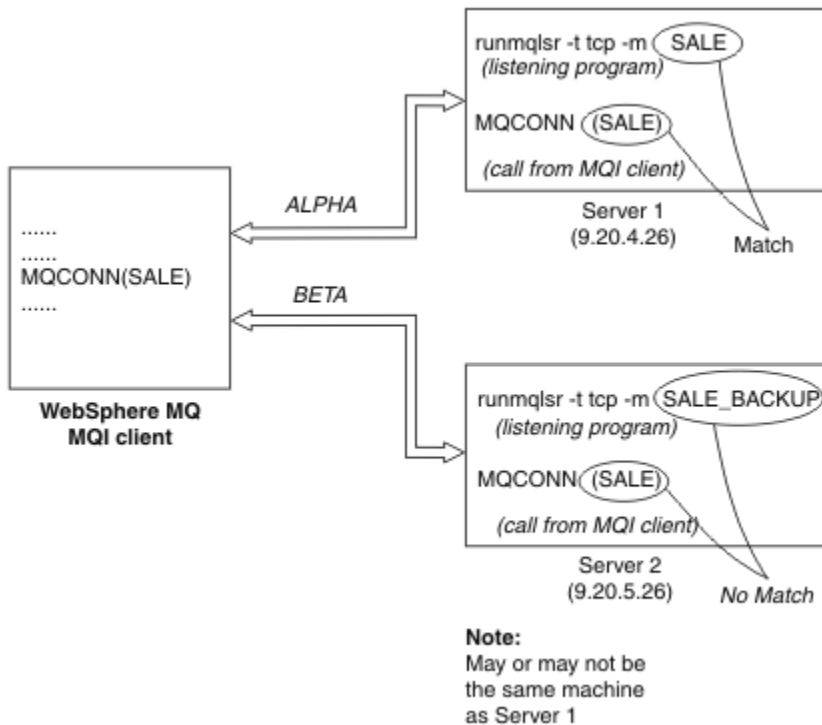


그림 128. MQCONN 예

이 예제에서 채널의 정의는 다음과 같습니다.

SALE 정의:

```
DEFINE CHANNEL(ALPHA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')

DEFINE CHANNEL(ALPHA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) DESCR('IBM MQ MQI client connection to server 1') +
QMNAME(SALE)

DEFINE CHANNEL(BETA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.5.26) DESCR('IBM MQ MQI client connection to server 2') +
QMNAME(SALE)
```

SALE_BACKUP 정의:

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')
```

클라이언트 채널 정의는 다음과 같이 요약할 수 있습니다.

이름	CHLTYPE	TRPTYPE	CONNAME	QMNAME
ALPHA	CLNTCONN	TCP	9.20.4.26	SALE
BETA	CLNTCONN	TCP	9.20.5.26	SALE

MQCONN 예제의 설명 내용

이 예제에는 백업 시스템으로 여러 큐 관리자를 사용하는 방법이 설명되어 있습니다.

서버 1에 대한 통신 링크가 일시적으로 끊어졌다고 가정해 보십시오. 백업 시스템으로 여러 큐 관리자를 사용하는 방법이 설명되어 있습니다.

각 예제마다 다른 MQCONN 호출을 다루며 다음 규칙을 적용하여 제시된 특정 예제에서 발생하는 상황을 설명합니다.

1. 클라이언트 채널 정의 테이블(CCDT)에서는 MQCONN 호출에 지정된 큐 관리자 이름에 해당하는 큐 관리자 이름(QMNAME 필드)이 알파벳 채널 이름순으로 스캔됩니다.
2. 일치하는 항목이 있으면 해당 채널 정의를 사용합니다.
3. 연결 이름(CONNAME)으로 식별된 시스템에 대해 채널 시작을 시도합니다. 시작 시도가 성공하면 애플리케이션은 계속 진행됩니다. 다음이 필요합니다.
 - 서버에서 실행할 리스너
 - 클라이언트에서 연결하려는 큐 관리자(지정된 경우)와 동일한 큐 관리자에 연결할 리스너
4. 채널 시작 시도가 실패하고 클라이언트 채널 정의 테이블에 둘 이상의 입력 항목이 있는 경우(이 예에는 두 개의 입력 항목이 있음) 파일에 일치 항목이 더 있는지 검색합니다. 일치 항목이 있으면 1단계에서 처리를 계속합니다.
5. 일치 항목이 없거나 클라이언트 채널 정의 테이블에 입력 항목이 더 이상 없고 채널을 시작하지 못한 경우, 애플리케이션을 연결할 수 없습니다. MQCONN 호출 시 적절한 이유 코드 및 완료 코드가 리턴됩니다. 애플리케이션은 리턴되는 이유 및 완료 코드에 따라 조치를 수행할 수 있습니다.

예 1. 큐 관리자 이름에 별표(*)가 포함됨

이 예제에서 애플리케이션은 연결되어 있는 큐 관리자와 관련이 없습니다. 애플리케이션은 별표를 포함하여 큐 관리자 이름에 대한 MQCONN 호출을 발행합니다. 적당한 채널이 선택됩니다.

애플리케이션은 다음을 발행합니다.

MQCONN (*SALE)

규칙에 따라 이 경우에는 다음과 같은 상황이 발생합니다.

1. 클라이언트 채널 정의 테이블(CCDT)에서는 애플리케이션 MQCONN 호출과 일치하는 큐 관리자 이름 SALE 이 스캔됩니다.
2. ALPHA 및 BETA의 채널 정의를 찾아냅니다.
3. 한 채널의 CLNTWGHT 값이 0인 경우 이 채널이 선택됩니다. 두 채널 모두 CLNTWGHT 값이 0인 경우, 알파벳순으로 첫 번째인 채널 ALPHA가 선택됩니다. 두 채널 모두 CLNTWGHT 값이 0이 아닌 경우, 해당 가중치에 근거하여 하나의 채널이 무작위로 선택됩니다.
4. 채널을 시작하려고 시도합니다.
5. BETA 채널이 선택된 경우 시작하려는 시도가 성공합니다.
6. ALPHA 채널이 선택된 경우 통신 링크가 중단됨으로 인해 채널을 시작하려는 시도가 실패합니다. 그러면 다음 단계가 적용됩니다.
 - a. 큐 관리자 이름 SALE에 대한 유일한 다른 채널은 BETA입니다.
 - b. 이 채널을 시작하려고 시도합니다. 이 시도가 성공합니다.
7. 리스너가 실행 중인지 점검한 결과 하나가 실행되고 있음이 확인되었습니다. SALE 큐 관리자에 연결되어 있지 않으나 MQI 호출 매개변수에 별표(*)가 포함되어 있으므로 확인하지 않습니다. 애플리케이션이 SALE_BACKUP 큐 관리자에 연결되어 처리를 계속합니다.

예 2. 큐 관리자 이름이 지정됨

이 예제에서 애플리케이션은 특정 큐 관리자에 연결해야 합니다. 애플리케이션은 해당 큐 관리자 이름에 대한 MQCONN 호출을 발행합니다. 적당한 채널이 선택됩니다.

다음 MQI 호출에서 볼 수 있듯이 애플리케이션을 SALE이라는 특정 큐 관리자에 연결해야 합니다.

MQCONN (SALE)

규칙에 따라 이 경우에는 다음과 같은 상황이 발생합니다.

1. 클라이언트 채널 정의 테이블(CCDT)에서는 애플리케이션 MQCONN 호출과 일치하는 큐 관리자 이름 SALE 이 알파벳 채널 이름순으로 스캔됩니다.
2. 일치하는 첫 번째 채널 정의는 ALPHA입니다.
3. 채널을 시작하려고 시도하지만 통신 링크가 끊어졌으므로 이 시도가 실패합니다.

4. 클라이언트 채널 정의 테이블에서는 다시 큐 관리자 이름 SALE이 스캔되고 채널 이름 BETA를 찾습니다.
5. 채널을 시작하려고 시도합니다. 이 시도가 성공합니다.
6. 리스너가 실행 중인지 검사한 결과 하나가 실행되고 있지만 SALE 큐 관리자에 연결되어 있지 않음을 보여줍니다.
7. 클라이언트 채널 정의 테이블에는 추가 입력 항목이 없습니다. 애플리케이션은 계속될 수 없으며 리턴 코드 MQRC_Q_MGR_NOT_AVAILABLE을 수신합니다.

예 3. 큐 관리자 이름이 공백 또는 별표(*)임

이 예제에서 애플리케이션은 연결되어 있는 큐 관리자와 관련이 없습니다. 애플리케이션은 공백 큐 관리자 이름 또는 별표를 지정하는 MQCONN을 발행합니다. 적당한 채널이 선택됩니다.

이는 932 페이지의 『예 1. 큐 관리자 이름에 별표(*)가 포함됨』에서와 같은 방식으로 처리됩니다.

참고: 이 애플리케이션이 IBM MQ MQI client 이외의 환경에서 실행 중이고 이름이 비어 있으면 기본 큐 관리자에 연결하려고 시도하는 것입니다. 이것은 클라이언트 환경에서 실행되는 경우가 아니며 액세스되는 큐 관리자는 채널이 연결된 리스너와 연관되어 있습니다.

애플리케이션은 다음을 발행합니다.

```
MQCONN ("")
```

또는

```
MQCONN (*)
```

규칙에 따라 이 경우에는 다음과 같은 상황이 발생합니다.

1. 클라이언트 채널 정의 테이블(CCDT)에서는 비어 있지만 애플리케이션 MQCONN 호출과 일치하는 큐 관리자 이름이 알파벳 채널 이름순으로 스캔됩니다.
2. ALPHA라는 채널 이름에 대한 입력 항목은 SALE의 정의에 큐 관리자 이름이 있습니다. 이는 공백의 큐 관리자 이름을 요구하는 MQCONN 호출 매개변수와 일치하지 않습니다.
3. 다음 입력 항목은 채널 이름 BETA입니다.
4. 정의에 있는 queue manager name은 SALE입니다. 이는 공백의 큐 관리자 이름을 요구하는 MQCONN 호출 매개변수와 또 다시 일치하지 않습니다.
5. 클라이언트 채널 정의 테이블에는 추가 입력 항목이 없습니다. 애플리케이션은 계속될 수 없으며 리턴 코드 MQRC_Q_MGR_NOT_AVAILABLE을 수신합니다.

클라이언트 환경에서의 트리거

애플리케이션이 실행 IBM MQ 중인 에서 전송된 메시지는 다른 메시지와 정확히 IBM MQ MQI clients 동일한 방식으로 트리거를 유발하는 데 기여하며, 서버와 클라이언트 양쪽에서 프로그램을 트리거하는 데 사용될 수 있습니다.

트리거는 [트리거 채널](#)에서 자세히 설명되어 있습니다.

트리거 모니터 및 시작할 애플리케이션은 동일한 시스템에 있어야 합니다.

트리거된 큐의 기본 특성은 서버 환경에 있는 특성과 동일합니다. 특히, 큐 z/OS 관리자에 로컬한 트리거드 큐에 메시지를 배치하는 클라이언트 애플리케이션에서 MQPMO 동기화점 제어 옵션이 지정되지 않은 경우, 메시지는 작업 단위 내에 배치됩니다. 그런 다음 트리거 조건이 충족되면, 트리거 메시지는 동일한 작업 단위 내의 이니시 에이션 큐에 놓여지고 작업 단위가 끝날 때까지 트리거 모니터로 검색할 수 없습니다. 트리거할 프로세스는 작업 단위가 끝날 때까지 시작되지 않습니다.

프로세스 정의

트리거링이 설정된 큐와 연관되어 있으므로 프로세스 정의를 서버에 정의해야 합니다.

프로세스 오브젝트는 트리거할 내용을 정의합니다. 클라이언트와 서버가 동일한 플랫폼에서 실행되고 있지 않은 경우, 트리거 모니터에서 시작된 프로세스는 *AppType*을 정의해야 합니다. 그렇지 않으면 서버는 기본 정의 (즉, 일반적으로 서버 시스템과 연관된 애플리케이션 유형)를 사용하므로 오류가 발생합니다.

예를 들어 트리거 모니터가 IBM MQ MQI client에서 실행 중이고 다른 운영 체제의 서버로 요청을 전송하려는 경우 MQAT_WINDOWS_NT를 정의해야 합니다. 그렇지 않으면 다른 운영 체제에서 기본 정의를 사용하고 프로세스는 실패합니다.

Multi 트리거 모니터

r에서 제공하는 트리거 모니터는 멀티플랫폼 IBM MQ for Multiplatforms 시스템의 클라이언트 환경에서 실행됩니다.

트리거 모니터를 실행하려면 다음 명령 중 하나를 실행하십시오.

- **IBM i** IBM i의 경우:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QmgrName '-q' InitQ)
```

- **ALW** AIX, Linux, and Windows 플랫폼:

```
runmqtmc [-m QMgrName] [-q InitQ]
```

기본 이니시에이션 큐는 기본 큐 관리자의 SYSTEM.DEFAULT.INITIATION.QUEUE입니다. 이니시에이션 큐는 트리거 모니터가 트리거 메시지를 찾는 위치입니다. 또한 해당 트리거 메시지에 대한 프로그램을 호출합니다. 이 트리거 모니터는 기본 애플리케이션 유형을 지원하며, 클라이언트 라이브러리를 링크하는 점을 제외하고는 runmqtm과 동일합니다.

트리거 모니터에서 빌드되는 명령 문자열은 다음과 같습니다.

1. 관련 프로세스 정의의 *ApplicId*. *ApplicId*는 명령행에 입력된 바와 같이, 실행할 프로그램의 이름입니다.
2. 이니시에이션 큐에서 확보되는, 따옴표로 묶인 MQTMC2 구조. 이 문자열을 포함하는 명령 문자열은 제공된 그대로 시스템 명령이 하나의 매개변수로 승인할 수 있도록 큰따옴표로 묶여 시작됩니다.
3. 관련 프로세스 정의의 *EnvrData*.

트리거 모니터는 시작한 애플리케이션이 완료될 때까지 이니시에이션 큐에 다른 메시지가 있는지 확인해보지 않습니다. 애플리케이션이 수행해야 할 처리가 많을 경우, 트리거 모니터는 도착하는 트리거 메시지 수를 유지하지 못할 수도 있습니다. 이 상황에 대처하는 두 가지 방법이 있습니다.

1. 추가 트리거 모니터 실행

보다 많은 트리거 모니터를 실행하도록 선택하면 어느 때든지 실행할 수 있는 최대 애플리케이션 수를 제어할 수 있습니다.

2. 백그라운드에서 시작된 애플리케이션 실행

백그라운드에서 애플리케이션을 실행하도록 선택하는 경우, IBM MQ는 실행할 수 있는 애플리케이션의 수에 제한을 두지 않습니다.

시스템에서 AIX and Linux 시작한 애플리케이션을 백그라운드에서 실행하려면 프로세스 정의의 *EnvrData* 명령어 끝에 &(앰퍼샌드)를 추가해야 합니다.

CICS 애플리케이션(비z/OS)

MQCONN 또는 MQCONNX 호출을 발행하는 비응용 CICSz/OS 프로그램은 CEDA에 RESIDENT로 정의되어야 합니다. CICS 서버 애플리케이션을 클라이언트로 다시 링크하면 동기점 지원 손실의 위험이 있습니다.

MQCONN 또는 MQCONNX 호출을 발행하는 비응용 CICSz/OS 프로그램은 CEDA에 RESIDENT로 정의되어야 합니다. 상주 코드를 가능한 한 작게 만들기 위해 MQCONN 또는 MQCONNX 호출을 발행하는 별도의 프로그램에 링크할 수 있습니다.

MQSERVER 환경 변수를 사용하여 클라이언트 연결을 정의하는 경우, 반드시 해당 CICSENV.CMD 파일에 명시해야 합니다.

IBM MQ 애플리케이션은 코드를 변경하지 않고 IBM MQ 클라이언트 또는 IBM MQ 서버 환경에서 실행할 수 있습니다. 그러나 서버 IBM MQ 환경에서는 SYNC POINT COORDINATOR 역할을 수행할 수 있으며 CICS, 와 MQCMIT 대신 EXEC CICS SYNCPOINT 및 EXEC CICS MQBACK SYNCPOINT ROLLBACK을 사용합니다. CICS 애플리케이션이 단순히 클라이언트로 다시 링크되면 동기점 지원이 손실됩니다. MQCMIT 그리고, IBM MQ MQI clientNET 플랫폼에서 실행되는 MQBACK 애플리케이션에 사용되어야 합니다.

ALW CICS 및 Tuxedo 애플리케이션 준비 및 실행

CICS 및 Tuxedo 애플리케이션을 클라이언트 애플리케이션으로 실행하려면, 서버 애플리케이션에서 사용되는 것과 다른 라이브러리를 사용합니다. 애플리케이션을 실행하는 사용자 ID 역시 다릅니다.

확장 트랜잭션 클라이언트 구성에 있는 지침을 따라 Tuxedo 애플리케이션을 애플리케이션으로 IBM MQ MQI client 실행할 수 있도록 CICS 준비하십시오.

다만, Tuxedo 애플리케이션 준비와 관련된 정보(IBM MQ에 CICS 포함된 샘플 프로그램 포함)는 사용자가 Tuxedo IBM MQ 서버 시스템에서 실행될 애플리케이션을 준비하고 있다고 가정한다는 점에 유의하십시오. 결과적으로, 이 정보는 서버 시스템에서 사용하도록 설계된 IBM MQ 라이브러리만을 참조합니다. 클라이언트 애플리케이션을 준비할 때 다음을 수행해야 합니다.

- 애플리케이션이 사용하는 언어 바인딩에 적절한 클라이언트 시스템 라이브러리를 사용하십시오. 예를 들면, 다음과 같습니다.
 -   AIX and Linux에서 C로 작성된 애플리케이션의 경우 libmqm 대신 libmqc 라이브러리를 사용하십시오.
 -  Windows 시스템에서는 mqm.lib 대신 mqc.lib 라이브러리를 사용하십시오.
- 936 페이지의 표 149 및 936 페이지의 표 150에 표시된 서버 시스템 라이브러리 대신, 동일한 클라이언트 시스템 라이브러리를 사용하십시오. 서버 시스템 라이브러리가 이 표에 나열되지 않은 경우 클라이언트 시스템에서 동일한 라이브러리를 사용하십시오.

표 149. AIX and Linux의 클라이언트 시스템 라이브러리	
IBM MQ 서버 시스템의 라이브러리	IBM MQ 클라이언트 시스템에서 사용되는 동등한 라이브러리
libmqmxa	libmqcxa
  libmqmxa64	  libmqcxa64

표 150. Windows 시스템의 클라이언트 시스템 라이브러리	
IBM MQ 서버 시스템의 라이브러리	IBM MQ 클라이언트 시스템에서 사용되는 동등한 라이브러리
mqmxa.lib	mqcxa.lib
mqmtux.lib	mqcxa.lib
mqmenc.lib	mqcxa.lib
mqmcics4.lib	mqccics4.lib

클라이언트 애플리케이션에서 사용되는 사용자 ID

서버 IBM MQ 애플리케이션을 에서 실행할 CICS 때, 일반적으로 CICS 사용자에서 트랜잭션의 사용자 ID로 전환됩니다. 그러나 아래에서 애플리케이션을 실행하면, IBM MQ MQI client 해당 CICS애플리케이션은 특권 CICS 권한을 유지합니다.

ALW CICS 및 Tuxedo 샘플 프로그램

CICS 및 Tuxedo 샘플 프로그램은 AIX, Linux, and Windows 시스템에서 사용하기 위한 것입니다.

936 페이지의 표 151에서는 AIX and Linux 클라이언트 시스템에서 사용하도록 제공되는 CICS 및 Tuxedo 샘플 프로그램을 나열합니다. 936 페이지의 표 152에는 Windows 클라이언트 시스템에 대한 해당 정보가 나열되어 있습니다. 또한 이 표에는 프로그램을 준비하고 실행하는 데 사용되는 파일도 나열되어 있습니다. 샘플 프로그램에 대한 설명은 991 페이지의 『CICS 트랜잭션 샘플』 및 1033 페이지의 『AIX, Linux, and Windows에서 TUXEDO 샘플 사용』의 내용을 참조하십시오.

표 151. AIX and Linux 클라이언트 시스템용 샘플 프로그램		
설명	소스	실행 가능 모듈
CICS 프로그램	amqscic0.ccs	amqscicc
CICS 프로그램의 헤더 파일	amqscih0.h	-
메시지를 넣을 Tuxedo 클라이언트 프로그램	amqstxpx.c	-
메시지를 가져올 Tuxedo 클라이언트 프로그램	amqstxgx.c	-
두 가지 클라이언트 프로그램의 Tuxedo 서버 프로그램	amqstxsx.c	-
Tuxedo 프로그램의 UBBCONFIG 파일	ubbstxcx.cfg	-
Tuxedo 프로그램의 필드 테이블 파일	amqstxvx.flds	-
Tuxedo 프로그램의 보기 설명 파일	amqstxvx.v	-

표 152. Windows 클라이언트 시스템용 샘플 프로그램		
설명	소스	실행 가능 모듈
CICS 트랜잭션	amqscic0.ccs	amqscicc
CICS 트랜잭션의 헤더 파일	amqscih0.h	-
메시지를 넣을 Tuxedo 클라이언트 프로그램	amqstxpx.c	-
메시지를 가져올 Tuxedo 클라이언트 프로그램	amqstxgx.c	-
두 가지 클라이언트 프로그램의 Tuxedo 서버 프로그램	amqstxsx.c	-
Tuxedo 프로그램의 UBBCONFIG 파일	ubbstxcx.cfg	-
Tuxedo 프로그램의 필드 테이블 파일	amqstxvx.fld	-
Tuxedo 프로그램의 보기 설명 파일	amqstxvx.v	-
Tuxedo 프로그램의 Make 파일	amqstxmc.mak	-
Tuxedo 프로그램의 ENVFILE 파일	amqstxen.env	-

ALW 오류 메시지 AMQ5203(CICS 및 Tuxedo 애플리케이션에 대해 수정한 경우)

확장 트랜잭션 클라이언트를 사용하는 CICS 또는 Tuxedo 애플리케이션을 실행할 때 표준 진단 메시지가 표시될 수 있습니다. 이들 중 하나는 확장 트랜잭션 클라이언트에서 사용하도록 수정되었습니다.

IBM MQ 오류 로그 파일에서 볼 수 있는 메시지는 [진단 메시지: AMQ4000-9999](#)에 설명되어 있습니다. 메시지 AMQ5203은 확장 트랜잭션 클라이언트에서 사용하도록 수정되었습니다. 다음은 수정된 메시지의 텍스트입니다.

AMQ5203: XA 인터페이스를 호출하는 중에 오류가 발생했습니다.

설명

오류 번호는 &2입니다. 여기서 값 1은 제공된 플래그 값 &1이 올바르지 않음을 나타내고, 2는 동일한 프로세스에서 스레드 및 비스레드 라이브러리를 사용하려고 시도했음을 나타내고, 3은 제공된 큐 관리자 이름 '&3'에 오류가 있음을 나타내고, 4는 자원 관리자 ID &1이 올바르지 않음을 나타내고, 5는 다른 큐 관리자가 이미 연결되었을 때 '&3'이라는 두 번째 큐 관리자를 사용하려고 시도했음을 나타내고, 6은 애플리케이션이 큐 관리자에 연결되어 있지 않을 때 트랜잭션 관리자가 호출되었음을 나타내고, 7은 다른 호출이 진행 중일 때 XA 호출이 수행되었음을 나타내고, 8은 매개변수 이름 '&5'에 대해 올바르지 않은 매개변수 값이 xa_open 호출의 xa_info 문자열 '&4'에 포함되었음을 나타내고, 9는 xa_open 호출의 xa_info 문자열 '&4'에 필수 매개변수(매개변수 이름 '&5')가 누락되었음을 나타냅니다.

사용자 응답

오류를 수정하고 조작을 다시 시도하십시오.

MTS 애플리케이션을 IBM MQ MQI client 애플리케이션으로 실행하기 위해 준비하려면 환경에 적합하도록 다음 지시사항을 따르십시오.

IBM MQ 리소스에 액세스하는 Microsoft 트랜잭션 서버(MTS) 애플리케이션을 개발하는 방법에 대한 일반적인 정보는 IBM MQ 헬프 센터의 MTS 섹션을 참조하세요.

MTS 애플리케이션을 IBM MQ MQI client 애플리케이션으로 실행하기 위해 준비하려면, 애플리케이션의 각 컴포넌트에 대해 다음 중 하나를 수행하십시오.

- 컴포넌트에서 MQI용 C 언어 바인딩을 사용하는 경우 952 페이지의 『Windows에서 C 프로그램 준비』의 지시사항을 따르십시오. 단, 컴포넌트를 mqic.lib 대신 라이브러리 mqicxa.lib에 링크하십시오.
- 컴포넌트에서 IBM MQ C++ 클래스를 사용하는 경우 516 페이지의 『Windows에 C++ 프로그램 빌드』의 지시사항을 따르십시오. 단, 컴포넌트를 imqc23vn.lib 대신 라이브러리 imqx23vn.lib에 링크하십시오.
- 컴포넌트가 `MQI` 용 Visual Basic 언어 바인딩을 사용하는 경우, Visual Basic 프로젝트를 정의할 때 '조건부 컴파일 인수' 필드에 를 입력하십시오 MqType=3.

IBM MQ JMS 애플리케이션 준비 및 실행

WebSphere Application Server 를 트랜잭션 관리자로 사용하여 클라이언트 모드에서 IBM MQ JMS 애플리케이션을 실행할 수 있습니다. 특정 경고 메시지가 표시될 수 있습니다.

WebSphere Application Server 를 트랜잭션 관리자로 사용하여 클라이언트 모드에서 IBM MQ JMS 애플리케이션을 준비하고 실행하려면 83 페이지의 『사용 IBM MQ classes for JMS/Jakarta Messaging』의 지시사항을 따르십시오.

IBM MQ JMS 클라이언트 애플리케이션을 실행할 때 다음 경고 메시지가 표시될 수 있습니다.

MQJE080

충분하지 않은 라이선스 갯수 - setmqcap 실행

MQJE081

라이선스 갯수 정보를 포함하는 파일의 형식이 올바르지 않음 - setmqcap 실행

MQJE082

라이선스 갯수 정보를 포함하는 파일이 없음 - setmqcap 실행

프로시저 애플리케이션 빌드

여러 프로시저 언어 중 하나로 IBM MQ 애플리케이션을 작성하고 여러 다른 플랫폼에서 애플리케이션을 실행할 수 있습니다.

AIX에서 절차적 애플리케이션 빌드

AIX 서적은 사용자가 작성하는 프로그램에서 실행 가능한 애플리케이션을 빌드하는 방법을 설명합니다.

이 주제에서는 AIX에서 실행할 IBM MQ for AIX 애플리케이션을 빌드할 때 수행해야 하는 추가 태스크 및 표준 태스크에 대한 변경사항을 설명합니다. C, C++ 및 COBOL이 지원됩니다. C++ 프로그램 준비에 대한 정보는 [C++ 사용의 내용](#)을 참조하십시오.

IBM MQ for AIX를 사용하여 실행 가능한 애플리케이션을 작성하기 위해 수행해야 할 태스크는 소스 코드가 기록된 프로그래밍 언어에 따라 다릅니다. 소스 코드로 MQI 호출을 코딩하는 것 외에도 사용 중인 언어에 대해 IBM MQ for AIX 포함 파일을 포함하도록 적절한 언어문을 추가해야 합니다. 이러한 파일의 콘텐츠를 숙지하십시오. 전체 설명은 677 페이지의 『IBM MQ 데이터 정의 파일』의 내용을 참조하십시오.

스레드 서버 또는 스레드 클라이언트 애플리케이션을 실행할 때 환경 변수 AIXTHREAD_SCOPE=S를 설정하십시오.

AIX에서 C 프로그램 준비

AIX 에서 IBMXL C/C++ 컴파일러 또는 C/C++ VisualAge 컴파일러를 사용하여 C 프로그램을 준비하고 컴파일하십시오.

시작하기 전에

다음 코드 예시에서는 가 IBM MQ 설치된 상위 `${MQ_INSTALLATION_PATH}` 디렉터리를 나타냅니다. 코드를 변경하지 않고 그대로 사용하면 변수에 설정한 값을 `${MQ_INSTALLATION_PATH}` 가져올 수 있으며, 또는 실제 경로로 `${MQ_INSTALLATION_PATH}` 대체할 수도 있습니다.

프로그램은 다음 라이브러리 파일 중 하나와 링크해야 합니다.

라이브러리 파일	프로그램/엑시트 유형
libmqm.a	C용 서버 (스레드 미사용 애플리케이션)
libmqm_r.a	C용 서버 (스레드 기반 애플리케이션)
libmqic.a & libmqm.a	C용 클라이언트(스레드 미사용 애플리케이션)
libmqic_r.a & libmqm_r.a	C용 클라이언트(스레드 기반 애플리케이션)

참고:

1. 둘 이상의 라이브러리에 링크할 수 없습니다. 즉, 동시에 스레드 및 비스레드 라이브러리 둘 모두에 링크할 수 없습니다.
2. 설치 가능 서비스를 작성하는 경우(추가 정보는 [관리 IBM MQ](#) 참조), 비스레드 애플리케이션의 `libmqmzf.a` 라이브러리 및 스레드 애플리케이션의 `libmqmzf_r.a` 라이브러리에 링크해야 합니다.
3. IBM TXSeries, Encina 또는 BEA Tuxedo와 같은 XA 준수 트랜잭션 관리자의 외부 조정을 위한 애플리케이션을 생성 중이면 `libmqmxa.a`(또는 트랜잭션 관리자가 'long' 유형을 64비트로 처리하는 경우 `libmqmxa64.a`) 및 비스레드 애플리케이션의 `libmqz.a` 라이브러리와 스레드 애플리케이션의 `libmqmxa_r.a`(또는 `libmqmxa64_r.a`) 및 `libmqz_r.a` 라이브러리에 링크해야 합니다.
4. 신뢰할 수 있는 애플리케이션을 스레드 IBM MQ 라이브러리에 링크해야 합니다. 그러나 IBM MQ for AIX or Linux 시스템에서 신뢰하는 애플리케이션에 있는 단 하나의 스레드만 한 번에 연결할 수 있습니다.
5. 다른 제품 라이브러리보다 먼저 IBM MQ 라이브러리를 링크해야 합니다.

이 태스크 정보

샘플 C 프로그램은 `${MQ_INSTALLATION_PATH}/samp/bin` 디렉터리에 제공됩니다.

다음 코드 예제 각각은 샘플 프로그램과 XL C/C++ amqsput0 컴파일러를 사용하며, Server for C 라이브러리 파일 중 하나와 링크합니다. 해당 프로그램이 설치된 IBM MQ MQI client for AIX 컴퓨터에서만 사용하려면, 예제 코드를 수정하여 클라이언트 라이브러리(-lmqic) 파일 중 하나와 연결하도록 변경하십시오.

C++ 프로그램에 C/C++ VisualAge 컴파일러를 사용하는 경우, 라이브러리 링크 시 모든 IBM MQ 심볼이 해결되도록 -q namemangling=v5 해당 옵션을 포함하십시오.

64비트 애플리케이션 작성에 대한 자세한 내용은 [64비트 플랫폼의 코딩 표준](#)을 참조하십시오.

예

단일 컴파일 유닛으로 샘플 C 프로그램을 컴파일하려면 다음 목록에서 적절한 명령을 실행하십시오. 이 명령어 예제는 AIX 컴파일러에 대해 IBM XL C/C++ 를 사용합니다.

Deprecated XL C/C++ 의 AIX 16 컴파일러 지원은 ([IBM MQ 9.4.0 Long Term Support](#) 릴리스에서는, 그리고 [IBM MQ 9.3.5 Continuous Delivery](#) 릴리스에서는) 더 이상 사용되지 않습니다. 이 컴파일러는 17 AIX 컴파일러용 Open XL C/C++ IBM 로 대체되었습니다.

V 9.4.2 **Removed** XL C/C++ 는 16 AIX 컴파일러용으로 에서 제거되었습니다 IBM MQ 9.4.2.

이 명령어 예시는 IBM Open XL C/C++ 를 사용하여 17 AIX 컴파일러를 대상으로 합니다:

32비트 비스레드 애플리케이션

```
ibm-clang -o amqsput_32 amqsput0.c -I${MQ_INSTALLATION_PATH}/inc -L${MQ_INSTALLATION_PATH}/lib -lmqm
```

32비트 스레드 애플리케이션

```
ibm-clang_r -o amqsput_32_r amqsput0.c -I${MQ_INSTALLATION_PATH}/inc -L$  
{MQ_INSTALLATION_PATH}/lib -lmqm_r
```

64비트 비스레드 애플리케이션

```
ibm-clang -m64 -o amqsput_64 amqsput0.c -I${MQ_INSTALLATION_PATH}/inc -L$  
{MQ_INSTALLATION_PATH}/lib64 -lmqm
```

64비트 스레드 애플리케이션

```
ibm-clang_r -m64 -o amqsput_64_r amqsput0.c -I${MQ_INSTALLATION_PATH}/inc -L$  
{MQ_INSTALLATION_PATH}/lib64 -lmqm_r
```

다음 명령어 예시는 16 AIX 컴파일러용 XL C/C++ 를 사용합니다:

32비트 비스레드 애플리케이션

```
xlc -o amqsput_32 amqsput0.c -I${MQ_INSTALLATION_PATH}/inc -L${MQ_INSTALLATION_PATH}/lib  
-lmqm
```

32비트 스레드 애플리케이션

```
xlc_r -o amqsputc_32_r amqsput0.c -I${MQ_INSTALLATION_PATH}/inc -L$  
{MQ_INSTALLATION_PATH}/lib -lmqm_r
```

64비트 비스레드 애플리케이션

```
xlc -q64 -o amqsput_64 amqsput0.c -I${MQ_INSTALLATION_PATH}/inc -L${MQ_INSTALLATION_PATH}/  
lib64 -lmqm
```

64비트 스레드 애플리케이션

```
xlc_r -q64 -o amqsputc_64_r amqsput0.c -I${MQ_INSTALLATION_PATH}/inc -L$  
{MQ_INSTALLATION_PATH}/lib64 -lmqm_r
```

AIX에서의 COBOL 프로그램 준비

COBOL 프로그램을 IBM COBOL Set 또는 Micro Focus COBOL AIX 에서 준비하고 컴파일하십시오.

시작하기 전에

다음 코드 예시에서는 가 IBM MQ 설치된 상위 \${MQ_INSTALLATION_PATH} 디렉터리를 나타냅니다. 코드를 변경하지 않고 그대로 사용하면 변수에 설정한 값을 \${MQ_INSTALLATION_PATH} 가져올 수 있으며, 또는 실제 경로로 \${MQ_INSTALLATION_PATH} 대체할 수도 있습니다.

- 32비트 COBOL 복사본이 디렉터리에 설치되며, 디렉터리에 \${MQ_INSTALLATION_PATH}/inc/cobcpy32 심볼릭 링크가 생성됩니다. \${MQ_INSTALLATION_PATH}/inc
- 64비트 COBOL 복사본 책들이 디렉터리에 설치됨 \${MQ_INSTALLATION_PATH}/inc/cobcpy64

프로그램은 다음 라이브러리 파일 중 하나와 링크해야 합니다.

라이브러리 파일	프로그램/엑시트 유형
libmqmcb.a	COBOL용 서버(비스레드 애플리케이션)
libmqmcb_r.a	COBOL용 서버(스레드 애플리케이션)
libmqicb.a	COBOL용 클라이언트(비스레드 애플리케이션)
libmqicb_r.a	COBOL용 클라이언트(스레드 애플리케이션)

이 태스크 정보

COBOL 샘플 프로그램이 함께 IBM MQ제공되며, 다음 코드 예제는 이러한 프로그램을 사용합니다. 프로그램에 따라 IBMCOBOL Set 컴파일러 또는 Micro Focus COBOL 컴파일러를 사용할 수 있습니다:

- amq0을 시작하는 프로그램은 두 컴파일러 모두에 적합합니다.
- 시작하는 프로그램은 Micro Focus COBOLamqm 컴파일러에 적합합니다.

예

COBOL 세트를 IBM 사용하여 COBOL 프로그램 준비 AIX

샘플 COBOL 프로그램을 컴파일하려면 다음 목록에서 적절한 명령을 실행하십시오:

32비트의 비스레드 서버 애플리케이션

```
cob2 -o amq0put0 amq0put0.cbl -L${MQ_INSTALLATION_PATH}/lib -lmqmb -qLIB \
-I${MQ_INSTALLATION_PATH}/inc/cobcpy32
```

32비트의 비스레드 클라이언트 애플리케이션

```
cob2 -o amq0put0 amq0put0.cbl -L${MQ_INSTALLATION_PATH}/lib -lmqicb -qLIB \
-I${MQ_INSTALLATION_PATH}/inc/cobcpy32
```

32비트의 스레드 서버 애플리케이션

```
cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L${MQ_INSTALLATION_PATH}/lib \
-lmqmb_r -qLIB -I${MQ_INSTALLATION_PATH}/inc/cobcpy32
```

32비트의 스레드 클라이언트 애플리케이션

```
cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L${MQ_INSTALLATION_PATH}/lib \
-lmqicb_r -qLIB -I${MQ_INSTALLATION_PATH}/inc/cobcpy32
```

64비트의 비스레드 서버 애플리케이션

```
cob2 -o amq0put0 amq0put0.cbl -q64 -L${MQ_INSTALLATION_PATH}/lib -lmqmb \
-qLIB -I${MQ_INSTALLATION_PATH}/inc/cobcpy64
```

64비트의 비스레드 클라이언트 애플리케이션

```
cob2 -o amq0put0 amq0put0.cbl -q64 -L${MQ_INSTALLATION_PATH}/lib -lmqicb \
-qLIB -I${MQ_INSTALLATION_PATH}/inc/cobcpy64
```

64비트의 스레드 서버 애플리케이션

```
cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L${MQ_INSTALLATION_PATH}/lib \
-lmqmb_r -qLIB -I${MQ_INSTALLATION_PATH}/inc/cobcpy64
```

64비트의 스레드 클라이언트 애플리케이션

```
cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L${MQ_INSTALLATION_PATH}/lib \
-lmqicb_r -qLIB -I${MQ_INSTALLATION_PATH}/inc/cobcpy64
```

Micro Focus를 사용하여 COBOL 프로그램 준비하기 COBOL

설정해야 하는 환경 변수에 대한 설명은 Micro Focus COBOL 문서에서 참조하십시오.

Micro Focus COBOL 을 사용하여 32비트 COBOL 프로그램을 컴파일하려면 먼저 COBCPY 환경 변수를 설정하십시오:

```
export COBCPY=${MQ_INSTALLATION_PATH}/inc/cobcpy32
```

그런 다음 다음 목록에서 적절한 명령을 입력하십시오:

32비트의 비스레드 서버 애플리케이션

```
cob32 -xvP amqminqx.cbl -L${MQ_INSTALLATION_PATH}/lib -lmqmcbl
```

32비트의 비스레드 클라이언트 애플리케이션

```
cob32 -xvP amqminqx.cbl -L${MQ_INSTALLATION_PATH}/lib -lmqicbl
```

32비트의 스레드 서버 애플리케이션

```
cob32 -xtvP amqminqx.cbl -L${MQ_INSTALLATION_PATH}/lib -lmqmcbl_r
```

32비트의 스레드 클라이언트 애플리케이션

```
cob32 -xtvP amqminqx.cbl -L${MQ_INSTALLATION_PATH}/lib -lmqicbl_r
```

Micro Focus COBOL 을 사용하여 64비트 COBOL 프로그램을 컴파일하려면 먼저 COBCPY 환경 변수를 설정하십시오:

```
export COBCPY=${MQ_INSTALLATION_PATH}/inc/cobcpy64
```

그런 다음 다음 목록에서 적절한 명령을 입력하십시오:

64비트의 비스레드 서버 애플리케이션

```
cob64 -xvP amqminqx.cbl -L${MQ_INSTALLATION_PATH}/lib64 -lmqmcbl
```

64비트의 비스레드 클라이언트 애플리케이션

```
cob64 -xvP amqminqx.cbl -L${MQ_INSTALLATION_PATH}/lib64 -lmqicbl
```

64비트의 스레드 서버 애플리케이션

```
cob64 -xtvP amqminqx.cbl -L${MQ_INSTALLATION_PATH}/lib64 -lmqmcbl_r
```

64비트의 스레드 클라이언트 애플리케이션

```
cob64 -xtvP amqminqx.cbl -L${MQ_INSTALLATION_PATH}/lib64 -lmqicbl_r
```

응용 프로그램 CICS 준비 AIX

XA 스위치 모듈을 사용하여 CICS를 IBM MQ와 링크합니다.

이 태스크 정보

이 제품에는 미리 빌드된 스위치 로드 파일이 amqzscIBM MQ 포함되어 있으며, 이 파일은 XA 초기화 루틴을 제공합니다. 이 제품에는 다른 트랜잭션 메시지에 대한 XA 스위치를 개발하는 데 도움이 되도록 샘플 소스 코드 파일도 amqzscix.c 함께 제공됩니다.

프로시저

1. 사전 빌드된 스위치 로드 파일 amqzsc (CICS for AIX)을 귀하의 XAD.Stanza.에 추가하십시오.

이 스위치는 `XA` 초기화 루틴을 제공합니다.

2. 다른 트랜잭션 메시지에 대한 `XA` 스위치를 구현하십시오.

이 작업을 수행하는 데 도움이 되도록 amqzscix.c 샘플 소스 코드 파일을 사용하십시오.

C 트랜잭션을 항상 스레드 안전 IBM MQ 라이브러리 (*libmqm_r.a*) 와 연결하십시오 그리고 COBOL 라이브러리 (*libmqmcb_r.a*) 를 사용한 COBOL 트랜잭션

[XA 스위치 구조에](#) 대해서도 참조하십시오.

IBM MQ on은 AIX `XA` 인터페이스를 사용하여 `COBOL` 또는 C로 작성된 프로그램을 CICS 지원합니다
TXSeries.

시작하기 전에

다음 코드 예시에서는 가 IBM MQ 설치된 상위 `${MQ_INSTALLATION_PATH}` 디렉터리를 나타냅니다. 코드를 변경하지 않고 그대로 사용하면 변수에 설정한 값을 `${MQ_INSTALLATION_PATH}` 가져올 수 있으며, 또는 실제 경로로 `${MQ_INSTALLATION_PATH}` 대체할 수도 있습니다.

C 트랜잭션을 항상 스레드 안전 IBM MQ 라이브러리 (`libmqm.r.a`) 와 연결하십시오 그리고 COBOL 라이브러리 (`libmqmcb.r.a`) 를 사용한 COBOL 트랜잭션

C 또는 COBOL 로 작성된 IBM MQ 프로그램을 동일한 CICS 영역에 로드하십시오. 동일한 CICS 리전에서 C 및 COBOL MQI 호출을 결합할 수 없습니다. 두 번째 언어로 사용된 대부분의 MQI 호출은 이유 코드 MQRC_HOBJ_ERROR와 함께 실패합니다.

이 태스크 정보

IBM COBOL Set for AIX 또는 Micro Focus COBOL을 사용하여 CICS을(를) 실행할 수 있습니다. C CICS 프로그램도 실행할 수 있습니다.

예

COBOL 프로그램을 CICS 준비 중입니다. IBMCOBOL Set for AIX

1. 다음과 같은 환경 변수를 내보내십시오.

```
export LDFLAGS="-qLIB -bI:/usr/lpp/cics/lib/cicsprIBMCOB.exp \
-I${MQ_INSTALLATION_PATH}/inc -I/usr/lpp/cics/include \
-e _iwz_cobol_main \
```

여기서 LIB는 컴파일러 지시문입니다.

2. 프로그램을 번역하고, 컴파일하고, 링크하십시오:

```
cicstcl -l IBMCOB yourprog.ccp
```

COBOL CICS 프로그램 준비하기 (Micro Focus 사용) COBOL

1. COBOL 런타임 IBM MQ 라이브러리 모듈을 런타임 라이브러리에 추가하십시오:

```
cicsmkcobol -L/usr/lib/dce -L${MQ_INSTALLATION_PATH}/lib \
${MQ_INSTALLATION_PATH}/lib/libmqmcb.r.a -lmqe_r
```

참고: `cicsmkcobol`을 사용하면, IBM MQ는 COBOL 애플리케이션에서 C 프로그래밍 언어로 MQI 호출을 수행하도록 허용하지 않습니다.

기존 애플리케이션에 이와 같은 호출이 있다면, 해당 함수들을 COBOL 애플리케이션에서 자체 라이브러리 (예: `myMQ.so`.)로 이동하십시오. 함수를 이동한 후, COBOL 애플리케이션을 빌드할 때 IBM `MQlibmqmcb.r.a` 라이브러리를 CICS 포함하지 마십시오.

`cicsmkcobol` COBOL 애플리케이션에서 COBOL MQI 호출을 전혀 수행하지 않는다면, 와 링크하지 `libmqmz_r` 마십시오.

그러면 Micro Focus COBOL 언어 메소드 파일이 작성되고 CICS 런타임 COBOL 라이브러리가 IBM MQ for AIX or Linux 시스템을 호출할 수 있습니다.

참고: 다음 제품 중 하나를 설치하는 경우에만 `cicsmkcobol`을 실행하십시오.

- Micro Focus COBOL의 새 버전 또는 릴리스
- 의 CICS 새 버전 또는 출시 AIX
- 지원되는 데이터베이스 제품의 새 버전 또는 릴리스(COBOL 트랜잭션에만 해당)

- IBM MQ의 새 버전 또는 릴리스
2. 다음과 같은 환경 변수를 내보내십시오.

```
COBCPY=${MQ_INSTALLATION_PATH}/inc export COBCPY
```

3. 프로그램을 번역하고, 컴파일하고, 링크하십시오:

```
cicstcl -l COBOL -e yourprog.ccp
```

C CICS 프로그램 준비하기

표준 CICS 기능을 사용하여 CICS C 프로그램을 빌드하십시오.

1. 다음 환경 변수 중 하나를 내보내십시오.

- `LDFLAGS="-L${MQ_INSTALLATION_PATH}/lib -lmqm_r" export LDFLAGS`

- `USERLIB="-L${MQ_INSTALLATION_PATH}/lib -lmqm_r" export USERLIB`

2. 프로그램을 번역하고, 컴파일하고, 링크하십시오:

```
cicstcl -l C amqscic0.ccs
```

CICS C 샘플 트랜잭션

트랜잭션에 AIXIBM MQ 대한 C 소스 샘플은 에서 제공됩니다 AMQSCIC0.CCS. 해당 트랜잭션은 기본 큐 관리자의 전송 SYSTEM.SAMPLE.CICS.WORKQUEUE 큐에서 메시지를 읽고, 메시지 전송 헤더에 포함된 큐 이름을 사용하여 로컬 큐에 배치합니다. 모든 실패는 대기열로 전송됩니다 SYSTEM.SAMPLE.CICS.DLQ. 샘플 MQSC 스크립트를 사용하여 이러한 큐와 AMQSCIC0.TST 샘플 입력 큐를 생성하십시오.

IBM i IBM i에서 절차적 애플리케이션 빌드

이 IBM i 출판물들은 작성한 프로그램으로부터 실행 가능한 애플리케이션을 구축하여 또는 System i IBM iSeries 시스템에서 실행하는 방법을 설명합니다.

이 항목에서는 시스템에서 IBM i 실행될 절차적 IBM MQ for IBM i 애플리케이션을 빌드할 때 수행해야 하는 추가 작업과 표준 작업에 대한 변경 사항을 설명합니다. COBOL, C, C++, Java 및 RPG 프로그래밍 언어가 지원됩니다. C++ 프로그램 준비에 대한 정보는 [C++ 사용의 내용](#)을 참조하십시오. Java 프로그램 준비에 대한 정보는 [사용 IBM MQ classes for Java의 내용](#)을 참조하십시오.

실행 가능한 IBM MQ for IBM i 애플리케이션을 작성하기 위해 수행해야 할 태스크는 소스 코드가 기록된 프로그래밍 언어에 따라 다릅니다. 소스 코드로 MQI 호출을 코딩하는 것 외에도 사용 중인 언어에 대해 IBM MQ for IBM i 데이터 정의 파일을 포함하도록 적절한 언어문을 추가해야 합니다. 이러한 파일의 콘텐츠를 숙지하십시오. 전체 설명은 677 페이지의 『IBM MQ 데이터 정의 파일』의 내용을 참조하십시오.

IBM i IBM i에서 C 프로그램 준비

IBM MQ for IBM i는 최대 100MB 크기의 메시지를 지원합니다. 16MB보다 큰 IBM MQ 메시지를 지원하며 ILE C로 기록된 애플리케이션 프로그램은 테라스페이스 컴파일러 옵션을 사용하여 이러한 메시지에 대해 충분한 메모리를 할당해야 합니다.

C 컴파일러 옵션에 대한 자세한 정보는 *WebSphere Development Studio ILE C/C++ 프로그래머 안내서*를 참조하십시오.

C 모듈을 컴파일하기 위해 IBM i 명령 **CRTCMOD**를 사용할 수 있습니다. 컴파일할 때 포함 파일(QMQM)을 포함하는 라이브러리가 라이브러리 목록에 있는지 확인하십시오.

그런 다음, **CRTPGM** 명령을 사용하여 서비스 프로그램과 컴파일러의 출력을 바인딩해야 합니다.

표 153. 비스레드 및 스레드 환경에서 CRTPGM 예제		
환경 유형	명령	프로그램/엑시트 유형
비스레드 환경	CRTPGM PGM(<i>pgmname</i>) MODULE(<i>pgmname</i>) BNDSRVPGM(QMQM/LIBMQM)	C용 서버 또는 클라이언트
스레드 환경	CRTPGM PGM(<i>pgmname</i>) MODULE(<i>pgmname</i>) BNDSRVPGM(QMQM/LIBMQM_R)	C용 서버 또는 클라이언트

여기서 *pgmname*은 프로그램의 이름입니다.

944 페이지의 표 154에는 비스레드 환경 및 스레드 환경의 IBM i에서 C 프로그램을 준비할 때 필요한 라이브러리가 나열되어 있습니다.

표 154. 비스레드 및 스레드 환경에 필요한 라이브러리		
환경 유형	라이브러리 파일	프로그램/엑시트 유형
비스레드 환경	LIBMQM	C용 서버
	LIBMQIC & LIBMQM	C용 클라이언트
스레드 환경	LIBMQM_R	C용 서버
	LIBMQIC_R & LIBMQM_R	C용 클라이언트

IBM i IBM i에서의 COBOL 프로그램 준비

IBM i에서 COBOL 프로그램을 준비하는 방법과 COBOL 프로그램에서 MQI에 액세스하는 방법에 대해 알아보니다.

이 태스크 정보

COBOL 프로그램에서 MQI에 액세스하기 위해 IBM MQ for IBM i는 서비스 프로그램에 의해 제공된 바인드된 프로시저 호출 인터페이스를 제공합니다. 이 인터페이스는 IBM MQ for IBM i의 모든 MQI 기능에 액세스하고 스레드 애플리케이션을 지원합니다. 이 인터페이스는 ILE COBOL 컴파일러에서만 사용할 수 있습니다.

표준 COBOL CALL 구문은 MQI 기능에 액세스하는 데 사용됩니다.

MQI에서 사용할 이름 지정된 상수 및 구조 정의를 포함하는 COBOL 사본 파일은 소스 실제 파일 QMQM/QCBLLESRC에 포함되어 있습니다.

COBOL 사본 파일은 작은따옴표 문자(')를 문자열 구분 기호로 사용합니다. COBOL IBM i 컴파일러는 구분자로 따옴표(")를 가정합니다. 컴파일러가 경고 메시지를 생성하지 않도록 하려면, **CRTBNDCL**, 또는 명령어에 **OPTION(CRTCBMOD*APOST CRTCBPLPGM)**를 지정하십시오.

컴파일러가 작은따옴표 문자(')를 COBOL 사본 파일의 문자열 구분 기호로 허용하게 하려면 컴파일러 옵션 \APOST를 사용하십시오.

참고: 동적 호출 인터페이스는 IBM MQ 9.0 또는 이상에서 제공되지 않습니다.

바인드된 프로시저 호출 인터페이스를 사용하려면 다음 단계를 완료하십시오.

프로시저

1. 매개변수를 지정하는 **CRTCBMOD** 컴파일러를 사용하여 모듈을 작성하십시오.

```
LINKLIT(*PRC)
```

2. 적절한 매개변수를 지정하면서, 프로그램 오브젝트를 작성하려면 **CRTPGM** 명령을 사용하십시오.

비스레드 애플리케이션의 경우:

BNDSRVPGM(QMQM/AMQ0STUB)
BNDSRVPGM(QMQM/AMQCSTUB)

Server for COBOL for non-threaded applications
Client for COBOL for non-threaded applications

스레드 애플리케이션의 경우:

BNDSRVPGM(QMQM/AMQ0STUB_R)
BNDSRVPGM(QMQM/AMQCSTUB_R)

Server for COBOL for threaded applications
Client for COBOL for threaded applications

참고: V4R4 ILE COBOL 컴파일러를 사용하여 작성되고 PROCESS문에 THREAD(SERIALIZE) 옵션을 포함하는 프로그램을 제외하고, COBOL 프로그램은 스레드된 IBM MQ 라이브러리를 사용해야 합니다. COBOL 프로그램이 이런 식으로 스레드에 안전하게 작성된 경우에도 애플리케이션을 설계할 때 주의해야 합니다. THREAD(SERIALIZE)가 모듈 레벨에서 COBOL 프로시저의 직렬화를 강제 실행하고 전체 성능에 영향을 줄 수 있기 때문입니다.

자세한 정보는 *WebSphere Development Studio: ILE COBOL* 프로그래머용 안내서 및 *WebSphere Development Studio: ILE COBOL* 참조서의 내용을 참조하십시오.

CICS 애플리케이션 컴파일에 대한 자세한 정보는 *CICS for IBM i Application Programming Guide*, SC41-5454의 내용을 참조하십시오.

IBM i IBM i에서 CICS 프로그램 준비

프로그램을 CICS 준비할 때 필요한 IBM i 단계에 대해 알아보세요.

EXEC CICS 명령문과 MQI 호출을 포함하는 프로그램을 작성하려면 다음 단계를 수행하십시오.

1. 필요한 경우, CRTICSMAP 명령을 사용하여 맵을 준비하십시오.
2. EXEC CICS 명령을 고유 언어문으로 변환하십시오. C 프로그램에 CRTICSC 명령을 사용하십시오. COBOL 프로그램에 CRTICSCBL 명령을 사용하십시오.

CRTCCICS 또는 CICSRTCBL 명령에 CICSOPT(*NOGEN) 포함하십시오. 이렇게 하면 적절한 CICS 및 IBM MQ 서비스 프로그램을 포함할 수 있도록 처리가 정지됩니다. 이 명령은 기본적으로 QTEMP/QACYCICS에 코드를 넣습니다.

3. CRTCMOD 명령(C 프로그램의 경우) 또는 CRTCLMOD 명령(COBOL 프로그램의 경우)을 사용하여 소스 코드를 컴파일하십시오.
4. CRTPGM을 사용하여 컴파일된 코드를 적절한 CICS 및 IBM MQ 서비스 프로그램과 링크하십시오. 이렇게 하면 실행 가능 프로그램이 작성됩니다.

이러한 코드의 예는 다음과 같습니다(제공된 CICS 샘플 프로그램을 컴파일함).

```
CRTICSC OBJ(QTEMP/AMQSCIC0) SRCFILE(/MQSAMP/QCSRC) +  
      SRCMBR(AMQSCIC0) OUTPUT(*PRINT) +  
      CICSOPT(*SOURCE *NOGEN)  
CRTCMOD MODULE(MQTEST/AMQSCIC0) +  
      SRCFILE(QTEMP/QACYCICS) OUTPUT(*PRINT)  
CRTPGM PGM(MQTEST/AMQSCIC0) MODULE(MQTEST/AMQSCIC0) +  
      BNDSRVPGM(QMQM/LIBMQIC QCICS/AEGEIPGM)
```

IBM i IBM i에서의 RPG 프로그램 준비

IBM i에서 실행될 RPG 애플리케이션을 구축할 때는 소스 코드에 'MQI' 호출을 작성하는 것 외에도, RPG 언어용 'copy' 파일을 IBM MQ for IBM i 포함시키기 위해 적절한 언어 지시문을 추가해야 합니다.

- 945 페이지의 『IBM MQ 복사 파일』
- 946 페이지의 『실행할 프로그램 준비』
- 946 페이지의 『IBM i 외부 동기점 관리자에 대한 인터페이스』
- 947 페이지의 『CICS for IBM i 애플리케이션의 동기점』

IBM MQ 복사 파일

IBM MQ for IBM i는 RPG 프로그래밍 언어에서 애플리케이션 작성을 도와주는 복사 파일을 제공합니다. 이 파일은 WebSphere Development 도구 세트(5722 WDS) ILE RPG 4 컴파일러에 사용하기에 적합합니다.

채널 이탈 메시지 작성을 지원하기 위해 제공하는 복사 IBM MQ for IBM i 파일은 [에서 설명합니다 710 페이지의 『메시지 채널에 대한 채널 엑시트 프로그램』](#).

RPG용 IBM MQ for IBM i 복사 파일의 이름에는 접두부 CMQ가 붙습니다. G 또는 H의 접미부가 있습니다. 이름 지정된 상수를 포함하는 개별 사본 파일과 각 구조에 대한 하나의 파일이 있습니다. 복사 파일은 [언어 관련 사항](#)에 나열되어 있습니다.

참고: ILE RPG/400의 경우, 해당 파일은 파일의 멤버로 제공됩니다.라이브러리 QMQM의 QRPGLSRC입니다.

구조 선언은 DS 명령문을 포함하지 않습니다. 이를 통해 응용 프로그램은 DS 문으로 데이터 구조(또는 다중 발생 데이터 구조)를 선언하고 /COPY 문을 사용하여 선언의 나머지 부분을 복사할 수 있습니다.

ILE RPG/400의 경우 명령문은 다음과 같습니다.

```
D*.1.....2.....3.....4.....5.....6.....7
D* Declare an MQMD data structure
D MQMD          DS
D/COPY CMQMDG
```

실행할 프로그램 준비

실행 가능 IBM MQ for IBM i 애플리케이션을 작성하려면 작성한 소스 코드를 컴파일해야 합니다. ILE RPG/400에 이 작업을 수행하려면 일반 IBM i 명령, CRTRPGMOD 및 CRTPGM을 사용할 수 있습니다.

*MODULE를 작성한 후, CRTPGM 명령에 BNDSRVPGM(QMQM/LIBMQM)을 지정해야 합니다. 여기에는 프로그램의 다양한 IBM MQ 프로시저가 포함됩니다.

컴파일을 수행할 때는 복사 파일(QMQM)이 들어 있는 라이브러리가 라이브러리 목록에 있는지 확인하십시오.

프로그래밍 관련 고려 사항(클라이언트 모드 포함)에 대한 자세한 내용은 [언어 관련 고려 사항](#)을 참조하십시오.

IBM i 외부 동기점 관리자에 대한 인터페이스

IBM MQ for IBM i는 기본 IBM i 커밋 제어를 외부 동기점 조정자로 사용합니다.

IBM i 커밋 제어 기능을 시작하려면 STRCMTCTL 시스템 명령을 사용하십시오. 커밋 제어를 종료하려면 ENDCMTCTL 시스템 명령을 사용하십시오.

참고: 커밋 정의 범위의 기본값은 *ACTGRP입니다. IBM MQ for IBM i에는 *JOB으로 정의되어야 합니다. 예를 들면, 다음과 같습니다.

```
STRCMTCTL LCKLVL(*ALL) CMTSCOPE(*JOB)
```

커밋 제어를 시작한 후 MQPUT, MQPUT1 또는 MQGET을 호출하고 PMSYP 또는 GMSYP를 지정하는 경우, IBM MQ for IBM i가 자체적으로 커밋 정의에 API 커밋 자원으로 추가됩니다. 이는 일반적으로 작업에서 최초의 호출입니다. 특정 커밋 정의에 따라 등록된 API 커밋 자원이 있는 동안은 해당 정의에 대한 커밋 제어를 종료할 수 없습니다.

현재 작업 단위에 보류 중인 MQI 작업이 없는 경우, 큐 관리자와 연결을 끊으면 IBM MQ for IBM i에서는 등록을 API 커밋 자원으로 제거합니다.

현재 작업 단위에 보류 중인 MQPUT, MQPUT1 또는 MQGET 조작이 있는 동안 큐 관리자로부터 연결을 끊은 경우에는 IBM MQ for IBM i가 여전히 API 커밋 자원으로 등록되어 있으므로 다음 커밋 또는 롤백에 대한 알림을 받습니다. 다음 동기점에 도달하면, IBM MQ는 필요에 따라 변경사항을 커밋하거나 롤백합니다. 애플리케이션이 활성 작업 단위 동안 큐 관리자의 연결을 끊었다가 다시 연결하고 동일한 작업 단위 내에서 추가 MQGET 및 MQPUT 조작을 수행할 수 있습니다(보류 중인 연결 끊기).

해당 커밋 정의에 대한 ENDCMTCTL 시스템 명령을 실행하려고 시도하면, 보류 중인 변경이 활성화되었음을 나타내는 메시지 CPF8355가 발행됩니다. 이 메시지는 작업이 끝날 때에도 작업 로그에 표시됩니다. 이러한 상황을 피하려면 모든 보류 중인 IBM MQ 조작을 커밋 또는 롤백하고 큐 관리자와 연결을 끊어야 합니다. 따라서 성공적으로 완료하기 위해 ENDCMTCTL 앞에 COMMIT 또는 ROLLBACK 명령을 사용하여 커밋 제어 종료를 사용하도록 설정해야 합니다.

IBM i 커밋 제어가 동기점 조정자로 사용되면 MQCMIT, MQBACK 및 MQBEGIN 호출이 발행되지 않을 수 있습니다. 이 함수의 호출이 이유 코드 RC2012와 함께 실패합니다.

작업 단위를 커밋하거나 롤백하려면 커밋 제어를 지원하는 프로그래밍 언어 중 하나를 사용하십시오. 예를 들면, 다음과 같습니다.

- CL 명령: COMMIT 및 ROLLBACK
- ILE C 프로그래밍 함수: _Rcommit 및 _Rrollback
- RPG/400: COMMIT 및 ROLBK
- COBOL/400: COMMIT 및 ROLLBACK

CICS for IBM i 애플리케이션의 동기점

IBM MQ for IBM i는 CICS와 함께 작업 단위에 참여합니다. CICS 애플리케이션에서 MQI를 사용하여 현재 작업 단위 내부에 메시지를 넣고 가져올 수 있습니다.

EXEC CICS SYNCPOINT 명령을 사용하여 IBM MQ for IBM i 조작을 포함하는 동기점을 설정할 수 있습니다. 이전 동기점까지 모든 변경사항을 백아웃하려면 EXEC CICS SYNCPOINT ROLLBACK 명령을 사용할 수 있습니다.

애플리케이션에서 PMSYP 또는 GMSYP 옵션이 설정된 상태로 MQPUT, MQPUT1 또는 MQGET을 사용하는 경우, 해당 CICS 애플리케이션이 API 커밋먼트 리소스로서의 등록을 해제할 IBM MQ for IBM i 때까지 CICS 로그 오프할 수 없습니다. 따라서 큐 관리자와 연결을 끊기 전에 보류 중인 Put 또는 Get 조작을 커밋하거나 백업해야 합니다. 그러면 CICS에서 로그오프할 수 있습니다.

Linux Linux에서 절차적 애플리케이션 빌드

이 정보는 Linux 애플리케이션을 실행하기 위해 IBM MQ를 빌드할 때 수행해야 하는 추가 태스크 및 표준 태스크에 대한 변경사항을 설명합니다.

C 및 C++가 지원됩니다. C++ 프로그램 준비에 대한 정보는 [C++ 사용의 내용](#)을 참조하십시오.

Linux Linux에서 C 프로그램 준비

사전 컴파일된 C 프로그램은 `MQ_INSTALLATION_PATH/samp/bin` 디렉토리에서 제공됩니다. 소스 코드에서 샘플을 빌드하려면 gcc 컴파일러를 사용하십시오.

`MQ_INSTALLATION_PATH`은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

정상적인 환경에서 작업하십시오. 64비트 애플리케이션 프로그래밍에 대한 추가 정보는 [64비트 플랫폼의 코딩 표준](#)을 참조하십시오.

라이브러리 링크

다음 표에는 Linux에서 C 프로그램으로 준비할 때 필요한 라이브러리가 나열되어 있습니다.

- 프로그램은 IBM MQ에서 제공하는 적절한 라이브러리와 링크해야 합니다.

비스레드 환경에서 다음 라이브러리 중 하나에만 링크하십시오.

라이브러리 파일	프로그램/엑시트 유형
libmqm.so	C용 서버
libmqic.so & libmqm.so	C용 클라이언트

스레드 환경에서 다음 라이브러리 중 하나에만 링크하십시오.

라이브러리 파일	프로그램/엑시트 유형
libmqm_r.so	C용 서버
libmqic_r.so & libmqm_r.so	C용 클라이언트

참고:

1. 둘 이상의 라이브러리에 링크할 수 없습니다. 즉, 동시에 스레드 및 비스레드 라이브러리 둘 모두에 링크할 수는 없습니다.

2. 설치 가능 서비스를 작성하는 경우(추가 정보는 [관리 IBM MQ](#) 참조) libmqmzf.so 라이브러리에 링크해야 합니다.
3. IBM TXSeries Encina 또는 BEA Tuxedo와 같은 XA 준수 트랜잭션 관리자의 외부 조정을 위한 애플리케이션을 생성 중이면 libmqmxa.so(또는 트랜잭션 관리자가 'long' 유형을 64비트로 처리하는 경우 libmqmxa64.so) 및 비스레드 애플리케이션의 libmqz.so 라이브러리와 스레드 애플리케이션의 libmqmxa_r.so(또는 libmqmxa64_r.so) 및 libmqz_r.so 라이브러리에 링크해야 합니다.
4. 다른 제품 라이브러리보다 먼저 IBM MQ 라이브러리를 링크해야 합니다.

Linux 32비트 애플리케이션 구축 Linux

이 주제에는 다양한 환경에서 32비트 프로그램을 빌드하는 데 사용되는 명령의 예가 있습니다.

`MQ_INSTALLATION_PATH`은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

C 클라이언트 애플리케이션, 32비트, 스레드되지 않음

```
gcc -m32 -o amqsputc_32 amqsputc0.c -I${MQ_INSTALLATION_PATH}/inc -L${MQ_INSTALLATION_PATH}/lib
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=/usr/lib -lmqic
```

C 클라이언트 애플리케이션, 32비트, 스레드됨

```
gcc -m32 -o amqsputc_32_r amqsputc0.c -I${MQ_INSTALLATION_PATH}/inc -L$
${MQ_INSTALLATION_PATH}/lib
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C 서버 애플리케이션, 32비트, 스레드되지 않음

```
gcc -m32 -o amqsput_32 amqsput0.c -I${MQ_INSTALLATION_PATH}/inc -L${MQ_INSTALLATION_PATH}/lib
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=/usr/lib -lmqm
```

C 서버 애플리케이션, 32비트, 스레드됨

```
gcc -m32 -o amqsput_32_r amqsput0.c -I${MQ_INSTALLATION_PATH}/inc -L${MQ_INSTALLATION_PATH}/lib
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

C++ 클라이언트 애플리케이션, 32비트, 스레드되지 않음

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsputc.cpp -I${MQ_INSTALLATION_PATH}/inc
-L${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

C++ 클라이언트 애플리케이션, 32비트, 스레드됨

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsputc.cpp -I${MQ_INSTALLATION_PATH}/inc
-L${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

C++ 서버 애플리케이션, 32비트, 스레드되지 않음

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I${MQ_INSTALLATION_PATH}/inc
-L${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

C++ 서버 애플리케이션, 32비트, 스레드되지 않음

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I${MQ_INSTALLATION_PATH}/inc
-L${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

C 클라이언트 엑시트, 32비트, 스레드되지 않음

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I${MQ_INSTALLATION_PATH}/inc -L${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=$
${MQ_INSTALLATION_PATH}/lib
-Wl,-rpath=/usr/lib -lmqic
```

C 클라이언트 엑시트, 32비트, 스레드됨

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c  
-I${MQ_INSTALLATION_PATH}/inc -L${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=${  
MQ_INSTALLATION_PATH}/lib  
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C 서버 엑시트, 32비트, 스레드되지 않음

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c -I${MQ_INSTALLATION_PATH}/inc  
-L${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=${MQ_INSTALLATION_PATH}/lib  
-Wl,-rpath=/usr/lib -lmqm
```

C 서버 엑시트, 32비트, 스레드됨

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c -I${MQ_INSTALLATION_PATH}/inc  
-L${MQ_INSTALLATION_PATH}/lib -Wl,-rpath=${MQ_INSTALLATION_PATH}/lib  
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux *Linux*의 64비트 애플리케이션 빌드

이 주제에는 다양한 환경에서 64비트 프로그램을 빌드하는 데 사용되는 명령의 예가 있습니다.

*MQ_INSTALLATION_PATH*은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

C 클라이언트 애플리케이션, 64비트, 스레드되지 않음

```
gcc -m64 -o amqsputc_64 amqsput0.c  
-I${MQ_INSTALLATION_PATH}/inc  
-L${MQ_INSTALLATION_PATH}/lib64  
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib64  
-Wl,-rpath=/usr/lib64 -lmqic
```

C 클라이언트 애플리케이션, 64비트, 스레드됨

```
gcc -m64 -o amqsputc_64_r amqsput0.c  
-I${MQ_INSTALLATION_PATH}/inc  
-L${MQ_INSTALLATION_PATH}/lib64  
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib64  
-Wl,-rpath=/usr/lib64 -lmqic_r -lpthread
```

C 서버 애플리케이션, 64비트, 스레드되지 않음

```
gcc -m64 -o amqsput_64 amqsput0.c  
-I${MQ_INSTALLATION_PATH}/inc  
-L${MQ_INSTALLATION_PATH}/lib64  
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib64  
-Wl,-rpath=/usr/lib64 -lmqm
```

C 서버 애플리케이션, 64비트, 스레드됨

```
gcc -m64 -o amqsput_64_r amqsput0.c  
-I${MQ_INSTALLATION_PATH}/inc  
-L${MQ_INSTALLATION_PATH}/lib64  
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib64  
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

C++ 클라이언트 애플리케이션, 64비트, 스레드되지 않음

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp  
-I${MQ_INSTALLATION_PATH}/inc  
-L${MQ_INSTALLATION_PATH}/lib64  
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib64  
-Wl,-rpath=/usr/lib64 -limqc23gl -limqb23gl -lmqic
```

C++ 클라이언트 애플리케이션, 64비트, 스레드됨

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp  
-I${MQ_INSTALLATION_PATH}/inc  
-L${MQ_INSTALLATION_PATH}/lib64  
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib64  
-Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

C++ 서버 애플리케이션, 64비트, 스레드되지 않음

```
g++ -m64 -fsigned-char -o imqspout_64 imqspout.cpp
-I${MQ_INSTALLATION_PATH}/inc
-L${MQ_INSTALLATION_PATH}/lib64
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

C++ 서버 애플리케이션, 64비트, 스레드됨

```
g++ -m64 -fsigned-char -o imqspout_64_r imqspout.cpp
-I${MQ_INSTALLATION_PATH}/inc
-L${MQ_INSTALLATION_PATH}/lib64
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

C 클라이언트 엑시트, 64비트, 스레드되지 않음

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I${MQ_INSTALLATION_PATH}/inc
-L${MQ_INSTALLATION_PATH}/lib64
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib64
-Wl,-rpath=/usr/lib64 -lmqic
```

C 클라이언트 엑시트, 64비트, 스레드됨

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64_r cliexit.c
-I${MQ_INSTALLATION_PATH}/inc
-L${MQ_INSTALLATION_PATH}/lib64
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib64
-Wl,-rpath=/usr/lib64 -lmqic_r -lpthread
```

C 서버 엑시트, 64비트, 스레드되지 않음

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I${MQ_INSTALLATION_PATH}/inc
-L${MQ_INSTALLATION_PATH}/lib64
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib64
-Wl,-rpath=/usr/lib64 -lmqm
```

C 서버 엑시트, 64비트, 스레드됨

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64_r srvexit.c
-I${MQ_INSTALLATION_PATH}/inc
-L${MQ_INSTALLATION_PATH}/lib64
-Wl,-rpath=${MQ_INSTALLATION_PATH}/lib64
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

Linux **Linux에서의 COBOL 프로그램 준비**

COBOL 프로그램 준비 방법과 COBOL 를 사용하여 LinuxCOBOLIBM 프로그램 준비 방법에 대해 x86 및 Micro Focus COBOLLinux 에서 확인하세요.

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

1. 32비트 COBOL 복사본은 다음 디렉터리에 설치됩니다:

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

그리고 기호 링크는 다음 디렉토리에서 작성됩니다.

```
MQ_INSTALLATION_PATH/inc
```

2. 64비트 플랫폼에서는 64비트 버전의 'COBOL' 복사본이 다음 디렉터리에 설치됩니다:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

3. 다음 예제에서 COBCPY는 32비트 애플리케이션의 경우 다음으로 설정하십시오.

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

32비트 애플리케이션의 경우, 그리고:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

64비트 애플리케이션용.

프로그램은 다음 중 하나와 링크해야 합니다.

라이브러리 파일	프로그램/엑시트 유형
libmqmcb.so	COBOL용 서버
libmqicb.so	COBOL용 클라이언트
libmqmcb_r.so	COBOL용 서버(스레드 애플리케이션)
libmqicb_r.so	COBOL용 클라이언트(스레드 애플리케이션)

x86 에서 Linux COBOLIBM 를 사용하여 COBOL 프로그램 준비

샘플 COBOL 프로그램은 IBMMQ 에서 제공됩니다. 이러한 프로그램을 컴파일하려면 다음 목록에서 적절한 명령을 입력하십시오.

32비트 비스레드 서버 애플리케이션

```
cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"  
-L${MQ_INSTALLATION_PATH}/lib -lmqmcb  
-ICOBPCPY_VALUE
```

32비트 비스레드 클라이언트 애플리케이션

```
cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"  
-L${MQ_INSTALLATION_PATH}/lib -lmqicb  
-ICOBPCPY_VALUE
```

32비트 스레드 기반 서버 애플리케이션

```
cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)" -qTHREAD  
-L${MQ_INSTALLATION_PATH}/lib -lmqmcb_r  
-ICOBPCPY_VALUE
```

32비트 스레드형 클라이언트 애플리케이션

```
cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)" -qTHREAD  
-L${MQ_INSTALLATION_PATH}/lib -lmqicb_r  
-ICOBPCPY_VALUE
```

Micro Focus COBOL을 사용한 COBOL 프로그램 준비

필요한 환경 변수에 대한 설명은 Micro Focus COBOL 문서를 참조하십시오.

프로그램 컴파일 전에 환경 변수를 설정하십시오:

```
export COBCPY=COBCPY_VALUE  
export LIB=${MQ_INSTALLATION_PATH} lib:$LIB
```

Micro Focus COBOL 을 사용하여 32비트 COBOL 프로그램(지원되는 환경에서)을 컴파일하십시오. 여기서 amqspnt 는 샘플 프로그램입니다:

```
cob32 -xvP amqspnt.cbl -L${MQ_INSTALLATION_PATH}/lib -lmqmcb Server for COBOL  
cob32 -xvP amqspnt.cbl -L${MQ_INSTALLATION_PATH}/lib -lmqicb Client for COBOL  
cob32 -xtvP amqspnt.cbl -L${MQ_INSTALLATION_PATH}/lib -lmqmcb_r Threaded Server for COBOL  
cob32 -xtvP amqspnt.cbl -L${MQ_INSTALLATION_PATH}/lib -lmqicb_r Threaded Client for COBOL
```

Micro Focus COBOL 을 사용하여 64비트 COBOL 프로그램을 컴파일하십시오. 여기서 amqsput 는 샘플 프로그램입니다:

```
cob64 -xvP amqsput.cbl -LMQ_INSTALLATION_PATH/lib64 -lmqmc Server for COBOL
cob64 -xvP amqsput.cbl -LMQ_INSTALLATION_PATH/lib64 -lmqic Client for COBOL
cob64 -xtvP amqsput.cbl -LMQ_INSTALLATION_PATH/lib64 -lmqmc_r Threaded Server for COBOL
cob64 -xtvP amqsput.cbl -LMQ_INSTALLATION_PATH/lib64 -lmqic_r Threaded Client for COBOL
```

Windows에서 절차적 애플리케이션 빌드

Windows 시스템 서적은 사용자가 작성하는 프로그램에서 실행 가능한 애플리케이션을 빌드하는 방법을 설명합니다.

이 주제에서는 Windows 시스템에서 실행할 IBM MQ for Windows 애플리케이션을 빌드할 때 수행해야 하는 추가 태스크 및 표준 태스크에 대한 변경사항을 설명합니다. C, C++, COBOL, Visual Basic 프로그래밍 언어가 지원됩니다. C++ 프로그램 준비에 대한 정보는 [C++ 사용](#)의 내용을 참조하십시오.

IBM MQ for Windows를 사용하여 실행 가능한 애플리케이션을 작성하기 위해 수행해야 할 태스크는 소스 코드가 기록된 프로그래밍 언어에 따라 다릅니다. 소스 코드로 MQI 호출을 코딩하는 것 외에도 사용 중인 언어에 대해 IBM MQ for Windows 포함 파일을 포함하도록 적절한 언어문을 추가해야 합니다. 이러한 파일의 콘텐츠를 숙지하십시오. 전체 설명은 677 페이지의 『IBM MQ 데이터 정의 파일』의 내용을 참조하십시오.

Windows의 64비트 애플리케이션 빌드

32비트와 64비트 애플리케이션 모두 IBM MQ for Windows에서 지원됩니다. IBM MQ 실행 파일 및 라이브러리 파일은 32비트와 64비트 양식 둘 모두로 제공됩니다. 작업 중인 애플리케이션에 따라 적절한 버전을 사용하십시오.

실행 파일 및 라이브러리

IBM MQ 라이브러리의 32비트 및 64비트 버전 모두 다음 위치에서 제공됩니다.

표 155. IBM MQ 라이브러리의 위치	
라이브러리 버전	라이브러리 파일을 포함하는 디렉토리
32비트	MQ_INSTALLATION_PATH\Tools\Lib
64비트	MQ_INSTALLATION_PATH\Tools\Lib64

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

32비트 애플리케이션은 마이그레이션 후 계속해서 정상적으로 작동합니다. 32비트 파일은 이전 버전의 제품과 동일한 디렉토리에 있습니다.

64비트 버전을 작성하려면 환경이 MQ_INSTALLATION_PATH\Tools\Lib64의 라이브러리 파일을 사용하도록 구성되어 있는지 확인해야 합니다. LIB 환경 변수가 32비트 라이브러리를 포함하는 폴더를 검색하도록 설정되어 있지 않은지 확인하십시오.

Windows에서 C 프로그램 준비

일반 Windows 환경에서 작업합니다. 따라서 IBM MQ for Windows의 경우 특별한 요구사항이 없습니다.

64비트 애플리케이션 프로그래밍에 대한 추가 정보는 [64비트 플랫폼의 코딩 표준](#)을 참조하십시오.

- IBM MQ에서 제공하는 적절한 라이브러리와 프로그램을 링크합니다.

라이브러리 파일 프로그램/엑시트 유형

MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib 32비트 C용 서버
H

라이브러리 파일 프로그램/엑시트 유형

MQ_INSTALLATION_PATH 32비트 C용 클라이언트
H
\Tools\Lib\mqic.lib

MQ_INSTALLATION_PATH 트랜잭션 조정을 포함한 32비트 C용 클라이언트
H
\Tools\Lib\mqicxa.lib

MQ_INSTALLATION_PATH 64비트 C용 서버
H
\Tools\Lib64\mqm.lib

MQ_INSTALLATION_PATH 64비트 C용 클라이언트
H
\Tools\Lib64\mqic.lib

MQ_INSTALLATION_PATH 트랜잭션 조정을 포함한 64비트 C용 클라이언트
H
\Tools\Lib64\mqicxa.lib

*MQ_INSTALLATION_PATH*은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

다음 명령은 (Microsoft Visual C++ 컴파일러를 사용하여) 샘플 프로그램 amqsget0을 컴파일하는 예를 제공합니다.

32비트 애플리케이션:

```
cl -MD amqsget0.c -Feamqsget.exe ${MQ_INSTALLATION_PATH}\Tools\Lib\mqm.lib
```

64비트 애플리케이션:

```
cl -MD amqsget0.c -Feamqsget.exe ${MQ_INSTALLATION_PATH}\Tools\Lib64\mqm.lib
```

참고:

- 설치 가능 서비스를 작성할 경우(추가 정보는 [관리 IBM MQ](#) 참조) mqmzf.lib 라이브러리에 링크해야 합니다.
- XA 준수 트랜잭션 관리자(예: IBM TXSeries Encina 또는 BEA Tuxedo)에 의해 외부 조정이 되는 애플리케이션을 생성할 경우 mqmxa.lib 또는 mqmxa.lib 라이브러리에 링크해야 합니다.
- CICS 엑시트를 작성할 경우 mqmcics4.lib 라이브러리에 링크하십시오.
- 다른 제품 라이브러리보다 먼저 IBM MQ 라이브러리를 링크해야 합니다.
- DLL은 지정한 경로(PATH)에 있어야 합니다.
- 가능할 때마다 소문자를 사용하는 경우 IBM MQ for Windows에서 소문자를 반드시 사용해야 하는 IBM MQ for AIX or Linux 시스템으로 이동할 수 있습니다.

CICS 및 Transaction Server 프로그램 준비

CICS IBM MQ 트랜잭션의 샘플 C 소스는 AMQSCIC0.CCS에 의해 제공됩니다. 표준 CICS 기능을 사용하여 이를 빌드합니다. 예를 들어, TXSeries for Windows 2000의 경우 다음과 같습니다.

1. 환경 변수를 설정하십시오. (한 행에 다음 코드를 입력하십시오.)

```
set CICS_IBMC_FLAGS=-I${MQ_INSTALLATION_PATH}\Tools\C\Include;  
%CICS_IBMC_FLAGS%
```

2. USERLIB 환경 변수를 설정하십시오.

```
set USERLIB=MQM.LIB;%USERLIB%
```

3. 샘플 프로그램을 변환, 컴파일 및 링크하십시오.

```
cicstcl -l IBM amqscic0.ccs
```

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

이에 대해서는 *Transaction Server for Windows NT* 애플리케이션 프로그래밍 안내서(CICS) V4에서 설명합니다.

지원되는 CICS 거래에 대한 자세한 내용은 [에서 확인할 수 관리 IBM MQ](#) 있습니다.

Windows Windows에서의 COBOL 프로그램 준비

이 정보를 사용하여 Windows에서의 COBOL 프로그램 준비 방법과 CICS 및 Transaction Server 프로그램 준비 방법을 알아봅니다.

1. 32비트 COBOL 복사본은 다음 디렉터리에 설치됩니다: MQ_INSTALLATION_PATH \Tools\cobol\CopyBook.
2. 64비트 COBOL 복사본은 다음 디렉터리에 설치됩니다: MQ_INSTALLATION_PATH \Tools\cobol\CopyBook64
3. 다음 예제에서 CopyBook은 32비트 애플리케이션의 경우 다음으로 설정하십시오.

```
CopyBook
```

32비트 애플리케이션의 경우, 그리고:

```
CopyBook64
```

64비트 애플리케이션용.

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

Windows 시스템에서 COBOL 프로그램을 준비하려면 IBM MQ가 제공하는 다음 라이브러리 중 하나에 프로그램을 링크하십시오.

라이브러리 파일	프로그램 또는 엑시트 유형
MQ_INSTALLATION_PATH \Tools\Lib\mqmcb	마이크로 포커스(Micro Focus)용 32비트 서버 COBOL
MQ_INSTALLATION_PATH \Tools\Lib\mqiccb	Micro Focus COBOL 용 32비트 클라이언트
MQ_INSTALLATION_PATH \Tools\Lib64\mqmcb	마이크로 포커스(Micro Focus)용 64비트 서버 COBOL
MQ_INSTALLATION_PATH \Tools\Lib64\mqiccb	마이크로 포커스(Micro Focus)용 64비트 클라이언트 COBOL

MQI 클라이언트 환경에서 프로그램을 실행 중일 때, DOSCALLS 라이브러리가 COBOL 또는 IBM MQ 라이브러리 앞에 표시되도록 하십시오.

Micro Focus COBOL을 사용한 COBOL 프로그램 준비

기존의 32비트 IBM MQ Micro Focus COBOL 프로그램을 또는 mqmcb.lib 라이브러리를 사용하여 mqiccb.lib 및 mqicbb mqmcb 라이브러리 대신 다시 링크하십시오.

예를 들어, Micro Focus COBOL을 사용하여 샘플 프로그램 amq0put0을(를) 컴파일하려면 다음을 수행하십시오.

1. IBM MQ COBOL 사본을 가리키도록 COBCPY 환경 변수를 설정하십시오. (한 행에 다음 코드를 입력하십시오.)

```
set COBCPY=${MQ_INSTALLATION_PATH}\
Tools\Cobol\Copybook
```

2. 오브젝트 파일을 제공하는 프로그램을 컴파일하십시오.

```
cobol amq0put0 LITLINK
```

3. 런타임 시스템에 오브젝트 파일을 링크하십시오.

- 컴파일러 COBOL 라이브러리를 가리키도록 LIB 환경 변수를 설정하십시오.
- IBM MQ 서버에서 사용할 오브젝트 파일을 링크하십시오.

```
cbllink amq0put0.obj mqmcb.lib
```

- 또는 IBM MQ 클라이언트에서 사용할 오브젝트 파일을 링크하십시오.

```
cbllink amq0put0.obj mqiccb.lib
```

CICS 및 Transaction Server 프로그램 준비

IBM VisualAge COBOL을 사용하여 TXSeries for Windows NT, V5.1 프로그램을 컴파일하고 링크하려면 다음을 수행하십시오.

1. 환경 변수를 설정하십시오. (한 행에 다음 코드를 입력하십시오.)

```
set CICS_IBMCOB_FLAGS=${MQ_INSTALLATION_PATH}\
Cobol\Copybook\VAcobol;%CICS_IBMCOB_FLAGS%
```

2. USERLIB 환경 변수를 설정하십시오.

```
set USERLIB=MQMCBB.LIB
```

3. 프로그램을 변환, 컴파일 및 링크하십시오.

```
cicstcl -l IBMCOB myprog.ccp
```

이에 대해서는 *Transaction Server for Windows NT, V4* 애플리케이션 프로그래밍 안내서에서 설명합니다.

Micro Focus COBOL을 사용하여 CICS for Windows V5 프로그램을 컴파일하고 링크하려면 다음을 수행하십시오.

- INCLUDE 변수를 설정하십시오.

```
set
INCLUDE=drive:\programname\ibm\websphere\tools\c\include;
drive:\opt\cics\include;%INCLUDE%
```

- COBCPY 환경 변수를 설정하십시오.

```
setCOBCPY=drive:\programname\ibm\websphere\tools\cobol\copybook;
drive:\opt\cics\include
```

- COBOL 옵션을 설정하십시오.

```
- set
- COBOPTS=/LITLINK /NOTRUNC
```

그리고 다음 코드를 실행하십시오.

```
cicstran cicsmq00.ccp
cobol cicsmq00.cbl /LITLINK /NOTRUNC
cbllink -D -Mcicsmq00 -Ocicsmq00.cbmfnt cicsmq00.obj
%CICSLIB%\cicsprCBMFNT.lib user32.lib msvcrt.lib kernel32.lib mqmcb.lib
```

Windows SSPI 보안 종료 Windows

IBM MQ for Windows는 IBM MQ MQI client 및 IBM MQ 서버 둘 모두에 대한 보안 엑시트를 제공합니다. 이는 보안 서비스 프로그래밍 인터페이스(SSPI)를 사용하여 IBM MQ 채널에 대한 인증을 제공하는 채널 엑시트 프로그램입니다. SSPI는 Windows 시스템의 통합 보안 기능을 제공합니다.

보안 패키지는 security.dll 또는 secur32.dll에서 로드됩니다. 이러한 DLL은 운영 체제와 함께 제공됩니다.

단방향 인증은 NTLM 인증 서비스를 사용하여 제공됩니다. 양방향 인증은 Kerberos 인증 서비스를 사용하여 제공됩니다.

보안 엑시트 프로그램은 소스 및 오브젝트 형식으로 제공됩니다. 오브젝트 코드를 그대로 사용하거나, 소스 코드를 고유 사용자 엑시트 프로그램 작성의 시작 지점으로 사용할 수 있습니다.

1041 페이지의 『Windows에서 SSPI 보안 엑시트 사용』도 참조하십시오.

보안 엑시트에 대한 소개

보안 엑시트는 두 개의 보안 엑시트 프로그램(송신용 MCA(Message Channel Agent) 프로그램과 수신용 MCA 프로그램) 사이에서 보안 연결을 생성합니다.

보안 연결을 시작하는 프로그램 즉, MCA 세션이 설정된 후 제어를 가져올 첫 번째 프로그램은 컨텍스트 시작기라고 합니다. 파트너 프로그램은 컨텍스트 억셉터라고 합니다.

다음 표는 일부 채널 유형 즉, 컨텍스트 시작기 및 연관된 컨텍스트 억셉터를 표시합니다.

표 156. 컨텍스트 시작기 및 연관된 컨텍스트 억셉터	
컨텍스트 시작기	컨텍스트 억셉터
MQCHT_CLNTCONN	MQCHT_SVRCONN
MQCHT_RECEIVER	MQCHT_SENDER
MQCHT_CLUSRCVR	MQCHT_CLUSSDR

보안 엑시트 프로그램은 두 개의 시작점을 가지고 있습니다.

• SCY_NTLM

이는 NTLM 인증 서비스를 사용하며, 단방향 인증을 제공합니다. NTLM은 서버가 해당 클라이언트의 ID를 확인하는 것을 허용합니다. 클라이언트가 서버의 ID를 확인하거나 한 서버가 다른 서버의 ID를 확인하는 것은 허용하지 않습니다. NTLM 인증은 서버가 진짜라고 가정된 네트워크 환경에서 사용하도록 설계되었습니다.

• SCY_KERBEROS

이는 Kerberos 상호 인증 서비스를 사용합니다. Kerberos 프로토콜은 네트워크 환경에 있는 서버가 진짜라고 가정하지 않습니다. 네트워크 연결의 양 끝에 있는 당사자는 상대방의 ID를 확인할 수 있습니다. 즉, 서버는 클라이언트와 다른 서버의 ID를 확인할 수 있고 클라이언트는 서버의 ID를 확인할 수 있습니다.

보안 엑시트가 수행하는 작업

이 주제에서는 SSPI 채널 엑시트 프로그램이 수행하는 작업에 대해 설명합니다.

제공된 채널 엑시트 프로그램은 세션 설정 중에 파트너 시스템의 단방향 또는 양방향(상호) 인증을 제공합니다. 특정 채널의 경우, 각 엑시트 프로그램에는 연관된 프린시펄(사용자 ID와 유사함, 957 페이지의 『IBM MQ 액세스 제어 및 Windows 프린시펄』 참조)이 있습니다. 두 엑시트 프로그램 사이의 연결은 두 프린시펄 사이의 연결입니다.

기본 세션이 설정된 후, 두 개의 보안 엑시트 프로그램(송신용 MCA 및 수신용 MCA) 사이에 보안 연결이 설정됩니다. 작업 순서는 다음과 같습니다.

1. 각 프로그램은 예를 들면, 명확한 로그인 조작의 결과로 특정 프린시펄과 연관됩니다.

2. 컨텍스트 시작기는 보안 패키지의 파트너(Kerberos의 경우, 이름 지정된 파트너)와의 보안 연결을 요청하고 토큰(token1이라 함)을 수신합니다. 토큰은 이미 설정된 기본 세션을 사용하여 파트너 프로그램으로 송신됩니다.
3. 파트너 프로그램(컨텍스트 억셉터)은 token1을 보안 패키지로 전달하여 컨텍스트 시작기가 정확한지 확인합니다. NTLM의 경우, 연결이 즉시 설정됩니다.
4. Kerberos 제공 보안 엑시트의 경우(즉, 상호 인증의 경우), 보안 패키지가 두 번째 토큰(token2라고 함)도 생성합니다. 이 경우 컨텍스트 억셉터는 기본 세션을 사용하여 컨텍스트 시작기로 돌아갑니다.
5. 컨텍스트 시작기는 token2를 사용하여 컨텍스트 억셉터가 정확한지 확인합니다.
6. 이 스테이지에서 두 애플리케이션이 모두 파트너 토큰의 인증에 만족하는 경우, 보안(인증된) 연결이 설정됩니다.

IBM MQ 액세스 제어 및 Windows 프린시פל

IBM MQ가 제공하는 액세스 제어는 사용자와 그룹을 기반으로 합니다. Windows가 제공하는 인증은 사용자 및 servicePrincipalName(SPN)과 같은 프린시פל을 기반으로 합니다. servicePrincipalName의 경우, 이러한 프린시פל 중 다수가 단일 사용자와 연관되어 있습니다.

SSPI 보안 엑시트는 인증에 관련된 Windows 프린시פל을 사용합니다. Windows 인증이 성공적이면, 엑시트는 액세스 제어를 위해 Windows 프린시פל과 연관된 사용자 ID를 IBM MQ에 전달합니다.

인증과 관련이 있는 Windows 프린시פל은 사용된 인증 유형에 따라 달라집니다.

- NTLM 인증의 경우, 컨텍스트 시작기의 Windows 프린시פל은 실행 중인 프로세스와 연관된 사용자 ID입니다. 이 인증은 단방향이기 때문에 컨텍스트 억셉터와 연관된 프린시פל은 상관없습니다.
- Kerberos 인증의 경우, CLNTCONN 채널의 Windows 프린시פל은 실행 중인 프로세스와 연관된 사용자 ID입니다. 그렇지 않으면, Windows 프린시פל은 다음 접두부를 QueueManagerName에 추가하여 형성되는 servicePrincipalName입니다.

```
ibmqSeries/
```

z/OS z/OS에서 절차적 애플리케이션 빌드

CICS, IMS 및 z/OS 서적은 이러한 환경에서 실행되는 애플리케이션을 빌드하는 방법을 설명합니다.

이 토픽 컬렉션은 이러한 환경의 IBM MQ for z/OS 애플리케이션을 빌드할 때 수행해야 할 표준 태스크의 변경사항 및 추가 태스크에 대해 설명합니다. COBOL, C, C++, 어셈블러 및 PL/I 프로그래밍 언어가 지원됩니다. (C++ 애플리케이션 빌드에 대한 정보는 [C++ 사용](#)의 내용을 참조하십시오.)

실행 가능한 IBM MQ for z/OS 애플리케이션을 작성하기 위해 수행해야 할 태스크는 프로그램이 기록된 프로그래밍 언어와 애플리케이션이 실행될 환경 두 가지 모두에 따라 다릅니다.

프로그램에서 MQI 호출을 코딩하는 것 외에도 사용 중인 언어에 대해 IBM MQ for z/OS 데이터 정의 파일을 포함하도록 적절한 언어문을 추가하십시오. 이러한 파일의 콘텐츠를 숙지하십시오. 전체 설명은 [677 페이지의 『IBM MQ 데이터 정의 파일』](#)의 내용을 참조하십시오.

참고

이름 **thlqual**은 z/OS에 있는 설치 라이브러리의 상위 레벨 규정자입니다.

z/OS 프로그램을 실행할 준비 z/OS

IBM MQ 애플리케이션에서 실행 가능한 애플리케이션을 작성할 수 있도록 프로그램을 작성한 경우, 해당 프로그램을 컴파일하거나 어셈블한 후 지원되는 각 환경에 대해 IBM MQ for z/OS가 제공하는 스텝 프로그램으로 결과 오브젝트 코드를 링크 편집해야 합니다.

프로그램을 준비하는 방법은 애플리케이션이 실행되는 환경(배치, CICS, IMS(BMP 또는 MPP), Linux 또는 z/OS UNIX System Services) 및 z/OS 설치의 데이터 세트 구조 둘 모두에 따라 달라집니다.

963 페이지의 『동적으로 IBM MQ 스텝 호출』에서는 IBM MQ 스텝을 링크 편집할 필요가 없도록 프로그램에서 MQI 호출을 수행하는 대체 방법에 대해 설명합니다. 이 방법이 모든 언어 및 환경에 대해 사용 가능한 것은 아닙니다.

프로그램이 실행되고 있는 IBM MQ for z/OS의 버전 레벨보다 상위 레벨의 스텝 프로그램은 링크 편집하지 마십시오. 예를 들어, V5.2 에서 MQSeries 실행되는 프로그램은 IBM MQ for z/OSV70S/390®와 함께 제공되는 스텝 프로그램과 링크 편집되어서는 안 됩니다.

▶ **z/OS** 64비트 C 애플리케이션 빌드 z/OS

z/OS에서 64비트 C 애플리케이션은 LP64 컴파일러 및 바인더 옵션을 사용하여 빌드됩니다. IBM MQ for z/OS *cmqc.h* 헤더 파일은 이 옵션이 컴파일러에 제공되는 시점을 인식하고 64비트 조작에 적합한 구조 및 IBM MQ 데이터 유형을 생성합니다.

이 옵션으로 빌드된 C 코드는 필요한 조정 시맨틱에 적절한 동적 링크 라이브러리(DLL)를 사용하도록 빌드되어야 합니다. 이 작업을 수행하려면 컴파일된 코드를 다음 테이블에 정의된 적절한 사이드덱과 바인딩합니다.

표 157. 각 조정 시맨틱에 필요한 사이드덱 이름	
조정	사이드덱 이름
단일 단계 커미트 MQI	CSQBMQ2X
RRS 조정의 2단계 커미트(RRS verb 사용)	CSQBRR2X
RRS 조정의 2단계 커미트(MQI verb 사용)	CSQBRI2X

참고: 31비트 C 애플리케이션의 경우 960 페이지의 『31비트 언어 환경 또는 XPLINK를 사용하여 z/OS 배치 애플리케이션 빌드』에 설명된 대로 호출 인터페이스(언어 환경 또는 XPLINK)의 컴파일러 옵션도 설정합니다. 지원되는 유일한 연결은 XPLINK이기 때문에 64비트 C 애플리케이션의 경우 호출 인터페이스를 지정하지 않습니다.

z/OS XL C/C++에서 제공하는 EDCQCB JCL 프로시저를 사용하여 다음과 같이 배치 작업으로서 단일 단계 커미트 IBM MQ 프로그램을 빌드하십시오.

```
//PROCS JCLLIB ORDER=CBC.SCCNPRC
//CLG EXEC EDCQCB,
// INFILE='thlqual.SCSQC37S(CSQ4BCG1)', < MQ SAMPLES
// CPARM='RENT,SSCOM,DLL,LP64,LIST,NOMAR,NOSEQ', < COMPILER OPTIONS
// LIBPRFX='CEE', < PREFIX FOR LIBRARY DSN
// LNGPRFX='CBC', < PREFIX FOR LANGUAGE DSN
// BPARM='MAP,XREF,RENT,DYNAM=DLL', < LINK EDIT OPTIONS
// OUTFILE='userid.LOAD(CSQ4BCG1),DISP=SHR'
//COMPILE.SYSLIB DD
// DD
// DD DISP=SHR,DSN=thlqual.SCSQC370
//BIND.SCSQDEFS DD DISP=SHR,DSN=thlqual.SCSQDEFS
//BIND.SYSIN DD *
INCLUDE SCSQDEFS(CSQBMQ2X)
NAME CSQ4BCG1
```

z/OS UNIX System Services에서 RRS 조정 프로그램을 빌드하려면 다음과 같이 컴파일하고 링크하십시오.

```
cc -o mqsamp -W c,LP64,DLL -W l,DYNAM=DLL,LP64 -I"//thlqual.SCSQC370" "//thlqual.SCSQDEFS(CSQBRR2X)" mqsamp.c
```

▶ **z/OS** 배치 애플리케이션 z/OS 구축 z/OS

z/OS 배치 애플리케이션을 빌드하는 방법과 해당 작업을 수행할 때 고려할 단계에 대해 알아봅니다.

배치 z/OS 환경에서 IBM MQ for z/OS 실행되는 애플리케이션을 구축하려면 다음 작업을 수행하는 작업 제어 언어(JCL)를 생성하십시오:

1. 프로그램을 컴파일(또는 어셈블)하여 오브젝트 코드를 생성합니다. 컴파일용 JCL은 컴파일러에서 제품 데이터 정의 파일을 사용할 수 있도록 하는 SYSLIB문을 포함해야 합니다. 데이터 정의는 다음 IBM MQ for z/OS 라이브러리에서 제공됩니다:
 - COBOL의 경우, **thlqual.SCSQCOBC**
 - 어셈블러 언어의 경우, **thlqual.SCSQMACS**

- C의 경우, **thlqual.SCSQC370**
 - PL/I의 경우, **thlqual.SCSQPLIC**
2. C 애플리케이션의 경우, [958 페이지의 『1』 단계](#)에서 작성된 오브젝트 코드를 사전 링크합니다.
 3. PL/I 애플리케이션의 경우, 컴파일러 옵션 EXTRN(SHORT)을 사용합니다.
 4. [958 페이지의 『1』 단계](#)(또는 C 애플리케이션의 경우 [959 페이지의 『2』 단계](#))에서 작성된 오브젝트 코드를 링크 편집하여 로드 모듈을 생성합니다. 코드를 링크 편집하는 경우 IBM MQ for z/OS 배치 스텝 프로그램 중 하나(CSQBSTUB 또는 RRS 스텝 프로그램 중 하나(CSQBRRSI 또는 CSQBRSTB))를 포함해야 합니다.

CSQBSTUB

IBM MQ for z/OS가 제공하는 1단계 커미트

CSQBRRSI

RRS가 MQI를 사용하여 제공하는 2단계 커미트

CSQBRSTB

RRS가 직접 제공하는 2단계 커미트

참고:

- a. CSQBRSTB를 사용하는 경우 SYS1.CSSLIB의 ATRSCSS로 애플리케이션을 링크 편집해야 합니다. [959 페이지의 그림 129](#) 및 [959 페이지의 그림 130](#)는 이 작업을 수행하는 JCL의 단편을 보여줍니다. 스텝은 언어 독립적이며 라이브러리 **thlqual.SCSQLOAD**에서 제공됩니다.
 - b. 애플리케이션이 언어 환경에서 실행되는 경우 [960 페이지의 『31비트 언어 환경 또는 XPLINK를 사용하여 z/OS 배치 애플리케이션 빌드』](#)에서 설명한 대로, 대신 언어 환경 DLL로 링크-편집을 보장해야 합니다.
5. 애플리케이션 로드 라이브러리에 로드 모듈을 저장합니다.

```

:
//*
//*  WEBSPPHRE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB  DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
:
//SYSIN    DD  *
          INCLUDE CSQSTUB(CSQBSTUB)
:
/*

```

그림 129. 배치 환경에서 1단계 커미트를 사용하여 오브젝트 모듈을 링크 편집하는 JCL의 단편

```

:
//*
//*  WEBSPPHRE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB  DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//CSSLIB   DD  DSN=SYS1.CSSLIB,DISP=SHR
//*
:
//SYSIN    DD  *
          INCLUDE CSQSTUB(CSQBRSTB)
          INCLUDE CSSLIB(ATRSCSS)
:
/*

```

그림 130. 배치 환경에서 2단계 커미트를 사용하여 오브젝트 모듈을 링크 편집하는 JCL의 단편

배치 또는 RRS 프로그램을 실행하려면 STEPLIB 또는 JOBLIB 데이터 세트 병합에 라이브러리 **thlqual.SCSQAUTH** 및 **thlqual.SCSQLOAD**를 포함해야 합니다.

TSO 프로그램을 실행하려면, TSO 세션에서 사용되는 STEPLIB에 라이브러리 **thlqual.SCSQAUTH** 및 **thlqual.SCSQLOAD**를 포함해야 합니다.

z/OS UNIX System Services 셸에서 배치 프로그램을 실행하려면 다음과 같이 \$HOME?.profile의 STEPLIB 스펙에 **thlqual.SCSQAUTH** 및 **thlqual.SCSQLOAD** 라이브러리를 추가하십시오.

```
STEPLIB= thlqual.SCSQAUTH: thlqual.SCSQLOAD
export STEPLIB
```

z/OS 31비트 언어 환경 또는 **XPLINK**를 사용하여 z/OS 배치 애플리케이션 빌드
IBM MQ for z/OS는 애플리케이션을 링크 편집할 때 사용해야 할 동적 링크 라이브러리(DLL) 세트를 제공합니다.

애플리케이션이 다음 호출 인터페이스 중 하나를 사용하는 데 이용할 수 있는 두 개의 라이브러리 변형이 있습니다.

- 31비트 언어 환경 호출 인터페이스.
- 31비트 XPLINK 호출 인터페이스. z/OS XPLINK는 C 애플리케이션에서 사용 가능한 고성능 호출 변환입니다. z/OS 2.2 문서의 [XPLINK](#) | [NOXPLINK](#)를 참조하십시오.

DLL을 사용할 경우, 이전 버전과 함께 제공된 스텝 대신 소위 사이드데크(*sidedeck*)에 대해 애플리케이션이 바인드되거나 링크됩니다. 사이드데크는 (SCSQLOAD 라이브러리 대신) SCSQDEFS 라이브러리에 있습니다.

표 158. 동적 링크 라이브러리의 변형			
커미트	31비트 언어 환경 DLL	31비트 XPLINK DLL	해당 스텝 이름
1단계 커미트 MQI 라이브러리	CSQBMQ1	CSQBMQ1X	CSQBSTUB
RRS 트랜잭션 제어 verb를 사용하여 RRS 조정을 통한 2단계 커미트	CSQBRR1	CSQBRR1X	CSQBRSTB
MQI 트랜잭션 제어 verb를 사용하여 RRS 조정을 통한 2단계 커미트	CSQBRI1	CSQBRI1X	CSQBRRSI

참고: 모든 사이드데크에는 CSQASTUB을 포함하여 이전에 해석된 데이터 변환 시작점 MQXCNCV의 정의가 들어 있습니다.

공통된 문제:

- 애플리케이션이 비동기 메시지 이용(MQCB, MQCTL 또는 MQSUB 호출)을 사용하고 이전 DLL 인터페이스가 사용되지 않으면 다음 메시지가 작업 로그에 표시됩니다.

CSQB001E z/OS 배치 또는 z/OS UNIX System Services에서 실행 중인 언어 환경 프로그램은 IBM MQ에 대한 DLL 인터페이스를 사용해야 합니다.

솔루션: 이전에 설명된 대로 스텝 대신 사이드데크를 사용하여 애플리케이션을 다시 빌드합니다.

- 프로그램 빌드 시간에는 다음 메시지가 표시됩니다.

IEW2469E *your-code* 섹션의 MQAPI-NAME에 대한 참조의 속성이 대상 기호의 속성과 일치하지 않습니다.

이유: 이는 cmqc.h의 V701(또는 이후) 버전으로 XPLINK 프로그램을 컴파일했으나 사이드데크와 바인딩되지 않는다는 의미입니다.

솔루션: SCSQLOAD의 스텝 대신에 SCSQDEFS의 적절한 사이드데크에 대해 바인딩하도록 프로그램의 빌드 파일을 변경하십시오.

다음 샘플 JCL은 31비트 언어 환경 DLL 호출 인터페이스를 사용할 수 있도록 C 프로그램을 컴파일 및 링크 편집하는 방법을 설명합니다.

```
//CLG EXEC EDCCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'          < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,DLL
```

```

SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//
// DD DISP=SHR,DSN=h1q.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=h1q.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEE0BJ
// DD DISP=SHR,DSN=h1q.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=h1q.LOAD(MYPROGAM)
//BIND.SYSIN DD *
ENTRY CEESTART
INCLUDE SYSOBJ(CSQBMQ1)
NAME MYPROGAM(R)
//

```

참고: 컴파일은 **DLL** 옵션을 사용합니다. 링크 편집은 **DYNAM=DLL** 옵션을 사용하고 **CSQBMQ1** 라이브러리를 참조합니다.

다음 샘플 JCL은 31비트 XPLINK DLL 호출 인터페이스를 사용할 수 있도록 C 프로그램을 컴파일 및 링크 편집하는 방법을 설명합니다.

```

//CLG EXEC EDCXCB,
// INFILE=MYPROGS.CPROGS(MYPROGRAM),
// CPARM='OPTF(DD:OPTF)',
// BPARM='XREF,MAP,DYNAM=DLL' < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,XPLINK,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//
// DD DISP=SHR,DSN=h1q.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=h1q.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEE0BJ
// DD DISP=SHR,DSN=h1q.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=h1q.LOAD(MYPROGAM)
//BIND.SYSIN DD *
ENTRY CEESTART
INCLUDE SYSOBJ(CSQBMQ1X)
NAME MYPROGAM(R)
//

```

참고: 컴파일은 **XPLINK** 및 **DLL** 옵션을 사용합니다. 링크 편집은 **DYNAM=DLL** 옵션을 사용하고 **CSQBMQ1X** 라이브러리를 참조합니다.

모듈의 각 프로그램에 컴파일 옵션 DLL을 추가하는지 확인하십시오. IEW2456E 9207 SYMBOL CSQ1BAK UNRESOLVED와 같은 메시지는 모든 프로그램이 DLL 옵션으로 컴파일되었는지 확인해야 한다는 표시입니다.

z/OS에서 CICS 애플리케이션 빌드

'z/OS'에서 'CICS 애플리케이션'을 빌드할 때 이 정보를 사용하십시오.

'CICS'에서 실행되는 'IBM MQ for z/OS'용 애플리케이션을 빌드하려면 다음을 수행해야 합니다:

- 프로그램의 CICS 명령을 나머지 프로그램이 기록된 언어로 변환하십시오.
- 변환기의 출력을 컴파일 또는 어셈블하여 오브젝트 코드를 생성하십시오.
 - PL/I 프로그램의 경우, 컴파일러 옵션 EXTRN(SHORT)을 사용합니다.
 - C 애플리케이션의 경우, 애플리케이션이 [XPLINK](#)를 사용하지 않는 경우 컴파일러 옵션 DEFINE(MQ_OS_LINKAGE=1)을 사용하십시오.
- 오브젝트 코드를 링크 편집하여 로드 모듈을 작성하십시오.

CICS는 지원하는 각각의 프로그래밍 언어에 대해 이러한 단계를 순서대로 실행하는 프로시저를 제공합니다.

- CICS Transaction Server for z/OS의 경우, *CICS Transaction Server for z/OS System Definition Guide*에 이러한 프로시저의 사용 방법이 설명되어 있고 *CICS/ESA Application Programming Guide*에는 번역 프로세스에 대한 자세한 정보가 제공됩니다.

포함해야 할 항목은 다음과 같습니다.

- 컴파일(또는 어셈블리) 스테이지의 SYSLIB문에는 컴파일러에서 제품 데이터 정의 파일을 사용 가능하게 하는 명령문을 포함해야 합니다. 데이터 정의는 다음 IBM MQ for z/OS 라이브러리에서 제공됩니다:

- COBOL의 경우, **thlqual.SCSQCOBC**
- 어셈블러 언어의 경우, **thlqual.SCSQMACS**
- C의 경우, **thlqual.SCSQC370**
- PL/I의 경우, **thlqual.SCSQPLIC**
- 링크 편집 JCL에는 IBM MQ for z/OS CICS 스텝 프로그램(CSQCSTUB)을 포함해야 합니다. [962 페이지의 그림 131](#)에서는 이 작업을 수행할 JCL 코드의 단편을 보여줍니다. 스텝은 언어 독립적이며 라이브러리 **thlqual.SCSQLOAD**에서 제공됩니다.

```

:
/*
/* * WEBSphere MQ FOR Z/OS LIBRARY CONTAINING CICS STUB
/* *
/* * CSQSTUB DD DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
/* *
:
/*LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQCSTUB)
:
/*

```

그림 131. CICS 환경에서 오브젝트 모듈을 링크 편집하는 JCL의 단편

- 'CICS' 버전이 'CICS TS 3.2' 낮은 경우 또는 'IBM MQ' 메시지 속성 API 또는 'IBM MQ' API인 MQCB, MQCTL, MQSTAT, MQSUB 또는 MQSUBR을 사용하려는 경우, 'CICS' 제공 스텝인 DFHMQSTB가 아닌 'IBM MQ' 제공 CSQCSTUB에 개체 코드를 연동해야 합니다. 예 CICS에 대한 프로그램 구축에 IBM MQ에 대한 자세한 내용은 제품 CICS 설명서의 “[MQI 호출에 액세스하기 IBM MQ 위한 API 스텝 프로그램](#)”을 참조하십시오.

이러한 단계를 완료했다면 로드 모듈을 애플리케이션 로드 라이브러리에 저장하고 일반적인 방법으로 CICS에 프로그램을 정의하십시오.

CICS 프로그램을 실행하기 전에, 시스템 관리자는 이를 CICS에 IBM MQ 프로그램 및 트랜잭션으로 정의해야 합니다. 그런 다음 일반적인 방법으로 해당 프로그램을 실행할 수 있습니다.

▶ **z/OS** IMS(BMP 또는 MPP) 애플리케이션 빌드

IMS(BMP 또는 MPP) 애플리케이션을 빌드할 때 이 정보를 사용합니다.

배치 DL/I 프로그램을 빌드할 경우 [958 페이지의 『배치 애플리케이션 z/OS 구축 z/OS』](#)의 내용을 참조하십시오. IMS에서 실행되는 다른 애플리케이션을 (BMP 또는 MPP로) 빌드하려면 이러한 작업을 수행하는 JCL을 작성하십시오.

1. 프로그램을 컴파일(또는 어셈블)하여 오브젝트 코드를 생성합니다. 컴파일용 JCL은 컴파일러에서 제품 데이터 정의 파일을 사용할 수 있도록 하는 SYSLIB문을 포함해야 합니다. 데이터 정의는 다음 IBM MQ for z/OS 라이브러리에서 제공됩니다:
 - COBOL의 경우, **thlqual.SCSQCOBC**
 - 어셈블러 언어의 경우, **thlqual.SCSQMACS**
 - C의 경우, **thlqual.SCSQC370**
 - PL/I의 경우, **thlqual.SCSQPLIC**
2. C 애플리케이션의 경우, [962 페이지의 『1』](#) 단계에서 작성된 오브젝트 모듈을 사전 링크합니다.
3. PL/I 프로그램의 경우, 컴파일러 옵션 EXTRN(SHORT)을 사용합니다.
4. C 애플리케이션의 경우, 애플리케이션이 XPLINK를 사용하지 않으면 컴파일러 옵션 DEFINE(MQ_OS_LINKAGE=1)을 사용하십시오.
5. '962 페이지의 『1』' 단계(또는 'C/370 애플리케이션의 경우' [962 페이지의 『2』](#) '단계)에서 생성한 객체 코드를 링크 편집하여 로드 모듈을 생성합니다:
 - a. IMS 언어 인터페이스 모듈(DFSLI000)을 포함합니다.
 - b. IBM MQ for z/OS IMS 스텝 프로그램(CSQSTUB)을 포함합니다. [963 페이지의 그림 132](#)은 이 작업을 수행할 JCL 코드의 단편을 보여줍니다. 스텝은 언어 독립적이며 라이브러리 **thlqual.SCSQLOAD**에서 제공됩니다.

참고: COBOL을 사용 중인 경우, 963 페이지의 『동적으로 IBM MQ 스텝 호출』에 설명된 대로 동적 링크를 사용하지 않으려면 연계 편집기에서 CSQSTUB에 대한 참조를 해석할 수 있도록 NODYNAM 컴파일러 옵션을 선택하십시오.

6. 애플리케이션 로드 라이브러리에 로드 모듈을 저장합니다.

```
...
//*
//* WEBSphere MQ FOR z/OS LIBRARY CONTAINING IMS STUB
//*
//CSQSTUB DD DSN=thlqual.SCSQLOAD,DISP=SHR
//*
...
//LKED.SYSIN DD *
INCLUDE CSQSTUB(CSQSTUB)
...
/*
```

그림 132. IMS 환경에서 오브젝트 모듈을 링크 편집하는 JCL의 단편

IMS 프로그램을 실행하기 전에, 시스템 관리자는 이를 IMS에 IBM MQ 프로그램 및 트랜잭션으로 정의해야 합니다. 그런 다음 일반적인 방법으로 해당 프로그램을 실행할 수 있습니다.

z/OS z/OS UNIX System Services 애플리케이션 빌드
z/OS UNIX System Services 애플리케이션을 빌드할 때 이 정보를 사용합니다.

'z/OS UNIX System Services'에서 실행되는 'IBM MQ for z/OS'용 C 애플리케이션을 빌드하려면 다음과 같이 애플리케이션을 컴파일하고 링크하세요:

```
cc -o mqsamp -W c,DLL -I "///' thlqual.SCSQC370'" mqsamp.c "///' thlqual.SCSQDEFS(CSQBMQ1)'"
```

여기서 **thlqual**은 설치 시 사용되는 상위 레벨 규정자입니다.

C 프로그램을 실행하려면 .profile 파일에 다음을 추가해야 합니다. 이는 루트 디렉토리에 있어야 합니다.

```
STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

변경사항을 인식하려면 z/OS UNIX System Services를 종료하고 z/OS UNIX System Services를 다시 시작해야 합니다.

다중 셸을 실행하려면 다음과 같이 행의 시작 부분에 단어 내보내기를 추가하십시오.

```
export STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

이 작업이 완료되면, CSQBSTUB을 링크하고 IBM MQ 호출을 발행할 수 있습니다.

963 페이지의 『동적으로 IBM MQ 스텝 호출』에서는 IBM MQ 스텝을 링크 편집할 필요가 없도록 프로그램에서 MQI 호출을 수행하는 대체 방법에 대해 설명합니다. 이 방법이 모든 언어 및 환경에 대해 사용 가능한 것은 아닙니다.

프로그램이 실행되고 있는 IBM MQ for z/OS의 버전 레벨보다 상위 레벨의 스텝 프로그램은 링크 편집하지 마십시오. 예를 들어, IBM WebSphere MQ for z/OS 7.1에서 실행 중인 프로그램은 IBM MQ for z/OS 8.0과 함께 제공된 스텝 프로그램으로 링크 편집해서는 안 됩니다.

z/OS 동적으로 IBM MQ 스텝 호출

IBM MQ 스텝 프로그램을 오브젝트 코드로 링크 편집하는 대신 프로그램 내에서 스텝을 동적으로 호출할 수 있습니다.

이 작업은 배치, IMS 및 CICS 환경에서 수행할 수 있습니다. 이 기능은 RRS 환경에서 지원되지 않습니다. 애플리케이션 프로그램이 RRS을 사용하여 업데이트를 통합하는 경우 967 페이지의 『RRS 고려사항』의 내용을 참조하십시오.

하지만 이 방법은

- 프로그램의 복잡도를 증가시킵니다.
- 실행 시간에 프로그램에 필요한 스토리지를 증가시킵니다.

- 프로그램의 성능을 저하시킵니다.
- 다른 환경에서 같은 프로그램을 사용할 수 없음을 의미합니다.

스텝을 동적으로 호출하는 경우 적절한 스텝 프로그램 및 해당 알리어스가 실행 시간에 사용 가능해야만 합니다. 이를 보장하려면 IBM MQ for z/OS 데이터 세트 SCSQLOAD를 포함하십시오.

- 배치 및 IMS의 경우, JCL의 STEPLIB 병합에 포함하십시오.
- CICS의 경우, CICS DFHRPL 병합에 포함하십시오.

동적 스텝을 포함하는 라이브러리(어댑터 설치 정보의 IMSIMS 어댑터 설정 단계에 설명된 대로 빌드됨)가 IMS 리전 JCL의 STEPLIB 연결에서 데이터 세트 SCSQLOAD보다 앞에 위치하도록 하십시오.

동적으로 스텝을 호출할 때 964 페이지의 표 159에 표시된 이름을 사용하십시오. PL/I에서는 프로그램에서 사용된 호출 이름만을 선언하십시오.

표 159. 동적 링크를 위한 호출 이름			
MQI 호출	배치(비RRS) 동적 호출 이름	CICS 동적 호출 이름	IMS 동적 호출 이름
MQBACK	CSQBBACK	지원되지 않음	지원되지 않음
MQBUFMH	CSQBBFMH	CSQCBFMH ¹	MQBUFMH
MQCB	CSQBCB	CSQCCB ¹	지원되지 않음
MQCLOSE	CSQBCLOS	CSQCCLOS	MQCLOSE
MQCMIT	CSQBCOMM	지원되지 않음	지원되지 않음
MQCONN	CSQBCONN	CSQCCONN	MQCONN
MQCONNX	CSQBCONX	CSQCCONX	MQCONNX
MQCRTMH	CSQBCTMH	CSQCCTMH ¹	MQCRTMH
MQCTL	CSQBCTL	CSQCCTL ¹	지원되지 않음
MQDISC	CSQBDISC	CSQCDISC	MQDISC
MQDLTMH	CSQBDTMH	CSQCDTMH ¹	MQDLTMH
MQDLTMP	CSQBDMTP	CSQCDTMP ¹	MQDLTMP
MQGET	CSQBGET	CSQCGET	MQGET
MQINQ	CSQBINQ	CSQCINQ	MQINQ
MQINQMP	CSQBIMQMP	CSQCIQMP ¹	MQINQMP
MQMHBUF	CSQBMHBF	CSQCMHBF ¹	MQMHBUF
MQOPEN	CSQBOPEN	CSQCOPEN	MQOPEN
MQPUT	CSQBPUT	CSQCPUT	MQPUT
MQPUT1	CSQBPUT1	CSQCPUT1	MQPUT1
MQSET	CSQBSET	CSQCSET	MQSET
MQSETMP	CSQBSTMP	CSQCSTMP ¹	MQSETMP
MQSTAT	CSQBSTAT	CSQCSTAT ¹	MQSTAT
MQSUB	CSQBSUB	CSQCSUB ¹	MQSUB
MQSUBRQ	CSQBSUBR	CSQCSUBR ¹	MQSUBRQ

참고: 1. 이러한 API 호출은 CICS TS 3.2 이상을 사용할 때만 사용 가능하며, CICS와 함께 제공되는 CSQCSTUB을 사용해야 합니다. CICS TS 3.2의 경우, APAR PK66866을 적용해야 합니다. CICS TS 4.1의 경우, APAR PK89844를 적용해야 합니다.

이 기술을 사용하는 방법에 대한 예제는 다음 그림을 참조하십시오.

- 배치 및 COBOL: [965 페이지의 그림 133](#) 참조
- CICS 및 COBOL: [965 페이지의 그림 134](#) 참조
- IMS 및 COBOL: [966 페이지의 그림 135](#) 참조
- 배치 및 어셈블러: [966 페이지의 그림 136](#) 참조
- CICS 및 어셈블러: [966 페이지의 그림 137](#) 참조
- IMS 및 어셈블러: [966 페이지의 그림 138](#) 참조
- 배치 및 C: [966 페이지의 그림 139](#)
- CICS 및 C: [967 페이지의 그림 140](#) 참조
- IMS 및 C: [967 페이지의 그림 141](#) 참조
- 배치 및 PL/I: [967 페이지의 그림 142](#) 참조
- IMS 및 PL/I: [967 페이지의 그림 143](#) 참조

```
...      WORKING-STORAGE SECTION.
...
...      05 WS-MQOPEN                      PIC X(8) VALUE 'CSQBOPEN'.
...
...      PROCEDURE DIVISION.
...
...          CALL WS-MQOPEN WS-HCONN
...                      MQOD
...                      WS-OPTIONS
...                      WS-HOBJ
...                      WS-COMPCODE
...                      WS-REASON.
...
...
```

그림 133. 배치 환경에서 COBOL을 사용한 동적 링크

```
...      WORKING-STORAGE SECTION.
...
...      05 WS-MQOPEN                      PIC X(8) VALUE 'CSQCOPEN'.
...
...      PROCEDURE DIVISION.
...
...          CALL WS-MQOPEN WS-HCONN
...                      MQOD
...                      WS-OPTIONS
...                      WS-HOBJ
...                      WS-COMPCODE
...                      WS-REASON.
...
...
```

그림 134. CICS 환경에서 COBOL을 사용한 동적 링크

```

...      WORKING-STORAGE SECTION.
...
...      05 WS-MQOPEN                      PIC X(8) VALUE 'MQOPEN'.
...
...      PROCEDURE DIVISION.
...
...          CALL WS-MQOPEN WS-HCONN
...                      MQOD
...                      WS-OPTIONS
...                      WS-HOBJ
...                      WS-COMPCODE
...                      WS-REASON.
...
...      * ----- *
...      *
...      *      If the compilation option 'DYNAM' is specified
...      *      then you may code the MQ calls as follows
...      *
...      * ----- *
...
...          CALL 'MQOPEN' WS-HCONN
...                      MQOD
...                      WS-OPTIONS
...                      WS-HOBJ
...                      WS-COMPCODE
...                      WS-REASON.
...

```

그림 135. IMS 환경에서 COBOL을 사용한 동적 링크

```

...      LOAD      EP=CSQBOPEN
...
...      CALL (15), (HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
...      DELETE EP=CSQBOPEN
...

```

그림 136. 배치 환경에서 어셈블리 언어를 사용한 동적 링크

```

...      EXEC CICS LOAD PROGRAM('CSQCOPEN') ENTRY(R15)
...
...      CALL (15), (HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
...      EXEC CICS RELEASE PROGRAM('CSQCOPEN')
...

```

그림 137. CICS 환경에서 어셈블리 언어를 사용한 동적 링크

```

...      LOAD      EP=MQOPEN
...
...      CALL (15), (HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
...      DELETE EP=MQOPEN
...

```

그림 138. IMS 환경에서 어셈블리 언어를 사용한 동적 링크

```

...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqbopen;
...
csqbopen = (CALL_ME *) fetch("CSQBOPEN");
(*csqbopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...

```

그림 139. 배치 환경에서 C 언어를 사용한 동적 링크

```

...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqcopen;
...
EXEC CICS LOAD PROGRAM("CSQCOPEN") ENTRY(csqcopen);
(*csqcopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...

```

그림 140. CICS 환경에서 C 언어를 사용한 동적 링크

```

...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * mqopen;
...
mqopen = (CALL_ME *) fetch("MQOPEN");
(*mqopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...

```

그림 141. IMS 환경에서 C 언어를 사용한 동적 링크

```

...
DCL CSQBOPEN ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
FETCH CSQBOPEN;

CALL CSQBOPEN(HQM,
              MQOD,
              OPTIONS,
              HOBJ,
              COMPCODE,
              REASON);

RELEASE CSQBOPEN;

```

그림 142. 배치 환경에서 PL/I를 사용한 동적 링크

```

...
DCL MQOPEN ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
FETCH MQOPEN;

CALL MQOPEN(HQM,
            MQOD,
            OPTIONS,
            HOBJ,
            COMPCODE,
            REASON);

RELEASE MQOPEN;

```

그림 143. IMS 환경에서 PL/I를 사용한 동적 링크

RRS 고려사항

애플리케이션 프로그램이 RRS를 사용하여 업데이트를 통합하는 경우 이 정보를 사용하는 방안을 고려하십시오.

IBM MQ는 RRS 조정이 필요한 배치 프로그램에 대해 두 가지 다른 스텝을 제공합니다(898 페이지의 『RRS 배치 어댑터』 참조). 차후 API 호출 작동의 차이는 MQCONN 시간에 배치 어댑터가 MQCONN 또는 MQCONN API의 스텝 루틴에 의해 전달된 정보로 판별합니다. 이는 적절한 스텝을 사용하여 IBM MQ에 대한 초기 연결을 완료한 경우 RRS 조정이 필요한 배치 애플리케이션에서 동적 API 호출이 사용 가능하다는 의미입니다. 다음 예제에 이러한 내용이 나와 있습니다.

```

WORKING-STORAGE SECTION.
  05 WS-MQOPEN PIC X(8) VALUE 'MQOPEN' .
.

```

```

.
.
PROCEDURE DIVISION.
.
.
.
*
* Static call to MQCONN must be resolved by linkage edit to
* CSQBRSTB or CSQBRSI for RRS coordination
*
CALL 'MQCONN' USING W00-QMGR
                   W03-HCONN
                   W03-COMPCODE
                   W03-REASON.
.
.
.
*
CALL WS-MQOPEN  WS-HCONN
                MQOD
                WS-OPTIONS
                WS-HOBJ
                WS-COMPCODE
                WS-REASON.

```

프로그램 디버깅 z/OS

이 정보를 사용하여 TSO 및 CICS 프로그램 디버깅에 대해 알아보고 CICS 추적을 이해하십시오.

IBM MQ for z/OS 애플리케이션 프로그램을 디버깅하는 데 도움이 되는 주된 요소는 각 API 호출에서 리턴한 이유 코드입니다. 정정 조치에 관한 아이디어를 비롯하여 이러한 코드 목록은 다음을 참조하십시오.

- [IBM MQ for z/OS 메시지, 완료 상태 및 사유 코드 for IBM MQ for z/OS](#)
- [메시지 및 오류 코드](#) 다른 모든 IBM MQ 플랫폼용

이 주제에서는 특정 환경에서 사용할 다른 디버깅 도구도 제안합니다.

TSO 프로그램 디버깅

다음 대화식 디버깅 도구는 TSO 프로그램에 사용할 수 있습니다.

- TEST 도구
- VS COBOL II 대화식 디버깅 도구
- C 및 PL/I 프로그램의 INSPECT 대화식 디버깅 도구

CICS 프로그램 디버깅

프로그램 또는 프로그램 준비 프로시저를 수정할 필요 없이 CEDF(CICS Execution Diagnostic Facility)를 사용하여 CICS 프로그램을 대화식으로 테스트할 수 있습니다.

EDF에 대한 정보는 *CICS Transaction Server for z/OS CICS 애플리케이션 프로그래밍 안내서*를 참조하십시오.

CICS 추적

CICS 추적 제어 트랜잭션(CETR)을 사용하여 CICS 추적 활동을 제어하는 것이 유용할 수도 있습니다.

CETR에 대한 자세한 정보는 *CICS Transaction Server for z/OS CICS-Supplied Transactions* 매뉴얼을 참조하십시오.

CICS 추적이 활성인지 판별하려면 CKQC 패널을 사용하여 연결 상태를 표시하십시오. 이 패널에는 추적 번호도 표시됩니다.

CICS 추적 항목을 해석하려면 [969 페이지의 표 160](#)의 내용을 참조하십시오.

이러한 값의 CICS 추적 항목은 AP0 xxx입니다(여기서 xxx는 CICS 어댑터를 사용할 때 지정된 추적 번호임). CSQCTEST를 제외한 모든 추적은 CSQCTRUE에서 발행합니다. CSQCTEST는 CSQCRST 및 CSQCDSP에서 발행합니다.

표 160. CICS 어댑터 추적 항목			
이름	설명	추적 시퀀스	추적 데이터
CSQCABNT	비정상 종료	END_THREAD ABNORMAL을 IBM MQ에 발행하기 전에. 그 이유는 애플리케이션에서 태스크의 끝과 암시적 백아웃을 수행할 수 있기 때문입니다. ROLLBACK 요청은 이 경우에 END_THREAD 호출에 포함됩니다.	작업 단위 정보. 작업의 상태에 대해 확인할 때 이 정보를 사용할 수 있습니다. (예를 들어, DISPLAY THREAD 명령 또는 IBM MQ for z/OS 로그 인쇄 유틸리티를 통해 생성된 출력을 확인할 수 있음).
CSQCBACK	동기점 백아웃	IBM MQ for z/OS에 BACKOUT을 발행하기 전에 수행합니다. 이는 애플리케이션으로부터의 명시적 백아웃 요청으로 인해서입니다.	작업 단위 정보.
CSQCCCRC	완료 코드 및 이유 코드	API 호출로부터 성공하지 못한 리턴 후에.	완료 코드 및 이유 코드.
CSQCCOMM	동기점 커밋	COMMIT를 IBM MQ for z/OS에 발행하기 전에. 이는 1단계 커밋 요청 또는 2단계 커밋 요청의 두 번째 단계로 인해서일 수 있습니다. 이 요청은 애플리케이션의 명확한 동기점 요청이 원인입니다.	작업 단위 정보.
CSQCEXER	해결 실행	EXECUTE_RESOLVE를 IBM MQ for z/OS에 발행하기 전에.	EXECUTE_RESOLVE을 발행하는 작업 단위의 작업 단위 정보. 이 단위는 재동기화 프로세스의 마지막 인다우트(in-doubt) 작업 단위입니다.
CSQCGETW	GET 대기	CICS 대기를 발행하기 전에.	대기할 ECB의 주소.
CSQCGMGD	GET 메시지 데이터	MQGET에서 성공적으로 리턴한 후에 수행합니다.	최대 40바이트의 메시지 데이터.
CSQCGMGH	GET 메시지 핸들	IBM MQ for z/OS에 MQGET을 발행하기 전에 수행합니다.	오브젝트 핸들.
CSQCGMGI	Get 메시지 ID	MQGET에서 성공적으로 리턴한 후에 수행합니다.	메시지 ID 및 메시지의 상관 ID.
CSQCINDL	인다우트 목록	두 번째 INQUIRE_INDOUBT에서의 성공적 리턴 후에.	작업 목록의 인다우트 단위.
CSQCINDO	IBM 전용		
CSQCINDS	인다우트 목록 크기	첫 번째 INQUIRE_INDOUBT에서 성공적으로 리턴한 후 인다우트 목록이 비어 있지 않을 때 수행합니다.	파일의 길이. 64로 나누면 작업의 인다우트 단위 수가 제공됩니다.
CSQCINQH	INQ 핸들	IBM MQ for z/OS에 MQINQ를 발행하기 전에 수행합니다.	오브젝트 핸들.
CSQCLOSH	CLOSE 핸들	IBM MQ for z/OS에 MQCLOSE를 발행하기 전에 수행합니다.	오브젝트 핸들.
CSQCLOST	손상된 처리	재동기화 프로세스 중에 CICS가 다시 시작되었으므로 재동기화 중인 작업 단위에 관한 처리 정보를 사용할 수 없음을 어댑터에 알립니다.	재동기화 중인 작업 단위에 대해 CICS에 알려진 작업 단위 ID.

표 160. CICS 어댑터 추적 항목 (계속)			
이름	설명	추적 시퀀스	추적 데이터
CSQCNIND	인다우트되지 않은 처리	재동기화 프로세스 중에 CICS에서 재동기화 중인 작업 단위를 인다우트하지 않아야 함(여전히 실행 중일 수 있음)을 어댑터에 알립니다.	재동기화 중인 작업 단위에 대해 CICS에 알려진 작업 단위 ID.
CSQCNORT	정상 종료	END_THREAD NORMAL을 IBM MQ for z/OS에 발행하기 전에. 그 이유는 태스크의 끝이므로 애플리케이션이 암시적 동기점 커미트를 수행할 수도 있기 때문입니다. COMMIT 요청은 이 경우에 END_THREAD 호출에 포함됩니다.	작업 단위 정보.
CSQCOPNH	OPEN 핸들	MQOPEN에서 성공적으로 리턴한 후에 수행합니다.	오브젝트 핸들.
CSQCOPNO	OPEN 오브젝트	IBM MQ for z/OS에 MQOPEN을 발행하기 전에 수행합니다.	오브젝트 이름.
CSQCPMGD	PUT 메시지 데이터	IBM MQ for z/OS에 MQPUT을 발행하기 전에 수행합니다.	최대 40바이트의 메시지 데이터.
CSQCPMGH	PUT 메시지 핸들	IBM MQ for z/OS에 MQPUT을 발행하기 전에 수행합니다.	오브젝트 핸들.
CSQCPMGI	PUT 메시지 ID	IBM MQ for z/OS에서 MQPUT에 성공한 후 수행합니다.	메시지 ID 및 메시지의 상관 ID.
CSQCPREP	동기점 준비	두 단계 커미트 처리 중 1단계에서 PREPARE를 IBM MQ for z/OS에 발행하기 전에. 이 호출은 또한 분산 큐잉 컴포넌트로부터 API 호출로서 발행될 수도 있습니다.	작업 단위 정보.
CSQCP1MD	PUTONE 메시지 데이터	IBM MQ for z/OS에 MQPUT1을 발행하기 전에 수행합니다.	최대 40바이트의 메시지 데이터.
CSQCP1MI	PUTONE 메시지 ID	MQPUT1에서 성공적으로 리턴한 후에 수행합니다.	메시지 ID 및 메시지의 상관 ID.
CSQCP1ON	PUTONE 오브젝트 이름	IBM MQ for z/OS에 MQPUT1을 발행하기 전에 수행합니다.	오브젝트 이름.
CSQCRBAK	해결된 백아웃	RESOLVE_ROLLBACK을 IBM MQ for z/OS에 발행하기 전에.	작업 단위 정보.
CSQCRCMT	해결된 커미트	RESOLVE_COMMIT를 IBM MQ for z/OS에 발행하기 전에.	작업 단위 정보.
CSQCRMIR	RMI 응답	특정 호출로부터 CICS RMI(자원 관리자 인터페이스)로 리턴하기 전에.	구조화된 RMI 응답 값. 해당 의미는 호출 유형에 따라 다릅니다. 이러한 값은 <i>CICS Transaction Server for z/OS</i> 사용자 정의 안내서에 설명되어 있습니다. 호출의 유형을 판별하려면 CICS RMI 컴포넌트에 의해 생성된 이전 추적 항목을 확인하십시오.
CSQCRSYN	재동기화	태스크에 대한 재동기화 프로세스가 시작되기 전에.	재동기화 중인 작업 단위에 대해 CICS에 알려진 작업 단위 ID.

표 160. CICS 어댑터 추적 항목 (계속)			
이름	설명	추적 시퀀스	추적 데이터
CSQCSETH	SET 핸들	IBM MQ for z/OS에 MQSET을 발 행하기 전에 수행합니다.	오브젝트 핸들.
CSQCTASE	IBM 전용		
CSQCTEST	추적 테스트	사용자에 의해 제공된 추적 번호 또 는 연결의 추적 상태를 확인하기 위 해 EXEC CICS ENTER TRACE 호 출에서 사용됩니다.	데이터 없음.
CSQCDCFF	IBM 전용		

Multi 멀티플랫폼용 샘플 절차적 프로그램 사용

이 샘플 프로시저 프로그램은 제품과 함께 전달됩니다. 샘플은 C 및 COBOL로 작성되며 MQI(Message Queue Interface)의 일반 사용을 보여줍니다.

시작하기 전에

샘플은 일반 프로그래밍 기술을 보여주기 위함이 아니므로 프로덕션 프로그램에 포함시키려고 할 수 있는 일부 오류 검사는 생략됩니다.

IBM i RPG 프로그래밍의 경우 [IBM i 애플리케이션 프로그래밍 참조서\(ILE/RPG\)](#)를 참조하십시오.

이 태스크 정보

모든 샘플에 대한 소스 코드는 제품과 함께 제공됩니다. 이 소스에는 프로그램에서 보여준 메시지 큐잉 기술을 설명하는 주석이 포함되어 있습니다.

샘플의 이름은 접두부 amq로 시작합니다. 네 번째 문자는 프로그래밍 언어 및 필요한 경우 컴파일러를 표시합니다.

- s: C 언어
- 0: IBM 및 Micro Focus 컴파일러 둘 다의 COBOL 언어
- i: IBM 컴파일러 전용의 COBOL 언어
- m: Micro Focus 컴파일러 전용의 COBOL 언어

실행 파일의 8번째 문자는 샘플이 로컬 바인딩 모드 또는 클라이언트 모드에서 실행되는지를 나타냅니다. 8번째 문자가 없는 경우에는 샘플이 로컬 바인딩 모드에서 실행됩니다. 8번째 문자가 'c'인 경우 샘플은 클라이언트 모드에서 실행됩니다.

샘플 애플리케이션을 실행하기 전에 먼저 큐 관리자를 작성하고 구성해야 합니다. 클라이언트 연결을 승인하도록 큐 관리자를 설정하려면 [979 페이지의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』](#)의 내용을 참조하십시오.

프로시저

- 다음 링크를 사용하여 샘플 프로그램에 대해 자세히 알아보십시오.
 - [972 페이지의 『C 및 COBOL MQI 샘플에 표시된 기능 AIX and Linux』](#)
 - [975 페이지의 『C 및 COBOL MQI 샘플에 표시된 기능 Windows』](#)
 - [977 페이지의 『C, COBOL 및 RPG MQI 샘플에 표시된 기능은 다음과 같습니다. IBM i』](#)
 - [979 페이지의 『다양한 플랫폼에서 샘플 프로그램 준비 및 실행』](#)
 - [986 페이지의 『API 엑시트 샘플 프로그램』](#)
 - [987 페이지의 『비동기 이용 샘플 프로그램』](#)

- [988 페이지의 『비동기 Put 샘플 프로그램』](#)
- [989 페이지의 『찾아보기 샘플 프로그램』](#)
- [990 페이지의 『브라우저 샘플 프로그램』](#)
- [991 페이지의 『CICS 트랜잭션 샘플』](#)
- [991 페이지의 『연결 샘플 프로그램』](#)
- [993 페이지의 『Data-Conversion 샘플 프로그램』](#)
- [993 페이지의 『데이터베이스 조정 샘플』](#)
- [999 페이지의 『데드-레터 큐 핸들러 샘플』](#)
- [1000 페이지의 『분배 목록 샘플 프로그램』](#)
- [1000 페이지의 『Echo 샘플 프로그램』](#)
- [1001 페이지의 『Get 샘플 프로그램』](#)
- [1003 페이지의 『고가용성 샘플 프로그램』](#)
- [1007 페이지의 『조회 샘플 프로그램』](#)
- [1008 페이지의 『메시지 핸들 샘플 프로그램의 조회 특성』](#)
- [1008 페이지의 『발행/구독 샘플 프로그램』](#)
- [1012 페이지의 『발행 엑시트 샘플 프로그램』](#)
- [1013 페이지의 『Put 샘플 프로그램』](#)
- [1015 페이지의 『참조 메시지 샘플 프로그램』](#)
- [1021 페이지의 『요청 샘플 프로그램』](#)
- [1027 페이지의 『설정 샘플 프로그램』](#)
- [1028 페이지의 『TLS 샘플 프로그램』](#)
- [1031 페이지의 『트리거링 샘플 프로그램』](#)
- [1033 페이지의 『AIX, Linux, and Windows에서 TUXEDO 샘플 사용』](#)
- [1041 페이지의 『Windows에서 SSPI 보안 엑시트 사용』](#)
- [1042 페이지의 『리모트 큐를 사용하여 샘플 실행』](#)
- [1042 페이지의 『클러스터 큐 모니터링 샘플 프로그램\(AMQSCLM\)』](#)
- [1050 페이지의 『CEPL\(Connection Endpoint Lookup\)을 위한 샘플 프로그램』](#)

관련 개념

[497 페이지의 『C++ 샘플 프로그램』](#)

메시지 가져오기 및 넣기를 시연하기 위해 네 개의 샘플 프로그램이 제공됩니다.

관련 태스크

[1079 페이지의 『다음의 샘플 절차적 프로그램을 사용하여 z/OS』](#)

IBM MQ for z/OS와 함께 전달된 샘플 프로시저 애플리케이션은 MQI(Message Queue Interface)의 일반적인 사용에 대해 설명합니다.

C 및 COBOL MQI 샘플에 표시된 기능 AIX and Linux

샘플 프로그램에서 시연한 기술은,에 IBM MQ 관한 AIX and Linux것입니다.

[981 페이지의 『AIX and Linux에서 샘플 프로그램 준비 및 실행』](#)의 내용을 참조하여 IBM MQ for AIX or Linux의 샘플 프로그램이 저장되는 위치를 알아보십시오.

[973 페이지의 표 161](#)에서는 제공되는 C 및 COBOL 소스 파일과 서버 또는 클라이언트 실행 파일이 포함되어 있는지 여부를 나열합니다.

표 161. AIX and Linux에서 MQI(C 및 COBOL)의 사용을 보여주는 샘플 프로그램

기술	C (소스) (974 페이지 의 『1』)	COBOL (소스) (974 페이지 의 『2』)	서버 (C 실행 가능)	클라이언트 (C 실행 가능)
발행/구독 인터페이스 사용	amqspuba amqssuba amqssbxa	샘플 없음	amqspub amqssub amqssbx	샘플 없음
MQPUT 호출을 사용한 메시지 넣기	amqsput0	amq0put0	amqsput	amqsputc
MQPUT1 호출을 사용한 단일 메시지 넣기	amqsinqa amqsecha	amqminqx amqmechx amqiinqx amqiechx	amqsinq amqsech	amqsechc
분배 목록에 메시지 넣기(974 페이지의 『3』)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
요청 메시지에 응답	amqsinqa	amqminqx amqiinqx	amqsinq	샘플 없음
찾아보기를 사용하여 메시지 가져오기(대기 없음)	amqsgbr0	amq0gbr0	amqsgbr	샘플 없음
메시지 가져오기(시간 제한이 있는 대기)	amqsget0	amq0get0	amqsget	amqsgetc
메시지 가져오기(무제한 대기)	amqstrg0	샘플 없음	amqstrg	amqstrgc
메시지 가져오기(데이터 변환 사용)	amqsecha	샘플 없음	amqsech	샘플 없음
큐에 참조 메시지 넣기(974 페이지의 『3』)	amqsprma	샘플 없음	amqsprm	amqsprmc
큐에서 참조 메시지 가져오기(974 페이지의 『3』)	amqsgrma	샘플 없음	amqsgrm	amqsgrmc
참조 메시지 채널 엑시트(974 페이지의 『3』)	amqsqrma amqsxrma	샘플 없음	amqsxrm	샘플 없음
메시지의 첫 20자 찾아보기	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
전체 메시지 찾아보기	amqsbcbg0	샘플 없음	amqsbcbg	amqsbcbgc
공유 입력 큐 사용	amqsinqa	amqminqx amqiinqx	amqsinq	amqsinqc
독점 입력 큐 사용	amqstrg0	amq0req0	amqstrg	amqstrgc
MQINQ 호출 사용	amqsinqa	amqminqx amqiinqx	amqsinq	샘플 없음
MQSET 호출 사용	amqsseta	amqmsetx amqisetx	amqsset	amqssetc
응답 대상 큐 사용	amqsreq0	amq0req0	amqsreq	amqsreqc
메시지 예외 요청	amqsreq0	amq0req0	amqsreq	샘플 없음
잘린 메시지 허용	amqsgbr0	amq0gbr0	amqsgbr	샘플 없음
해석된 큐 이름 사용	amqsgbr0	amq0gbr0	amqsgbr	샘플 없음
프로세스 트리거	amqstrg0	샘플 없음	amqstrg	amqstrgc
데이터 변환 사용	(975 페이지 의 『4』)	샘플 없음	샘플 없음	샘플 없음

표 161. AIX and Linux에서 MQI(C 및 COBOL)의 사용을 보여주는 샘플 프로그램 (계속)				
기술	C (소스) (974 페이지 의 『1』)	COBOL (소스) (974 페이지 의 『2』)	서버 (C 실행 가능)	클라이언트 (C 실행 가능)
SQL을 사용하여 단일 데이터베이스에 액세스하는 IBM MQ (XA 준수 데이터베이스 관리자 통합)	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	샘플 없음	샘플 없음
SQL을 사용하여 두 개의 데이터베이스에 액세스하는 IBM MQ (XA 준수 데이터베이스 관리자 통합)	amqsxag0.c amqsxab0.sqc amqsxaf0.sqc	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	샘플 없음	샘플 없음
CICS 트랜잭션(975 페이지의 『5』)	amqscic0.ccs	샘플 없음	amqscic0	샘플 없음
Encina 트랜잭션(974 페이지의 『3』)	amqsxae0	샘플 없음	amqsxae0	샘플 없음
메시지를 넣는 TUXEDO 트랜잭션(975 페이지의 『6』)	amqstxpx	샘플 없음	샘플 없음	샘플 없음
메시지를 가져오는 TUXEDO 트랜잭션(975 페이지의 『6』)	amqstxgx	샘플 없음	샘플 없음	샘플 없음
TUXEDO에 대한 서버(975 페이지의 『6』)	amqstxsx	샘플 없음	샘플 없음	샘플 없음
데드-레터 큐 핸들러	디렉토리 ./ tools/c/ Samples/ dlq(975 페이지의 『7』)	샘플 없음	amqsdlq	샘플 없음
MQI 클라이언트에서 메시지 넣기	샘플 없음	샘플 없음	샘플 없음	amqsputc
MQI 클라이언트에서 메시지 가져오기	샘플 없음	샘플 없음	샘플 없음	amqsgetc
MQCONN를 사용하여 큐 관리자에 연결	amqscnxc	샘플 없음	샘플 없음	amqscnxc
API 엑시트 사용	amqsaxe0	샘플 없음	amqsaxe	샘플 없음
클러스터 워크로드 밸런싱 엑시트	amqswlm0	샘플 없음	amqswlm	샘플 없음
메시지를 비동기적으로 넣고 MQSTAT 호출을 사용하여 상태 가져오기	amqsapt0	샘플 없음	amqsapt	amqsaptc
다시 연결 가능한 클라이언트	amqsphac amqsg hac amqsm hac	샘플 없음	적용할 수 없음	amqsphac amqsg hac amqsm hac
메시지 이용자를 사용하여 여러 큐의 메시지를 비동기적으로 이용	amqscbf0	샘플 없음	amqscbf	amqscbfc
MQCONN에서 TLS 연결 정보 지정	amqssslc	샘플 없음	적용할 수 없음	amqssslc

참고:

1. IBM MQ MQI client 샘플의 실행 파일 버전은 서버 환경에서 실행되는 샘플과 동일한 소스를 공유합니다.
2. Micro Focus COBOL 컴파일러가 포함된 'amqm'으로 시작하는 컴파일 프로그램, IBM COBOL 컴파일러가 포함된 'amqi'로 시작하는 프로그램 및 둘 중 하나가 포함된 'amq0'으로 시작하는 프로그램.
3.  지원되는 플랫폼은 IBM MQ for AIX 뿐입니다.

4. **AIX** 이 프로그램은 IBM MQ for AIX 다음과 같이 불립니다 amqsvfc0.c
5. **AIX** CICS 에 의해 지원됩니다 IBM MQ for AIX .
6. **Linux** TUXEDO는 에서 Linux 에 IBM MQ 대해 System p지원되지 않습니다.
7. 데드-레터 큐 핸들러의 소스는 여러 파일로 구성되며 별도의 디렉토리에서 제공됩니다.

AIX and Linux 시스템 지원에 대한 자세한 정보는 [시스템 요구 사항 IBM MQ](#)의 내용을 참조하십시오.

관련 개념

1143 페이지의 『C 언어로 ‘MQI’ 앱 개발하기』

C로 IBM MQ 프로그램을 코딩할 때 다음 섹션의 정보를 참고하십시오.

1162 페이지의 『COBOL 에서 MQI 앱 개발하기』

COBOL로 IBM MQ 프로그램을 코딩하는 경우 다음 섹션의 정보를 참고하십시오.

Windows C 및 COBOL MQI 샘플에 표시된 기능 Windows

샘플 프로그램에서 시연한 기술은.에 IBM MQ 관한 Windows것입니다.

975 페이지의 표 162에서는 제공되는 C 및 COBOL 소스 파일과 서버 또는 클라이언트 실행 파일이 포함되어 있는지 여부를 나열합니다.

표 162. MQI의 사용을 보여주는 IBM MQ for Windows 샘플 프로그램(C and COBOL)				
기술	C(소스)	COBOL(소스)	서버 (C 실행 가능)	클라이언트 (C 실행 가능)
발행/구독 인터페이스 사용	amqspuba amqssuba amqssbxa	샘플 없음	amqspub amqssub amqssbx	샘플 없음
MQPUT 호출을 사용한 메시지 넣기	amqsput0	amq0put0	amqsput	amqsputc
MQPUT1 호출을 사용한 단일 메시지 넣기	amqsinqa amqsecha	amqminq2 amqmech2 amqiinq2 amqiech2	amqsinq amqsech	amqsinqc amqsechc
분배 목록에 메시지 넣기	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
요청 메시지에 응답	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
메시지 가져오기(대기 없음)	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
메시지 가져오기(시간 제한이 있는 대기)	amqsget0	amq0get0	amqsget	amqsgetc
메시지 가져오기(무제한 대기)	amqstrg0	샘플 없음	amqstrg	amqstrgc
메시지 가져오기(데이터 변환 사용)	amqsecha	샘플 없음	amqsech	amqsechc
큐에 참조 메시지 넣기	amqsprma	샘플 없음	amqsprm	amqsprmc
큐에서 참조 메시지 가져오기	amqsgrma	샘플 없음	amqsgrm	amqsgrmc
참조 메시지 채널 엑시트	amqsqrma amqsxrma	샘플 없음	amqsxrm	샘플 없음
메시지의 첫 20자 찾아보기	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
전체 메시지 찾아보기	amqsbcg0	샘플 없음	amqsbcg	amqsbcg
공유 입력 큐 사용	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc

표 162. MQI의 사용을 보여주는 IBM MQ for Windows 샘플 프로그램(C and COBOL) (계속)				
기술	C(소스)	COBOL(소스)	서버 (C 실행 가능)	클라이언트 (C 실행 가능)
독점 입력 큐 사용	amqstrg0	amq0req0	amqstrg	amqstrgc
MQINQ 호출 사용	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
MQSET 호출 사용	amqsseta	amqmset2 amqiset2	amqsset	amqssetc
MQINQMP 호출 사용	amqsiqma	샘플 없음	샘플 없음	샘플 없음
응답 대상 큐 사용	amqsreq0	amq0req0	amqsreq	amqsreqc
메시지 예외 요청	amqsreq0	amq0req0	amqsreq	amqsreqc
잘린 메시지 허용	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
해석된 큐 이름 사용	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
프로세스 트리거	amqstrg0	샘플 없음	amqstrg	amqstrgc
데이터 변환 사용	amqsvfc0	샘플 없음	샘플 없음	샘플 없음
SQL을 사용하여 단일 데이터베이스에 액세스하는 IBM MQ (XA 준수 데이터베이스 관리자 통합)	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sq b	샘플 없음	샘플 없음
SQL을 사용하여 두 개의 데이터베이스에 액세스하는 IBM MQ (XA 준수 데이터베이스 관리자 통합)	amqsxag0.c amqsxab0.sq c Db2 amqsxaf0.sqc Db2	amq0xag0.cbl amq0xab0.sq b amq0xaf0.sqb	샘플 없음	샘플 없음
메시지를 넣는 TUXEDO 트랜잭션	amqstxpx	샘플 없음	샘플 없음	샘플 없음
메시지를 가져오는 TUXEDO 트랜잭션	amqstxgx	샘플 없음	샘플 없음	샘플 없음
TUXEDO를 위한 서버	amqstxsx	샘플 없음	샘플 없음	샘플 없음
데드-레터 큐 핸들러	디렉토리 ./ tools/c/ Samples/ dlq(977 페 이지의 『1』)	샘플 없음	amqsdlq	샘플 없음
IBM MQ MQI client로부터 메시지 넣기	샘플 없음	샘플 없음	샘플 없음	amqsputc
IBM MQ MQI client에서 메시지 가져오기	샘플 없음	샘플 없음	샘플 없음	amqsgetc
MQCONN를 사용하여 큐 관리자에 연결	amqscnxc	샘플 없음	샘플 없음	amqscnxc
API 엑시트 사용	amqsaxe0	샘플 없음	amqsaxe	샘플 없음
클러스터 워크로드 밸런싱	amqswlm0	샘플 없음	amqswlm	샘플 없음
SSPI 보안 루틴	amqsspin	샘플 없음	amqrspin.dll	amqrspin.dll

표 162. MQI의 사용을 보여주는 IBM MQ for Windows 샘플 프로그램(C and COBOL) (계속)				
기술	C(소스)	COBOL(소스)	서버 (C 실행 가능)	클라이언트 (C 실행 가능)
메시지를 비동기적으로 넣고 MQSTAT 호출을 사용하여 상태 가져오기	amqsapt0	샘플 없음	amqsapt	amqsaptc
다시 연결 가능한 클라이언트	amqsphac amqsghac amqsmhac	샘플 없음	적용할 수 없음	amqsphac amqsghac amqsmhac
메시지 이용자를 사용하여 여러 큐의 메시지를 비동기적으로 이용	amqscbf0	샘플 없음	amqscbf	amqscbfc
MQCONN에서 TLS 연결 정보 지정	amqssslc	샘플 없음	적용할 수 없음	amqssslc

참고:

1. 데드-레터 큐 핸들러의 소스는 여러 파일로 구성되며 별도의 디렉토리에서 제공됩니다.

관련 개념

[1143 페이지의 『C 언어로 ‘MQI’ 앱 개발하기』](#)

C로 IBM MQ 프로그램을 코딩할 때 다음 섹션의 정보를 참고하십시오.

[1162 페이지의 『COBOL에서 MQI 앱 개발하기』](#)

COBOL로 IBM MQ 프로그램을 코딩하는 경우 다음 섹션의 정보를 참고하십시오.

IBM i C, COBOL 및 RPG MQI 샘플에 표시된 기능은 다음과 같습니다. IBM i

샘플 프로그램에서 시연한 기법을 IBM MQIBM i 보여줍니다.

일부 기술은 둘 이상의 샘플 프로그램에서 발생하지만 하나의 프로그램만 테이블에 나열됩니다.

표 163. IBM i에서 MQI(C 및 COBOL) 사용을 보여주는 샘플 프로그램				
기술	C (소스) (978 페이지 의 『1』)	COBOL (소스) (979 페이지 의 『2』)	RPG (소스) (979 페이지 의 『3』)	클라이언트 (C 실행 가능)(4)
MQPUT 호출을 사용한 메시지 넣기	AMQSPUT0	AMQ0PUT4	AMQ3PUT4	AMQSPUTC
MQPUT 호출을 사용하여 데이터 파일의 메시지 넣기	AMQSPUT4	샘플 없음	샘플 없음	샘플 없음
MQPUT1 호출을 사용한 단일 메시지 넣기	AMQSINQ4, AMQSECH4	AMQ0INQ4, AMQ0ECH4	AMQ3INQ4, AMQ3ECH4	AMQSINQC, AMQSECHC
분배 목록에 메시지 넣기	AMQSPTL4	샘플 없음	샘플 없음	AMQSPTLC
요청 메시지에 응답	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
메시지 가져오기(대기 없음)	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
메시지 가져오기(시간 제한이 있는 대기)	AMQSGET4	AMQ0GET4	AMQ3GET4	AMQSGETC
메시지 가져오기(무제한 대기)	AMQSTRG4	샘플 없음	AMQ3TRG4	AMQSTRGC
메시지 가져오기(데이터 변환 사용)	AMQSECH4	AMQ0ECH4	AMQ3ECH4	AMQSECHC
큐에 참조 메시지 넣기	AMQSPRM4	샘플 없음	샘플 없음	AMQSPRMC
큐에서 참조 메시지 가져오기	AMQSGRM4	샘플 없음	샘플 없음	AMQSGRMC
참조 메시지 채널 엑시트	AMQSQRM4, AMQSXRM4	샘플 없음	샘플 없음	샘플 없음

표 163. IBM i에서 MQI(C 및 COBOL) 사용을 보여주는 샘플 프로그램 (계속)				
기술	C (소스) (978 페이지 의 『1』)	COBOL (소스) (979 페이지 의 『2』)	RPG (소스) (979 페이지 의 『3』)	클라이언트 (C 실행 가능)(4)
메시지 엑시트	AMQSCMX4	샘플 없음	샘플 없음	샘플 없음
메시지의 처음 49자 찾아보기	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
전체 메시지 찾아보기	AMQSBCG4	샘플 없음	샘플 없음	AMQSBCGC
공유 입력 큐 사용	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
독점 입력 큐 사용	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
MQINQ 호출 사용	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
MQSET 호출 사용	AMQSSET4	AMQ0SET4	AMQ3SET4	AMQSSETC
응답 대상 큐 사용	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
메시지 예외 요청	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
잘린 메시지 허용	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
해석된 큐 이름 사용	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
프로세스 트리거	AMQSTRG4	샘플 없음	AMQ3TRG4	AMQSTRGC
서버 트리거	AMQSERV4	샘플 없음	AMQ3SRV4	샘플 없음
트리거 서버 사용(CICS 트랜잭션 포함)	AMQSERV4	샘플 없음	AMQ3SRV4	샘플 없음
데이터 변환 사용	AMQSVFC4	샘플 없음	샘플 없음	샘플 없음
API 엑시트 사용	AMQSAXE0	샘플 없음	샘플 없음	샘플 없음
클러스터 워크로드 밸런싱	AMQSWLM0	샘플 없음	샘플 없음	샘플 없음
메시지를 비동기적으로 넣고 MQSTAT 호출을 사용하여 상태 가져오기	AMQSAPT0	샘플 없음	샘플 없음	AMQSAPTC
발행/구독 인터페이스 사용	AMQSPUBA, AMQSSUBA, AMQSSBXA	샘플 없음	샘플 없음	AMQSPUBC, AMQSSUBC, AMQSSBXC
다시 연결 가능한 클라이언트(5)	AMQSPHAC, AMQSGHAC, AMQSMHAC	샘플 없음	샘플 없음	샘플 없음
메시지 이용자를 사용하여 여러 큐의 메시지를 비동기적으로 이용 (5)	AMQSCBFO	샘플 없음	샘플 없음	샘플 없음
MQCONN에서 TLS 연결 정보 지정	AMQSSSLC	샘플 없음	샘플 없음	AMQSSSLC
MQCONN를 사용하여 큐 관리자에 연결	AMQSCNXC	샘플 없음	샘플 없음	AMQSCNXC
메시지 큐에서 MQINQM을 사용하여 메시지 핸들의 속성을 조회합니다	AMQISQMA	샘플 없음	샘플 없음	AMQISQMC
MQSETMP를 사용하여 메시지 핸들의 속성을 설정하고 메시지 큐에 넣으십시오	AMQSSQMA	샘플 없음	샘플 없음	AMQSSQMC

참고:

1. C 샘플에 대한 소스는 QMQMSAMP/QCSRC 파일에 있습니다. 포함 파일은 QMQM/H 파일에서 멤버로 존재합니다.

2. COBOL 샘플에 대한 소스는 QMQMSAMP/QCBLLESRC 파일에 있습니다. 멤버는 AMQ0 xxx 4로 이름이 지정되며 여기서 xxx는 샘플 함수를 나타냅니다.
3. RPG 샘플에 대한 소스는 QMQMSAMP/QRPGLESRC에 있습니다. 멤버는 AMQ3 xxx 4로 이름이 지정되며 여기서 xxx는 샘플 함수를 나타냅니다. 복사 멤버는 QMQM/QRPGLESRC에 존재합니다. 각 멤버 이름에는 G가 접두부로 지정되어 있습니다.
4. IBM MQ MQI client 샘플의 실행 파일 버전은 서버 환경에서 실행되는 샘플과 동일한 소스를 공유합니다. 클라이언트 환경에서 샘플에 대한 소스는 서버와 동일합니다. IBM MQ MQI client 샘플은 클라이언트 라이브러리 LIBMQIC와 링크되어 있으며 IBM MQ 서버 샘플은 서버 라이브러리 LIBMQM과 링크되어 있습니다.
5. 다시 연결 가능한 클라이언트의 샘플 애플리케이션 및 비동기적 사용자 애플리케이션에 대한 클라이언트 실행 파일이 실행되어야 하는 경우 컴파일되고 스레드 라이브러리 LIBMQIC_R과 링크되어야 합니다. 이런 이유로 스레드 환경에서 실행되어야 합니다. 환경 변수 QIBM_MULTI_THREADED를 'Y'로 설정하고 qsh에서 애플리케이션을 실행하십시오.

자세한 내용은 [JMS 설정 및 Java IBM MQ 구성](#)을 참조하십시오.

자세한 정보는 [983 페이지의 『C 및 COBOL 샘플 준비 및 실행 IBM i』](#)의 내용을 참조하십시오.

이 뿐만 아니라 IBM MQ for IBM i 샘플 옵션에는 샘플 프로그램 AMQSDATA 및 관리 태스크를 표시하는 샘플 CL 프로그램에 대한 입력으로 사용하는 샘플 데이터 파일이 포함되어 있습니다. CL 샘플은 [IBM i 관리](#)에 설명되어 있습니다. 이 항목에서 설명한 샘플 프로그램과 함께 사용할 큐를 생성하려면 샘플 [AMQSAMP4 CL 프로그램](#)을 사용할 수 있습니다.

관련 개념

[1143 페이지의 『C 언어로 ‘MQI’ 앱 개발하기』](#)

C로 IBM MQ 프로그램을 코딩할 때 다음 섹션의 정보를 참고하십시오.

[1162 페이지의 『COBOL 에서 MQI 앱 개발하기』](#)

COBOL로 IBM MQ 프로그램을 코딩하는 경우 다음 섹션의 정보를 참고하십시오.

[1205 페이지의 『RPG로 ‘MQI’ 앱 개발하기』](#)

IBM MQ 문서에서, 호출의 매개변수, 데이터 유형 이름, 구조의 필드 및 상수 이름은 모두 긴 이름을 사용하여 설명됩니다. RPG에서 이러한 이름은 6개 이하의 대문자로 단축됩니다.

관련 참조

[1068 페이지의 『RPG 샘플 프로그램에 표시된 기능들 IBM i』](#)

IBM MQ for IBM i 샘플 프로그램에 나타나는 기술 설명 표

다양한 플랫폼에서 샘플 프로그램 준비 및 실행

일부 초기 준비를 완료한 후에 샘플 프로그램을 실행할 수 있습니다.

이 태스크 정보

샘플 프로그램을 실행하기 전에 먼저 큐 관리자를 작성하고 필요한 큐도 작성해야 합니다. 예를 들어, COBOL 샘플을 실행하려는 경우 추가 준비가 필요할 수도 있습니다. 필요한 준비를 완료한 후에 샘플 프로그램을 실행할 수 있습니다.

프로시저

샘플 프로그램을 준비하고 실행하는 방법에 대한 정보는 다음 토픽을 참조하십시오.

- [979 페이지의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』](#)
- [983 페이지의 『C 및 COBOL 샘플 준비 및 실행 IBM i』](#)
- [981 페이지의 『AIX and Linux에서 샘플 프로그램 준비 및 실행』](#)
- [984 페이지의 『Windows에서 샘플 프로그램 준비 및 실행』](#)

멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성

샘플 애플리케이션을 실행하기 전에 먼저 큐 관리자를 작성해야 합니다. 그런 다음 클라이언트 모드로 실행 중인 애플리케이션으로부터 수신되는 연결 요청을 안전하게 승인하도록 큐 관리자를 구성할 수 있습니다.

시작하기 전에

큐 관리자가 이미 존재하며 시작되었는지 확인하십시오. MQSC 명령을 발행하여 채널 인증 레코드가 이미 사용으로 설정되어 있는지 판별하십시오.

```
DISPLAY QMGR CHLAUTH
```

중요사항: 이 태스크에서는 채널 인증 레코드를 사용할 수 있는 것으로 예상합니다. 이것이 기타 사용자 및 애플리케이션에서 사용하는 큐 관리자인 경우 이 설정을 변경하면 기타 모든 사용자 및 애플리케이션에 영향을 미칩니다. 큐 관리자가 채널 인증 레코드를 이용하지 않는 경우 MCAUSER를 980 페이지의 『1』 단계에서 획득하는 *non-privileged-user-id*로 설정하는 대체 인증 방법(예: 보안 엑시트)으로 4단계를 대체할 수 있습니다.

애플리케이션이 사용할 것으로 예상되는 채널 이름을 알아야 애플리케이션이 해당 채널을 사용하도록 허용할 수 있습니다. 애플리케이션이 사용할 것으로 예상되는 오브젝트(예: 큐 또는 토픽)를 알아야 애플리케이션이 해당 오브젝트를 사용하도록 허용할 수 있습니다.

이 태스크 정보

이 태스크는 큐 관리자에 연결하는 클라이언트 애플리케이션에 사용될 권한이 없는 사용자 ID를 작성합니다. 이 사용자 ID를 사용하면 클라이언트 애플리케이션이 필요한 큐 및 필요한 채널만을 사용할 수 있도록 액세스가 부여됩니다.

프로시저

1. 큐 관리자가 실행 중인 시스템에서 사용자 ID를 확보하십시오. 이 태스크의 경우 이 사용자 ID는 권한이 있는 관리자가 아니어야 합니다. 이 사용자 ID를 사용하면 큐 관리자에서 클라이언트 연결을 실행할 수 있는 권한이 부여됩니다.
2. 다음 명령으로 리스너 프로그램을 시작하십시오. 여기서,

*qmgr-name*은 큐 관리자의 이름입니다.
*nnnn*은 선택한 포트 번호입니다.

a) **ALW**

AIX, Linux, and Windows 시스템의 경우

```
runmqclsr -t tcp -m qmgr-name -p nnnn
```

b) **IBM i**

IBM i의 경우:

```
STRMQMLSr MQMNAME(qmgr-name) PORT(nnnn)
```

3. 애플리케이션에서 SYSTEM.DEF.SVRCONN을 사용하는 경우 이 채널은 이미 정의되어 있습니다. 애플리케이션이 다른 채널을 사용하는 경우 다음 MQ SC 명령을 실행하여 작성하십시오.

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

여기서 *channel-name*은 채널의 이름입니다.

4. 다음 MQSC 명령을 실행하여 클라이언트 시스템의 IP 주소에서만 채널을 사용하도록 하는 채널 인증 규칙을 작성하십시오.

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

여기서,

*channel-name*은 채널의 이름입니다.

*client-machine-IP-address*는 클라이언트 시스템의 IP 주소입니다. 샘플 클라이언트 애플리케이션이 큐 관리자와 동일한 시스템에서 실행 중인 경우 애플리케이션이 'localhost'를 사용하여 연결할 경우 IP 주소 '127.0.0.1'을 사용하십시오. 서로 다른 여러 클라이언트 시스템에 연결하려는 경우, 단일 IP 주소 대신 패턴 또는 범위를 사용할 수 있습니다. 자세한 내용은 [일반 IP 주소](#)의 내용을 참조하십시오.

*non-privileged-user-id*는 980 페이지의 『1』 단계에서 획득한 사용자 ID입니다.

- 애플리케이션이 SYSTEM.DEFAULT.LOCAL.QUEUE를 사용하는 경우 이 큐는 이미 정의되어 있습니다. 애플리케이션이 다른 큐를 사용하는 경우 다음 MQ SC 명령을 실행하여 작성하십시오.

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

여기서 *queue-name*은 큐의 이름입니다.

- 다음 MQSC 명령을 실행하여 큐 관리자에 대한 연결 및 조회 액세스 권한을 부여하십시오.

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

여기서 *non-privileged-user-id*는 980 페이지의 『1』 단계에서 확보한 사용자 ID입니다.

- 애플리케이션이 지점간 애플리케이션 즉 큐를 사용하는 애플리케이션인 경우, 다음 MQSC 명령을 실행하여 사용할 사용자 ID별로 큐를 사용하는 메시지 조회, 넣기, 가져오기가 허용되는 액세스 권한을 부여하십시오.

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

여기서,

*queue-name*은 큐 이름입니다.

*non-privileged-user-id*는 980 페이지의 『1』 단계에서 획득한 사용자 ID입니다.

- 애플리케이션이 발행/구독 애플리케이션인 경우 즉, 토픽을 사용하는 경우 다음 MQSC 명령을 발행함으로써 사용될 사용자 ID별로 토픽을 사용하여 발행 및 구독할 수 있도록 허용하는 액세스를 부여하십시오.

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

여기서,

*non-privileged-user-id*는 980 페이지의 『1』 단계에서 획득한 사용자 ID입니다.

이 사용자 ID를 사용하면 토픽 트리에 있는 모든 토픽에 *non-privileged-user-id* 액세스가 부여되며, 또는 **DEFINE TOPIC**을 사용하여 토픽 오브젝트를 정의한 후 해당 토픽 오브젝트가 참조하는 토픽 트리의 파티션에만 액세스를 부여할 수 있습니다. 자세한 내용은 [토픽에 대한 사용자 액세스 제어](#)의 내용을 참조하십시오.

다음에 수행할 작업

이제 클라이언트 애플리케이션은 큐 관리자에 연결하고 큐를 사용하여 메시지를 삽입하거나 가져올 수 있습니다.

관련 개념

 객체에 IBM MQ 대한 접근 권한 부여 AIX, Linux, and Windows

관련 참조

[SET CHLAUTH](#)

[DEFINE CHANNEL](#)

[DEFINE QLOCAL](#)

[SET AUTHREC](#)

 IBM MQ 권위자들 IBM i

  **AIX and Linux에서 샘플 프로그램 준비 및 실행**

AIX and Linux에서 샘플 프로그램을 실행하기 전에 먼저 큐 관리자를 작성하고 필요한 큐도 작성해야 합니다. COBOL 샘플을 실행하려는 경우 추가 준비가 필요할 수 있습니다.

이 태스크 정보

설치 시 기본값을 사용한 경우, 시스템 샘플 982 페이지의 표 164 IBM MQAIX and Linux 파일은 에 나열된 디렉터리에 있습니다.

표 164. 어디에서 시스템용 IBM MQAIX and Linux 샘플을 찾을 수 있나요?	
컨텐츠	디렉토리
소스 파일	<code>MQ_INSTALLATION_PATH/samp</code>
데드-레터 큐 핸들러 소스 파일	<code>MQ_INSTALLATION_PATH/samp/dlq</code>
실행 파일	<code>MQ_INSTALLATION_PATH/samp/bin</code>

`MQ_INSTALLATION_PATH`은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

샘플은 작업하기 위한 큐 세트가 필요합니다. 고유 큐를 사용하거나 샘플 MQSC 파일인 `amqscos0.tst`를 실행하여 세트를 작성할 수 있습니다. 샘플을 실행하려면, 제공된 실행 파일 버전을 사용하거나 다른 애플리케이션처럼 ANSI 컴파일러를 사용하여 소스 버전을 컴파일하십시오.

다음 샘플 프로그램들은 인증 기능을 갖추고 있습니다:

- `amqsbcg0.c`
- `amqsfhac.c`
- `amqsget0.c`
- `amqsg hac.c`
- `amqsm hac.c`
- `amqsp hac.c`
- `amqsp uba.c`
- `amqsput0.c`
- `amqssslc.c`
- `amqss uba.c`

이 샘플들의 실행 파일 버전에는 인증 기능이 활성화되어 있습니다. 그러나 인증을 활성화한 상태로 소스 버전을 컴파일하려면 컴파일 플래그가 **`SAMPLE_AUTH_ENABLED`** 정의되어야 하며, 소스 `amqsauth.c` 파일은 원하는 샘플과 함께 컴파일되어야 합니다. 예를 들면, 다음과 같습니다.

- 인증 없이 `amqsput0.c` 컴파일:

```
gcc -m64 -I /opt/mqm/inc -L /opt/mqm/lib64 -lmqic -fsanitize=address -o /bin/amqsput0.c
```

- 인증 기능을 활성화한 상태로 `amqsput0.c` 컴파일:

```
gcc -m64 -I /opt/mqm/inc -L /opt/mqm/lib64 -lmqic -fsanitize=address -D SAMPLE_AUTH_ENABLED -o /bin/amqsputc_auth amqsauth.c amqsput0.c
```

프로시저

1. 큐 관리자를 작성하고 기본 정의를 설정하십시오.

샘플 프로그램을 실행하기 전에 이를 수행해야 합니다. 큐 관리자 작성에 대한 자세한 정보는 [IBM MQ 관리를 참조하십시오](#). 클라이언트 모드로 실행 중인 애플리케이션으로부터 수신되는 연결 요청을 안전하게 승인하도록 큐 관리자를 구성하는 방법에 대한 정보는 [979 페이지의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』](#)의 내용을 참조하십시오.

2. 고유 큐를 사용하지 않는 경우 샘플 MQSC 파일인 `amqscos0.tst`를 실행하여 큐 세트를 작성하십시오. AIX and Linux 시스템에서 이를 수행하려면 다음을 입력하십시오.

```
runmqsc QManagerName <amqscos0.tst > /tmp/sampobj.out
```

`sampobj.out` 파일을 검토하여 오류가 없는지 확인하십시오.

3. Inquire, Set 및 Echo 예의 COBOL 버전을 사용하려는 경우 이러한 샘플을 실행하기 전에 프로세스 정의를 변경하십시오.

Inquire, Set 및 Echo 예의 경우 샘플 정의는 이러한 샘플의 C 버전을 트리거합니다. COBOL 버전을 원하는 경우 프로세스 정의를 변경해야 합니다.

- SYSTEM.SAMPLE.INQPROCESS
- SYSTEM.SAMPLE.SETPROCESS
- SYSTEM.SAMPLE.ECHOPROCESS

해당 날짜에 AIX and Linux, 명령을 **runmqsc** 사용하여 이 샘플들을 실행하기 전에 파일을 **amqscos0.tst** 편집하고 C 실행 파일 이름을 COBOL 실행 파일 이름으로 변경하십시오.

4. 샘플 프로그램을 실행하십시오.

샘플을 실행하려면 이름을 입력하고 뒤에 매개변수를 입력하십시오. 예를 들어, 다음과 같습니다.

```
amqspout myqueue qmanagername
```

여기서 *myqueue*는 메시지가 넣어지는 큐의 이름이며 *qmanagername*은 *myqueue*를 소유하는 큐 관리자입니다.

각 샘플에서 예상하는 매개변수에 대한 자세한 정보는 개별 샘플에 대한 설명을 참조하십시오.

참고: COBOL 샘플 프로그램의 경우, 매개변수로서 큐 이름을 전달하면 필요한 경우 공백 문자로 채워진 48개의 문자를 제공해야 합니다. 48개 외의 문자는 이유 코드 2085와 함께 프로그램이 실패하게 됩니다.

관련 참조

972 페이지의 『C 및 COBOL MQI 샘플에 표시된 기능 AIX and Linux』

샘플 프로그램에서 시연한 기술은,에 IBM MQ 관한 AIX and Linux것입니다.

IBM i C 및 COBOL 샘플 준비 및 실행 IBM i

샘플을 실행하려면 QMQM 라이브러리에 제공된 C 실행 파일 버전을 사용하거나 다른 IBM MQ 애플리케이션과 비슷한 방법으로 컴파일하십시오. COBOL 샘플을 실행하려는 경우 추가 준비가 필요할 수 있습니다.

시작하기 전에

RPG 샘플은 에서 참조하십시오 1069 페이지의 『RPG 샘플 프로그램 준비 및 실행 IBM i』.

C 및 COBOL 샘플을 실행하기 전에, IBM i 큐 관리자(queue manager)와 큐(queues)가 이미 존재해야 합니다.

이 태스크 정보

IBM MQ for IBM i 샘플 프로그램의 소스는 QCSRC, QCLSRC, QCBLLSRC 및 QRPGLSRC 멤버로 QMQMSAMP 라이브러리에 제공됩니다.

샘플을 실행할 때 사용자 정의 큐를 사용할 수 있으며, 샘플 프로그램을 실행하여 AMQSAMP4 샘플 큐를 생성할 수도 있습니다. 이 프로그램의 소스 코드는 라이브러리 QMQMSAMP의 파일 QCLSRC에 포함되어 있습니다. CRTCLPGM 명령을 사용하여 이를 컴파일할 수 있습니다.

다음 샘플 프로그램들은 인증 기능을 갖추고 있습니다:

- amqsbcg0.c
- amqsfhac.c
- amqsget0.c
- amqsgnac.c
- amqsmnac.c
- amqspnac.c
- amqsphac.c
- amqsput0.c

- amqssslc.c
- amqssuba.c

이 샘플들의 실행 파일 버전에는 인증 기능이 활성화되어 있습니다. 그러나 인증을 활성화한 상태로 소스 버전을 컴파일하려면 컴파일 플래그가 **SAMPLE_AUTH_ENABLED** 정의되어야 하며, 소스 amqsauth.c 파일은 원하는 샘플과 함께 컴파일되어야 합니다. 예를 들면, 다음과 같습니다.

- 인증 기능이 활성화되지 않은 상태로 프로그램 amqssslc 생성:

```
CRTCMOD MODULE(MYLIB/AMQSSSLC) SRCFILE(QMQMSAMP/QCSRC)
CRTPGM PGM(MYLIB/AMQSSSLC) MODULE(MYLIB/AMQSSSLC) BNDSRVPGM(QMQM/LIBMQIC)
```

- 인증 기능을 활성화한 상태로 amqssslc 생성:

```
CRTCMOD MODULE(MYLIB/AMQSSSLC) DEFINE('SAMPLE_AUTH_ENABLED') SRCFILE(QMQMSAMP/QCSRC)
CRTCMOD MODULE(MYLIB/AMQSAUTH) SRCFILE(QMQMSAMP/QCSRC)
CRTPGM PGM(MYLIB/AMQSSSLC_AUTH) MODULE(MYLIB/AMQSSSLC MYLIB/AMQSAUTH) BNDSRVPGM(QMQM/LIBMQIC)
```

프로시저

1. 큐 관리자를 작성하고 기본 정의를 설정하십시오.

샘플 프로그램을 실행하기 전에 이를 수행해야 합니다. 큐 관리자 작성에 대한 자세한 정보는 **IBM MQ** 관리를 참조하십시오. 클라이언트 모드로 실행 중인 애플리케이션으로부터 수신되는 연결 요청을 안전하게 승인하도록 큐 관리자를 구성하는 방법에 대한 정보는 [979 페이지의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』](#)의 내용을 참조하십시오.

2. QMQMSAMP 라이브러리의 AMQSDATA 파일에 있는 PUT 멤버의 데이터를 사용하여 샘플 프로그램 중 하나를 호출하려면 다음과 같은 명령을 사용하십시오.

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

참고: IFS 파일 시스템을 사용하는 컴파일된 모듈의 경우 CRTCMOD에 옵션 SYSIFCOPT(*IFSIO)를 지정한 다음, 매개변수로 전달된 파일 이름을 다음 형식으로 지정해야 합니다.

```
home/me/myfile
```

3. Inquire, Set 및 Echo 예의 COBOL 버전을 사용하려는 경우 이러한 샘플을 실행하기 전에 프로세스 정의를 변경하십시오.

Inquire, Set 및 Echo 예의 경우 샘플 정의는 이러한 샘플의 C 버전을 트리거합니다. COBOL 버전을 원하는 경우 프로세스 정의를 변경해야 합니다.

- SYSTEM.SAMPLE.INQPROCESS
- SYSTEM.SAMPLE.SETPROCESS
- SYSTEM.SAMPLE.ECHOPROCESS

에서 명령어(자세한 내용은 [변경 CHGMQMPRCMQ](#) 프로세스(IBM iCHGMQMPRC) 참조)를 사용하거나, 대체 정의로 프로그램을 **AMQSAMP4** 편집 및 실행할 수 있습니다.

4. 샘플 프로그램을 실행하십시오.

각 샘플에서 예상하는 매개변수에 대한 자세한 정보는 개별 샘플에 대한 설명을 참조하십시오.

참고: COBOL 샘플 프로그램의 경우, 매개변수로서 큐 이름을 전달하면 필요한 경우 공백 문자로 채워진 48개의 문자를 제공해야 합니다. 48개 외의 문자는 이유 코드 2085와 함께 프로그램이 실패하게 됩니다.

Windows에서 샘플 프로그램 준비 및 실행

Windows에서 샘플 프로그램을 실행하기 전에 먼저 큐 관리자를 작성하고 필요한 큐도 작성해야 합니다. COBOL 샘플을 실행하려는 경우 추가 준비가 필요할 수 있습니다.

이 태스크 정보

설치 시 기본값이 사용된 경우 IBM MQ for Windows 샘플 파일은 [985 페이지의 표 165](#)에 나열된 디렉토리에 있습니다. 설치 드라이브의 기본값은 <c:>입니다.

표 165. IBM MQ for Windows에 대한 샘플이 있는 위치	
컨텐츠	디렉토리
C 소스 코드	<code>MQ_INSTALLATION_PATH\도구\C\샘플</code>
데드 레터 핸들러 샘플에 대한 소스 코드	<code>MQ_INSTALLATION_PATH\도구\C\샘플\DLQ</code>
COBOL 소스 코드	<code>MQ_INSTALLATION_PATH\도구\코볼\샘플</code>
C 실행 파일 ¹	<code>MQ_INSTALLATION_PATH\Tools\C\Samples\Bin(32비트 버전)</code> <code>MQ_INSTALLATION_PATH\Tools\C\Samples\Bin64(64비트 버전)</code>
샘플 MQSC 파일	<code>MQ_INSTALLATION_PATH\도구\MQSC\샘플</code>
Visual Basic 소스 코드	<code>MQ_INSTALLATION_PATH\Tools\VB\SampVB6</code>
.NET 샘플	<code>MQ_INSTALLATION_PATH\도구\dotnet\샘플</code>

`MQ_INSTALLATION_PATH`은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

참고: 64비트 버전은 일부 C 실행 파일 샘플을 사용할 수 있습니다.

샘플은 작업하기 위한 큐 세트가 필요합니다. 고유 큐를 사용하거나 샘플 MQSC 파일인 `amqscos0.tst`를 실행하여 큐 세트를 작성할 수 있습니다. 샘플을 실행하려면, 제공된 실행 파일을 사용하거나 다른 IBM MQ for Windows 애플리케이션처럼 소스 버전을 컴파일하십시오.

다음 샘플 프로그램들은 인증 기능을 갖추고 있습니다:

- `amqsbcg0.c`
- `amqsfhac.c`
- `amqsget0.c`
- `amqsg hac.c`
- `amqsm hac.c`
- `amqsphac.c`
- `amqspub a.c`
- `amqsput0.c`
- `amqsssl.c`
- `amqssub a.c`

이 샘플들의 실행 파일 버전에는 인증 기능이 활성화되어 있습니다. 그러나 인증을 활성화한 상태로 소스 버전을 컴파일하려면 컴파일 플래그가 **SAMPLE_AUTH_ENABLED** 정의되어야 하며, 소스 `amqsauth.c` 파일은 원하는 샘플과 함께 컴파일되어야 합니다. 예를 들면, 다음과 같습니다.

- 인증 없이 `amqsput0.c` 컴파일:

```
CL amqsput0.c /link mqic.lib /OUT:Bin\amqsputc.exe
```

- 인증 기능을 활성화한 상태로 `amqsput0.c` 컴파일:

```
CL /D SAMPLE_AUTH_ENABLED amqsauth.c amqsput0.c /link mqic.lib /OUT:Bin\amqsputc_auth.exe
```

프로시저

1. 큐 관리자를 작성하고 기본 정의를 설정하십시오.

샘플 프로그램을 실행하기 전에 이를 수행해야 합니다. 큐 관리자 작성에 대한 자세한 정보는 [IBM MQ](#) 관리를 참조하십시오. 클라이언트 모드로 실행 중인 애플리케이션으로부터 수신되는 연결 요청을 안전하게 승인하도록 큐 관리자를 구성하는 방법에 대한 정보는 [979 페이지의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』](#)의 내용을 참조하십시오.

2. 고유 큐를 사용하지 않는 경우 샘플 MQSC 파일인 `amqscos0.tst`를 실행하여 큐 세트를 작성하십시오. Windows 시스템에서 이를 수행하려면 다음을 입력하십시오.

```
runmqsc QManagerName < amqscos0.tst > sampobj.out
```

`sampobj.out` 파일을 검토하여 오류가 없는지 확인하십시오. 이 파일은 현재 디렉토리에 있습니다.

3. Inquire, Set 및 Echo 예의 COBOL 버전을 사용하려는 경우 이러한 샘플을 실행하기 전에 프로세스 정의를 변경하십시오.

Inquire, Set 및 Echo 예의 경우 샘플 정의는 이러한 샘플의 C 버전을 트리거합니다. COBOL 버전을 원하는 경우 프로세스 정의를 변경해야 합니다.

- SYSTEM.SAMPLE.INQPROCESS
- SYSTEM.SAMPLE.SETPROCESS
- SYSTEM.SAMPLE.ECHOPROCESS

해당 날짜에 Windows, 명령을 **runmqsc** 사용하여 이 샘플들을 실행하기 전에 파일을 `amqscos0.tst` 편집하고 C 실행 파일 이름을 COBOL 실행 파일 이름으로 변경하십시오.

4. 샘플 프로그램을 실행하십시오.

샘플을 실행하려면 이름을 입력하고 뒤에 매개변수를 입력하십시오. 예를 들어, 다음과 같습니다.

```
amqsput myqueue qmanagername
```

여기서 `myqueue`는 메시지가 놓여지는 큐의 이름이며 `qmanagername`은 `myqueue`를 소유하는 큐 관리자입니다.

각 샘플에서 예상하는 매개변수에 대한 자세한 정보는 개별 샘플에 대한 설명을 참조하십시오.

참고: COBOL 샘플 프로그램의 경우, 매개변수로서 큐 이름을 전달하면 필요한 경우 공백 문자로 채워진 48개의 문자를 제공해야 합니다. 48개 외의 문자는 이유 코드 2085와 함께 프로그램이 실패하게 됩니다.

관련 참조

[975 페이지의 『C 및 COBOL MQI 샘플에 표시된 기능 Windows』](#)
샘플 프로그램에서 시연한 기술은,에 IBM MQ 관한 Windows것입니다.

API 엑시트 샘플 프로그램

이 샘플 API 종료 시, 환경 변수에 **MQAPI_TRACE_LOGFILE** 정의된 접두사를 사용하여 사용자가 지정한 파일에 'MQI' 추적 정보를 생성합니다.

API 엑시트에 대한 자세한 정보는 [702 페이지의 『멀티플랫폼에 API 엑시트 작성 및 컴파일』](#)의 내용을 참조하십시오.

소스

`amqsaxe0.c`

2진

`amqsaxe`

샘플 엑시트에 대한 구성

1. `ApiExitLocal`의 `qm.ini` 해당 섹션에 다음 정보를 추가하십시오.

Windows 이외의 플랫폼

```
ApiExitLocal:  
Sequence=100  
Function=EntryPoint
```

```
Module= MQ_INSTALLATION_PATH\samp\bin\amqsaxe
Name=SampleApiExit
```

여기서 `MQ_INSTALLATION_PATH`는 IBM MQ가 설치되어 있는 디렉토리입니다.

Windows

```
ApiExitLocal:
Sequence=100
Function=EntryPoint
Module= MQ_INSTALLATION_PATH\Tools\c\Samples\bin\amqsaxe
Name=SampleApiExit
```

여기서 `MQ_INSTALLATION_PATH`는 IBM MQ가 설치되어 있는 디렉토리입니다.

2. 환경 `MQAPI_TRACE_LOGFILE` 변수를 설정하십시오

```
MQAPI_TRACE_LOGFILE=/tmp/MqiTrace
```

3. 애플리케이션을 실행하십시오.

출력 파일은 /tmp 디렉터리에 다음과 같은 이름으로 생성됩니다: `MqiTrace.pid.tid.log`.

비동기 이용 샘플 프로그램

`amqscbf` 샘플 프로그램은 다중 큐에서 비동기적으로 메시지를 이용하기 위한 `MQCB` 및 `MQCTL`의 사용을 보여줍니다.

`amqscbf`는 AIX, Linux, and Windows 플랫폼에서 C 소스 코드 및 2진 클라이언트 및 서버 실행 파일로서 제공됩니다.

프로그램이 명령행에서 시작되고 다음의 선택적 매개변수를 사용합니다.

```
Usage: [Options] Queue Name {queue_name}
where Options are:
-m Queue Manager Name
-o Open options
-r Reconnect Type
  d Reconnect Disabled
  r Reconnect
  m Reconnect Queue Manager
```

여러 큐에서 메시지를 읽기 위해 둘 이상의 큐 이름을 제공하십시오(샘플에 의해 최대 10개 큐가 지원됩니다.).

참고: `Reconnect type` '은 클라이언트 프로그램에만 유효합니다.

예

예는 QL1에서 하나의 메시지를 읽은 후 중지되는 서버 프로그램으로 `amqscbf` 실행을 표시합니다.

IBM MQ 탐색기를 사용하여 QL1에 테스트 메시지를 넣으십시오. Enter를 눌러 프로그램을 중지하십시오.

```
C:\>amqscbf QL1
Sample AMQSCBF0 start

Press enter to end
Message Call (9 Bytes) :
Message 1

Sample AMQSCBF0 end
```

amqscbf에서 증명하는 사항

샘플은 도착하는 순서대로 여러 큐에서 메시지를 읽는 방법을 표시합니다. 동기 `MQGET`을 사용하여 더 많은 코드가 필요하게 됩니다. 비동기 이용의 경우 폴링은 필요하지 않으며 IBM MQ에서 스레드 및 스토리지 관리를 수행합니다. "real world" 예는 오류를 처리해야 합니다. 샘플에서 오류가 콘솔로 작성됩니다.

샘플 코드에는 다음 단계가 있습니다.

1. 단일 메시지 이용 콜백 함수를 정의하고,

```
void MessageConsumer(MQHCONN hConn,
                    MQMD * pMsgDesc,
                    MQGMO * pGetMsgOpts,
                    MQBYTE * Buffer,
                    MQCBC * pContext)
{ ... }
```

2. 큐 관리자에 연결하고,

```
MQCONN(QMName,&cnno,&Hcon,&CompCode,&CReason);
```

3. 입력 큐를 열고 각 큐를 MessageConsumer 콜백 함수와 연관시키고,

```
MQOPEN(Hcon,&od,0_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

cbd.CallbackFunction은 각 큐에 대해 설정할 필요가 없습니다. 입력 전용 필드입니다. 그러나 다른 콜백 함수를 각 큐와 연관시킬 수 있습니다.

4. 메시지 이용을 시작하십시오.

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. 사용자가 Enter를 누를 때까지 기다린 후 메시지 이용을 중지하십시오.

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. 마지막으로 큐 관리자에서 연결을 끊으십시오.

```
MQDISC(&Hcon,&CompCode,&Reason);
```

비동기 Put 샘플 프로그램

amqsapt 샘플 실행 및 비동기 Put 샘플 프로그램의 디자인에 대해 자세히 알아보십시오.

비동기 Put 샘플 프로그램은 비동기 MQPUT 호출을 사용하여 큐에 메시지를 넣은 후 MQSTAT 호출을 사용하여 상태 정보를 검색합니다. 다음 플랫폼별 프로그램 명칭은 해당 항목을 참조하십시오:

- ▶ **Linux** ▶ **AIX** [C 및 COBOL MQI 샘플에 표시된 기능 AIX and Linux](#)
- ▶ **Windows** [C 및 COBOL MQI 샘플에 표시된 기능 Windows](#)
- ▶ **IBM i** [C, COBOL 및 RPG MQI 샘플에 표시된 기능 IBM i](#)

amqsapt 샘플 실행

이 프로그램은 최대 6개의 매개변수를 사용합니다.

1. 대상 큐의 이름(필수)
2. 큐 관리자의 이름(선택사항)
3. 열기 옵션(선택사항)
4. 닫기 옵션(선택사항)
5. 대상 큐 관리자의 이름(선택사항)
6. 동적 큐의 이름(선택사항)

큐 관리자가 지정되지 않은 경우 amqsapt는 기본 큐 관리자에 연결합니다.

비동기 Put 샘플 프로그램의 디자인

프로그램은 메시지 넣기를 위한 대상 큐를 열기 위해 제공된 출력 옵션 또는 MQOO_OUTPUT 및 MQOO_FAIL_IF_QUIESCING 옵션과 함께 MQOPEN 호출을 사용합니다.

큐를 열 수 없는 경우 프로그램은 MQOPEN 호출에서 리턴한 이유 코드가 포함된 오류 메시지를 출력합니다. 프로그램을 단순하게 유지하기 위해 이 호출 및 후속 MQI 호출에서 프로그램은 여러 옵션의 기본값을 사용합니다.

입력의 각 행에 대해 프로그램은 텍스트를 버퍼로 해석하고 MQPMO_ASYNC_RESPONSE와 함께 MQPUT 호출을 사용하여 해당 행의 텍스트를 포함하는 데이터그램 메시지를 작성하고 비동기적으로 대상 큐에 메시지를 넣습니다. 프로그램은 입력의 끝에 도달하거나 MQPUT 호출에 실패할 때까지 계속됩니다. 프로그램이 입력의 끝에 도달하는 경우 MQCLOSE 호출을 사용하여 큐를 닫습니다.

그런 다음 프로그램은 MQSTS 구조를 리턴하는 MQSTAT 호출을 발행하고 성공적으로 넣어진 메시지 수, 경고와 함께 넣어진 메시지 수 및 실패 수를 포함하는 메시지를 표시합니다.

찾아보기 샘플 프로그램

찾아보기 샘플 프로그램은 MQGET 호출을 사용하여 큐에서 메시지를 찾습니다.

다음 플랫폼별 프로그램 명칭은 해당 항목을 참조하십시오:

- ▶ **Linux** ▶ **AIX** [C 및 COBOL MQI 샘플에 표시된 기능 AIX and Linux](#)
- ▶ **Windows** [C 및 COBOL MQI 샘플에 표시된 기능 Windows](#)
- ▶ **IBM i** [C, COBOL 및 RPG MQI 샘플에 표시된 기능 IBM i](#)

찾아보기 샘플 프로그램의 디자인

프로그램은 MQOO_BROWSE 옵션과 함께 MQOPEN 호출을 사용하여 대상 큐를 엽니다. 큐를 열 수 없는 경우 프로그램은 MQOPEN 호출에서 리턴한 이유 코드가 포함된 오류 메시지를 출력합니다.

큐의 각 메시지의 경우 프로그램은 큐에서 메시지를 복사하기 위해 MQGET 호출을 사용한 후 메시지에 포함된 데이터를 표시합니다. MQGET 호출은 이러한 옵션을 사용합니다.

MQGMO_BROWSE_NEXT

MQOPEN 호출 후 찾아보기 커서가 큐의 첫 번째 메시지 앞에 논리적으로 위치하므로 이 옵션은 호출이 처음 작성될 때 첫 번째 메시지가 리턴되도록 합니다.

MQGMO_NO_WAIT

큐 위의 메시지가 없는 경우 프로그램은 대기하지 않습니다.

MQGMO_ACCEPT_TRUNCATED_MSG

MQGET 호출은 고정된 크기의 버퍼를 지정합니다. 메시지가 이 버퍼보다 긴 경우 프로그램은 메시지가 잘렸다는 경과와 함께 잘린 메시지를 표시합니다.

호출은 이러한 필드를 검색한 메시지에 포함된 값으로 설정하기 때문에 프로그램은 각 MQGET 호출 후에 MQMD 구조의 *MsgId* 및 *CorrelId* 필드를 어떻게 지워야 하는지 방법을 보여줍니다. 이러한 필드를 지운다는 것은 연속적인 MQGET 호출이 메시지가 큐에 보유되는 순서대로 메시지를 검색한다는 것을 의미합니다.

프로그램은 큐의 끝까지 계속됩니다. MQGET 호출은 MQRC_NO_MSG_AVAILABLE 이유 코드를 리턴하며 프로그램은 경고 메시지를 표시합니다. MQGET 호출에 실패하는 경우 프로그램은 이유 코드를 포함하는 오류 메시지를 표시합니다.

그런 다음 프로그램은 MQCLOSE 호출을 사용하여 큐를 닫습니다.

▶ **ALW** **AIX, Linux, and Windows의 찾아보기 샘플 프로그램**

AIX, Linux, and Windows에서의 찾아보기 샘플 프로그램에 대해 학습하는 경우 다음 토픽을 사용하십시오.

프로그램의 C 버전은 2개의 매개변수를 사용합니다.

1. 소스 큐의 이름(필수)
2. 큐 관리자의 이름(선택사항)

큐 관리자가 지정되지 않은 경우 기본 관리자에 연결합니다. 예를 들어, 다음 중 하나를 입력하십시오.

- amqsgbr myqueue qmanagername
- amqsgbrc myqueue qmanagername

- amq0gbr0 myqueue

여기서 myqueue는 메시지가 표시될 큐의 이름이며 qmanagername은 myqueue를 소유하는 큐 관리자입니다. qmanagername을 생략하면 C 샘플을 실행하는 경우 기본 큐 관리자가 큐를 소유한다고 가정합니다.

COBOL 버전에는 매개변수가 없습니다. 기본 큐 관리자에 연결하고 실행하는 경우 다음 프롬프트가 표시됩니다.

```
Please enter the name of the target queue
```

각 메시지의 첫 50자만 표시되며, 그 이후에는 - - - truncated 가 표시됩니다.

IBM i IBM i의 찾아보기 샘플 프로그램

프로그램을 호출하면 각 프로그램은 사용자가 지정한 큐의 모든 메시지 사본을 검색합니다. 메시지는 큐에 남아 있습니다.

제공된 큐 SYSTEM.SAMPLE.LOCAL을 사용할 수 있습니다. Put 샘플 프로그램을 먼저 실행하여 큐에 일부 메시지를 넣으십시오. 동일한 로컬 큐에 대한 별명 이름인 SYSTEM.SAMPLE.ALIAS 큐를 사용할 수 있습니다. 프로그램은 큐의 끝에 도달하거나 MQI 호출에 실패할 때까지 계속됩니다.

C 샘플을 통해 일반적으로 Windows 시스템 샘플과 유사한 방법으로 두 번째 매개변수로서 큐 관리자 이름을 지정합니다. 예를 들면, 다음과 같습니다.

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER' 'QM01')
```

큐 관리자가 지정되지 않은 경우 기본 관리자에 연결합니다. 또한 이는 RPG 샘플과 관련됩니다. 그러나 RPG 샘플을 사용하여 기본으로 허용하는 대신 큐 관리자 이름을 제공해야 합니다.

관련 참조

1070 페이지의 『Browse RPG 샘플 프로그램 IBM i』

찾아보기 샘플 프로그램, AMQ3GBR4는 MQGET 호출을 사용하여 큐에서 메시지를 찾습니다.

ALW 브라우저 샘플 프로그램

브라우저 샘플 프로그램은 큐에 있는 모든 메시지의 메시지 디스크립터 및 메시지 콘텐츠 필드를 모두 읽고 기록합니다.

샘플 프로그램은 단지 기술을 보여주기 위해서가 아니라 유틸리티로서 작성됩니다. 다음 플랫폼별 프로그램 명칭은 해당 항목을 참조하십시오:

- ▶ Linux ▶ AIX [C 및 COBOL MQI 샘플에 표시된 기능 AIX and Linux](#)
- ▶ Windows [C 및 COBOL MQI 샘플에 표시된 기능 Windows](#)
- ▶ IBM i [C, COBOL 및 RPG MQI 샘플에 표시된 기능 IBM i](#)

이 프로그램은 이러한 위치 매개변수를 사용합니다.

1. 소스 큐의 이름(필수)
2. 큐 관리자의 이름(필수)
3. 특성에 대한 선택적 매개변수(선택사항)

다음 환경 변수를 사용하여 큐 관리자와의 인증에 사용되는 자격 증명을 제공하십시오:

MQSAMP_사용자_ID

큐 매니저에 사용자 ID와 비밀번호를 사용하여 인증하려면 연결 인증에 사용할 사용자 ID를 설정하십시오. 프로그램은 사용자 ID에 해당하는 비밀번호를 입력하도록 요청합니다.

Linux AIX MQSAMP_TOKEN

큐 관리자와 인증하기 위해 인증 토큰을 제공하려면 비어 있지 않은 값으로 설정하십시오. 프로그램이 인증 토큰을 요청합니다. 인증 토큰은 클라이언트 바인딩을 사용하는 amqsbcgc 샘플에서만 사용할 수 있습니다.

이러한 프로그램을 실행하려면 다음 명령 중 하나를 입력하십시오.

- amqsbcg myqueue qmanagername

- `amqsbcgc myqueue qmanagername`

여기서 `myqueue`는 메시지를 찾게 되는 큐의 이름이며 `qmanagername`은 `myqueue`를 소유하는 큐 관리자입니다.

큐에서 각 메시지를 읽고 다음을 `stdout`에 기록합니다.

- 형식화된 메시지 디스크립터 필드
- 메시지 데이터(가능한 경우 16진 및 문자 형식으로 덤프됨)

표 166. 특성 매개변수에 대해 허용 가능한 값	
값	동작
0	기본 동작입니다. 애플리케이션에 전달시키는 특성은 메시지가 검색되는 PropertyControl 큐 속성에 따라 달라집니다.
1	메시지 핸들이 작성되고 <code>MQGET</code>과 함께 사용됩니다. 메시지 디스크립터(또는 확장)에 포함된 특성을 제외하고, 메시지의 특성은 메시지 디스크립터와 유사한 방식으로 표시됩니다. 예를 들면, 다음과 같습니다. <pre>****Message properties**** property name: property value</pre> 또는 특성을 사용할 수 없는 경우, <pre>****Message properties**** None</pre> 숫자 값은 <code>printf</code>를 사용하여 표시되고, 문자열 값은 작은따옴표로 묶으며, 바이트 문자열은 메시지 디스크립터의 경우 <code>X</code>와 작은따옴표로 묶습니다.
2	메시지 디스크립터 특성만 리턴되도록 <code>MQGMO_NO_PROPERTIES</code> 가 지정됩니다.
3	모든 특성이 메시지 데이터에 리턴되도록 <code>MQGMO_PROPERTIES_FORCE_MQRFH2</code> 가 지정됩니다.
4	IBM MQ 특성이 포함되는지에 따라 모든 특성을 리턴할 수 있으며 그렇지 않은 경우 특성이 제거되도록 <code>MQGMO_PROPERTIES_COMPATIBILITY</code> 가 지정됩니다.

프로그램은 메시지의 처음 65535자만 출력하도록 제한되며, 더 긴 메시지가 읽히면 해당 `truncated msg` 사유로 실패합니다.

이 유틸리티의 출력 예는 [큐 브라우징](#)을 참조하십시오.

CICS 트랜잭션 샘플

샘플 CICS 트랜잭션 프로그램이 소스 코드의 경우 `amqscic0.ccs`라는 이름으로 실행 파일의 경우 `amqscic0`이라는 이름으로 제공됩니다. 표준 CICS 기능을 사용하여 트랜잭션을 빌드할 수 있습니다.

플랫폼에 필요한 명령의 세부사항은 [937 페이지의 『프로시저 애플리케이션 빌드』](#)의 내용을 참조하십시오.

트랜잭션은 기본 큐 관리자의 전송 큐 `SYSTEM.SAMPLE.CICS.WORKQUEUE`에서 메시지를 읽고 메시지의 전송 헤더에 포함되어 있는 이름의 로컬 큐에 메시지를 놓습니다. 모든 실패는 `SYSTEM.SAMPLE.CICS.DLQ` 큐로 송신됩니다.

참고: 이러한 큐 및 샘플 입력 큐를 작성하기 위해 샘플 MQSC 스크립트 `amqscic0.tst`를 사용할 수 있습니다.

연결 샘플 프로그램

연결 샘플 프로그램을 통해 클라이언트에서 `MQCONN` 호출과 해당 옵션을 탐색할 수 있습니다. 샘플은 `MQCONN` 호출을 사용하여 큐 관리자에 연결하고 `MQINQ` 호출을 사용하여 큐 관리자의 이름에 대해 조사한 후 이름을 표시합니다. 또한 `amqscnxc` 샘플 실행에 대해 알아보십시오.

참고: 연결 샘플 프로그램은 클라이언트 샘플입니다. 서버에서 컴파일하고 실행할 수 있지만 기능은 클라이언트에서만 의미가 있으며 클라이언트 실행 파일만 제공됩니다.

amqscnxc 샘플 실행

연결 샘플 프로그램의 명령행 구문은 다음과 같습니다.

```
amqscnxc [-x ConnName [-c SvrconnChannelName]] [-u User] [QMgrName]
```

매개변수는 선택적이며 QMgrName과 같이 지정된 경우에 마지막에 와야 하는 경우를 제외하고 매개변수의 순서는 중요하지 않습니다. 매개변수는 다음과 같습니다.

ConnName

서버 큐 관리자의 TCP/IP 연결 이름

TCP/IP 연결 이름을 지정하지 않은 경우 *ClientConnPtr*을 NULL로 설정하여 MQCONNx가 발행됩니다.

SvrconnChannelName

서버 연결 채널의 이름

TCP/IP 연결 이름을 지정하고 서버 연결 채널은 지정하지 않은 경우(반대 순서는 허용되지 않음) 샘플은 이름 SYSTEM.DEF.SVRCONN을 사용합니다.

사용자

연결 인증에 사용될 사용자 이름

이 매개변수를 지정하면 프로그램은 해당 사용자 ID와 함께 비밀번호를 입력하도록 프롬프트를 표시합니다.

QMgrName

대상 큐 관리자의 이름

대상 큐 관리자를 지정하지 않은 경우 샘플은 제공된 TCP/IP 연결 이름에서 청취 중인 큐 관리자에 연결됩니다.

참고: 유일한 매개변수로 물음표를 입력하거나 올바르지 않은 매개변수를 입력하면, 프로그램을 사용하는 방법을 설명하는 메시지가 표시됩니다.

명령행 옵션 없이 샘플을 실행하면, MQSERVER 환경 변수의 콘텐츠가 연결 정보를 판별하는 데 사용됩니다. (이 예에서 MQSERVER는 SYSTEM.DEF.SVRCONN/TCP/machine.site.company.com으로 설정됩니다.) 다음과 같은 출력이 표시됩니다.

```
Sample AMQSCNXC start
Connecting to the default queue manager
with no client connection information specified.
Connection established to queue manager machine

Sample AMQSCNXC end
```

샘플을 실행하고 TCP/IP 연결 이름과 서버 연결 채널 이름을 제공하지만 대상 큐 관리자 이름은 제공하지 않은 경우 다음과 같습니다.

```
amqscnxc -x machine.site.company.com -c SYSTEM.ADMIN.SVRCONN
```

기본 큐 관리자 이름이 사용되고 출력은 다음과 같습니다.

```
Sample AMQSCNXC start
Connecting to the default queue manager
using the server connection channel SYSTEM.ADMIN.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

샘플을 실행하고 TCP/IP 연결 이름과 대상 큐 관리자 이름을 제공하는 경우 다음과 같습니다.

```
amqscnxc -x machine.site.company.com MACHINE
```

다음과 같은 출력이 표시됩니다.

```
Sample AMQSCNXC start
Connecting to queue manager MACHINE
using the server connection channel SYSTEM.DEF.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

Data-Conversion 샘플 프로그램

데이터 변환 샘플 프로그램은 데이터 변환 엑시트 루틴의 스켈레톤입니다. 데이터 변환 샘플의 디자인에 대해 알아보십시오.

다음 플랫폼별 프로그램 명칭은 해당 항목을 참조하십시오:

-   C 및 COBOL MQI 샘플에 표시된 기능 [AIX and Linux](#)
-  C 및 COBOL MQI 샘플에 표시된 기능 [Windows](#)
-  C, COBOL 및 RPG MQI 샘플에 표시된 기능 [IBM i](#)

데이터 변환 샘플의 설계

각 데이터 변환 엑시트 루틴은 단일 이름이 지정된 메시지 형식을 변환합니다. 이 스켈레톤은 데이터 변환 엑시트 생성 유틸리티 프로그램에 의해 생성된 코드 단편에 대한 랩퍼로 사용하도록 작성되었습니다.

유틸리티는 각 데이터 구조에 하나의 코드 단편을 생성합니다. 이러한 여러 구조는 형식을 구성하므로 전체 형식의 데이터 변환을 수행하기 위한 루틴을 생성하도록 이 스켈레톤에 여러 코드 단편이 추가됩니다.

그런 다음 프로그램은 변환이 성공 또는 실패인지 확인하고 호출자에게 필요한 값을 리턴합니다.

데이터베이스 조정 샘플

IBM MQ가 IBM MQ 업데이트 및 동일한 작업 단위 내의 데이터베이스 업데이트 모두 조정할 수 있는 방법을 보여주는 두 가지 샘플이 제공됩니다.

이러한 샘플은 다음과 같습니다.

1. AMQXSAS0(C로 작성) 또는 AMQ0XAS0(COBOL로 작성), IBM MQ 작업 단위 내에서 단일 데이터베이스를 업데이트합니다.
2. AMQSXAG0(C로 작성) 또는 AMQ0XAG0(COBOL로 작성), AMQXSAB0(C로 작성) 또는 AMQ0XAB0(COBOL로 작성) 및 AMQSXAF0(C로 작성) 또는 AMQ0XAF0(COBOL로 작성), IBM MQ 작업 단위 내에서 두 개 데이터베이스를 함께 업데이트하며 다중 데이터베이스에 액세스할 수 있는 방법을 표시합니다. 이러한 샘플은 MQBEGIN 호출, 혼합된 SQL 및 IBM MQ 호출의 사용뿐만 아니라 데이터베이스에 연결하는 위치와 시기를 표시하기 위해 제공됩니다.

[994 페이지의 그림 144](#)에서는 제공된 샘플이 데이터베이스를 업데이트하는 데 어떻게 사용되는지를 표시합니다.

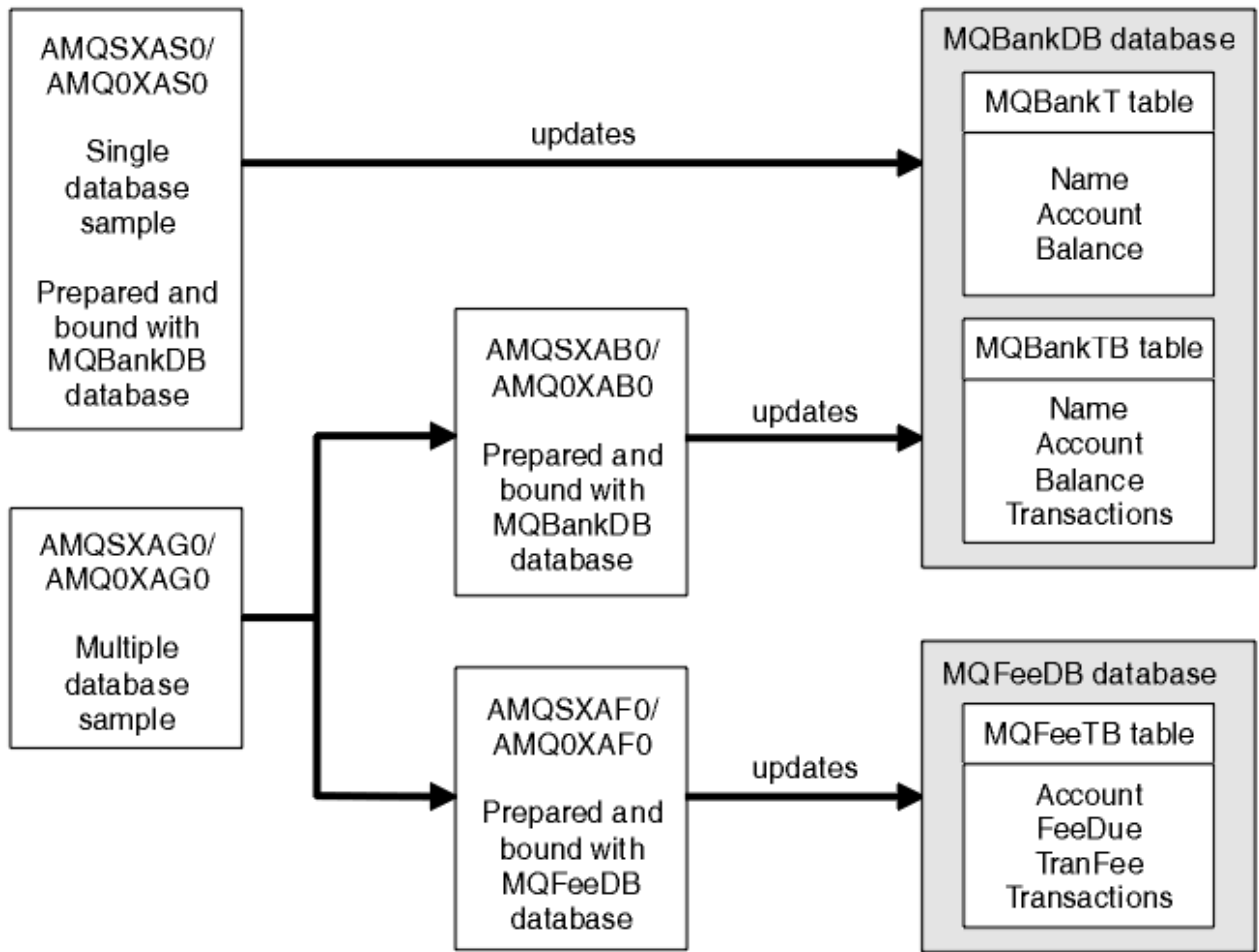


그림 144. 데이터베이스 조정 샘플

프로그램은 (동기점 아래에 있는) 큐에서 메시지를 읽은 후 메시지의 해당 정보를 사용하여 데이터베이스에서 관련 정보를 얻고 해당 사항을 업데이트합니다. 그러면 데이터베이스의 새로운 상태가 출력됩니다.

프로그램 로직은 다음과 같습니다.

1. 프로그램 인수의 입력 큐 이름을 사용합니다.
2. MQCONN을 사용하여 기본 큐 관리자(또는 선택적으로 C로 제공된 이름)에 연결합니다.
3. 실패가 없는 동안 (MQOPEN을 사용하여) 입력에 대한 큐를 엽니다.
4. MQBEGIN을 사용하여 작업 단위를 시작합니다.
5. 동기점 아래에 있는 큐에서 (MQGET을 사용하여) 다음 메시지를 가져옵니다.
6. 데이터베이스에서 정보를 가져옵니다.
7. 데이터베이스의 정보를 업데이트합니다.
8. MQCMIT을 사용하여 변경사항을 커밋합니다.
9. 업데이트된 정보를 출력합니다(실패로 계산할 수 있는 메시지가 없고 루프가 종료됨).
10. MQCLOSE을 사용하여 큐를 닫습니다.
11. MQDISC를 사용하여 큐에서 연결을 끊습니다.

메시지가 처리되는 동안 데이터베이스에서 읽기(즉, 다중 인스턴스)가 잠기도록 SQL 커서가 샘플에 사용되므로 이러한 프로그램의 여러 인스턴스를 동시에 실행할 수 있습니다. 커서는 명시적으로 열리지만 MQCMIT 호출에 의해 내재적으로 닫힙니다.

단일 데이터베이스 샘플(AMQSXA0 또는 AMQ0XA0)에는 SQL CONNECT 명령문이 없으며 MQBEGIN 호출로 데이터베이스에 대한 연결이 IBM MQ에 의해 내재적으로 작성됩니다. 일부 데이터베이스 제품에서 단 하나의 활성 연결을 허용하는 것처럼 여러 데이터베이스 샘플(AMQSXA0 또는 AMQ0XA0, AMQSXA0 또는

AMQOXAB0, AMQSXAFO 또는 AMQ0XAFO)은 SQL CONNECT 명령문을 가지고 있습니다. 사용자 데이터베이스 제품에 해당하지 않는 경우 또는 여러 데이터베이스 제품에서 단일 데이터베이스에 액세스하는 경우 SQL CONNECT 명령문을 제거할 수 있습니다.

샘플은 IBM Db2 데이터베이스 제품과 함께 준비되므로 다른 데이터베이스 제품에 대한 작업을 위해서는 샘플을 수정해야 할 수 있습니다.

SQL 오류 검사는 Db2에서 제공하는 UTIL.C 및 CHECKERR.CBL로 작성된 루틴을 사용합니다. 컴파일 및 연결하기 전에 컴파일하거나 대체해야 합니다.

참고: SQL 오류 검사를 위해 Micro Focus COBOL 소스 CHECKERR.MFC를 사용 중인 경우 AMQOXASO를 올바르게 링크하려면 프로그램 ID를 대문자로 즉, CHECKERR로 변경해야 합니다.

데이터베이스 및 테이블 작성

샘플을 컴파일하기 전에 데이터베이스 및 테이블을 작성하십시오.

데이터베이스를 작성하려면, 데이터베이스 제품에 대한 일반적인 방법을 사용하십시오. 예를 들면, 다음과 같습니다.

```
DB2 CREATE DB MQBankDB
DB2 CREATE DB MQFeeDB
```

다음과 같이 SQL문을 사용하여 테이블을 작성하십시오.

C의 경우:

```
EXEC SQL CREATE TABLE MQBankT(Name          VARCHAR(40) NOT NULL,
                                Account       INTEGER   NOT NULL,
                                Balance       INTEGER   NOT NULL,
                                PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQBankTB(Name          VARCHAR(40) NOT NULL,
                                Account       INTEGER   NOT NULL,
                                Balance       INTEGER   NOT NULL,
                                Transactions  INTEGER,
                                PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQFeeTB(Account      INTEGER   NOT NULL,
                                FeeDue       INTEGER   NOT NULL,
                                TranFee     INTEGER   NOT NULL,
                                Transactions  INTEGER,
                                PRIMARY KEY (Account));
```

COBOL의 경우:

```
EXEC SQL CREATE TABLE
  MQBankT(Name          VARCHAR(40) NOT NULL,
           Account      INTEGER   NOT NULL,
           Balance      INTEGER   NOT NULL,
           PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQBankTB(Name          VARCHAR(40) NOT NULL,
            Account      INTEGER   NOT NULL,
            Balance      INTEGER   NOT NULL,
            Transactions  INTEGER,
            PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQFeeTB(Account      INTEGER   NOT NULL,
            FeeDue      INTEGER   NOT NULL,
            TranFee     INTEGER   NOT NULL,
            Transactions  INTEGER,
            PRIMARY KEY (Account))
  END-EXEC.
```

다음과 같이 SQL문을 사용하여 테이블에 데이터를 입력하십시오.

```
EXEC SQL INSERT INTO MQBankT VALUES ('Mr Fred Bloggs',1,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Mrs S Smith',2,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Ms Mary Brown',3,0);
```

```

:
EXEC SQL INSERT INTO MQBankTB VALUES ('Mr Fred Bloggs',1,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Mrs S Smith',2,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Ms Mary Brown',3,0,0);
:
EXEC SQL INSERT INTO MQFeeTB VALUES (1,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (2,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (3,0,50,0);
:

```

참고: COBOL의 경우 동일한 SQL문을 사용하지만 각 행의 끝에 END_EXEC를 추가하십시오.

샘플 사전 컴파일링, 컴파일링 및 연결

C 및 COBOL로 작성된 샘플의 사전 컴파일링, 컴파일링 및 연결에 대해 알아보십시오.

.SQC 파일(C의 경우) 및 .SQB 파일(COBOL의 경우)을 사전 컴파일하고 적절한 데이터베이스에 대해 바인드하여 .C 또는 .CBL 파일을 작성하십시오. 이를 수행하려면, 데이터베이스 제품에 대한 일반적인 방법을 사용하십시오.

C에서 사전 컴파일링

```

db2 connect to MQBankDB
db2 prep AMQXSAS0.SQC
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQXSAB0.SQC
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQXAF0.SQC
db2 connect reset

```

COBOL에서 사전 컴파일링

```

db2 connect to MQBankDB
db2 prep AMQOXAS0.SQB bindfile target ibmcob
db2 bind AMQOXAS0.BND
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQOXAB0.SQB bindfile target ibmcob
db2 bind AMQOXAB0.BND
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQOXAF0.SQB bindfile target ibmcob
db2 bind AMQOXAF0.BND
db2 connect reset

```

컴파일링 및 연결

다음 샘플 명령은 기호 *DB2TOP* 및 *MQ_INSTALLATION_PATH*를 사용합니다. *DB2TOP*은 Db2 제품의 설치 디렉토리를 나타냅니다. *MQ_INSTALLATION_PATH*는 IBM MQ가 설치되어 있는 상위 레벨 디렉토리를 나타냅니다.

- ▶ **AIX** AIX에서 디렉토리 경로는 다음과 같습니다.

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

- ▶ **Windows** Windows 시스템에서 디렉토리 경로는 제품을 설치할 때 선택한 경로에 따라 달라집니다. 기본 설정을 선택한 경우 경로는 다음과 같습니다.

```
c:\sqllib
```

참고: Windows 시스템에서 링크 명령을 발행하기 전에 LIB 환경 변수에 Db2 및 IBM MQ 라이브러리에 대한 경로가 포함되어 있는지 확인하십시오.

다음 파일을 임시 디렉토리에 복사하십시오.

- IBM MQ 설치의 amqsxag0.c 파일

참고: 이 파일은 다음 디렉토리에서 찾을 수 있습니다.

-   AIX and Linux 시스템:

```
MQ_INSTALLATION_PATH/samp/xatm
```

-  Windows 시스템:

```
MQ_INSTALLATION_PATH\tools\c\samples\xatm
```

- .sqc 소스 파일, amqsxas0.sqc, amqsxaf0.sqc 및 amqsxab0.sqc을(를) 사전 컴파일하여 얻은 .c 파일.
- Db2 설치의 util.c 및 util.h 파일.

참고: 이러한 파일은 디렉토리에서 찾을 수 있습니다.

```
DB2TOP/samples/c
```

사용 중인 플랫폼에 대한 다음 컴파일러 명령을 사용하여 각 .c 파일에 대한 오브젝트 파일을 빌드하십시오.

-  AIX

```
xlc_r -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o  
FILENAME.o FILENAME.c
```

-  Windows 시스템

```
cl /c /I MQ_INSTALLATION_PATH\tools\c\include /I DB2TOP\include  
FILENAME.c
```

사용 중인 플랫폼에 대한 다음 링크 명령을 사용하여 amqsxag0 실행 파일을 빌드하십시오.

-  AIX

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

-  Windows 시스템

```
link util.obj amqsxaf0.obj amqsxab0.obj amqsxag0.obj mqm.lib db2api.lib  
/out:amqsxag0.exe
```

사용 중인 플랫폼에 다음 컴파일 및 링크 명령을 사용하여 amqsxas0 실행 파일을 빌드하십시오.

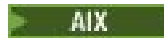
-  AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2  
-L MQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
```

-  Windows 시스템

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

추가 정보

-  Oracle 를 작업 AIX 중이며 해당 파일에 접근하려면 xlc_r 컴파일러를 사용하고 libmqm_r.a 로 링크하십시오.

샘플 실행

이 정보를 사용하여 C 및 COBOL에서 데이터베이스 조정 샘플을 실행하기 전에 큐 관리자를 구성하는 방법에 대해 알아보십시오.

샘플을 실행하기 전에 사용하고 있는 데이터베이스 제품으로 큐 관리자를 구성하십시오. 수행 방법에 대한 정보는 [시나리오 1: 큐 관리자가 조정 수행의 내용을 참조하십시오](#).

다음 제목은 C 및 COBOL에서 샘플을 실행하는 방법에 대한 정보를 제공합니다.

- [998 페이지의 『C 샘플』](#)
- [998 페이지의 『COBOL 샘플』](#)

C 샘플

메시지는 큐에서 읽게 될 다음 형식이어야 합니다.

```
UPDATE Balance change=nnn WHERE Account=nnn
```

큐에 메시지를 넣는 데 AMQSPUT을 사용할 수 있습니다.

데이터베이스 조정 샘플은 두 개의 매개변수를 사용합니다.

1. 큐 이름(필수)
2. 큐 관리자 이름(선택사항)

singDBQ라는 큐를 사용하여 singDBQM이라는 단일 데이터베이스 샘플에 대해 큐 관리자를 작성하고 구성했으며 다음과 같이 Mr Fred Blog의 계정이 50까지 증가되었다고 가정합니다.

```
AMQSPUT singDBQ singDBQM
```

다음 메시지의 키:

```
UPDATE Balance change=50 WHERE Account=1
```

여러 메시지를 큐에 넣을 수 있습니다.

```
AMQSXAS0 singDBQ singDBQM
```

그런 다음 Mr Fred Blog 계정의 업데이트된 상태가 출력됩니다.

multDBQ라는 큐를 사용하여 multDBQM이라는 다중 데이터베이스 샘플에 대해 큐 관리자를 작성하고 구성했으며 다음과 같이 Ms Mary Brown의 계정이 75까지 감소되었다고 가정합니다.

```
AMQSPUT multDBQ multDBQM
```

다음 메시지의 키:

```
UPDATE Balance change=-75 WHERE Account=3
```

여러 메시지를 큐에 넣을 수 있습니다.

```
AMQSXAG0 multDBQ multDBQM
```

그런 다음 Ms Mary Brown 계정의 업데이트된 상태가 출력됩니다.

COBOL 샘플

메시지는 큐에서 읽게 될 다음 형식이어야 합니다.

```
UPDATE Balance change=snnnnnnnn WHERE Account=nnnnnnnn
```

단순함을 위해 Balance change는 서명된 8자 숫자여야 하며 Account는 8자 숫자여야 합니다.

샘플 AMQSPUT을 메시지에 큐를 넣는 데 사용할 수 있습니다.

샘플은 매개변수를 사용하지 않으며 기본 큐 관리자를 사용합니다. 언제든지 샘플 중 하나만 실행하도록 구성할 수 있습니다. `singDBQ`라는 큐를 사용하여 단일 데이터베이스 샘플에 대해 큐 관리자를 구성했으며 다음과 같이 Mr Fred Blog의 계정이 50까지 증가되었다고 가정합니다.

```
AMQSPUT singDBQ
```

다음 메시지의 키:

```
UPDATE Balance change=+000000050 WHERE Account=000000001
```

여러 메시지를 큐에 넣을 수 있습니다.

```
AMQ0XAS0
```

큐의 이름을 입력하십시오.

```
singDBQ
```

그런 다음 Mr Fred Blog 계정의 업데이트된 상태가 출력됩니다.

`multDBQ`라는 큐를 사용하여 다중 데이터베이스 샘플에 대해 큐 관리자를 구성했으며 다음과 같이 Ms Mary Brown의 계정이 75까지 감소되었다고 가정합니다.

```
AMQSPUT multDBQ
```

다음 메시지의 키:

```
UPDATE Balance change=-000000075 WHERE Account=000000003
```

여러 메시지를 큐에 넣을 수 있습니다.

```
AMQ0XAGO
```

큐의 이름을 입력하십시오.

```
multDBQ
```

그런 다음 Ms Mary Brown 계정의 업데이트된 상태가 출력됩니다.

데드-레터 큐 핸들러 샘플

샘플 데드레터 큐 핸들러가 제공되며, 실행 파일 버전의 이름은 `amqsd1q`와 다른 데드 레터 큐 **`RUNMQDLQ`** 핸들러를 원한다면, 샘플의 소스 코드를 기반으로 사용할 수 있습니다.

샘플은 제품 내에 제공된 데드 레터 핸들러와 유사하지만 추적 및 오류 보고가 다릅니다. 두 개의 환경 변수를 사용할 수 있습니다.

`ODQ TRACE`

추적을 켜려면 **`YES`** 또는 **`yes`**로 설정하십시오.

`ODQ MSG`

오류 및 정보 메시지를 포함하는 파일의 이름으로 설정하십시오. 제공된 파일의 이름은 `amqsd1q.msg`입니다.

`export` 또는 **`set`** 명령을 사용하여 이러한 변수를 환경에 알릴 필요가 있습니다. 플랫폼에 따라 **`unset`** 명령을 사용하여 추적이 꺼집니다.

오류 메시지 파일인 `amqsd1q.msg`을 자신의 요구 사항에 맞게 수정할 수 있습니다. 샘플은 메시지를 오류 로그 IBM MQ 파일이 **아닌** 표준 출력(`stdout`)으로 출력합니다.

데드레터 핸들러의 작동 방식 및 실행 방법에 대한 자세한 내용은 [IBM MQ 데드레터 큐에서 메시지 처리](#) 또는 해당 플랫폼의 시스템 관리 가이드를 참조하십시오.

분배 목록 샘플 프로그램

분배 목록 샘플 amqsptl0는 여러 메시지 큐에 메시지를 넣는 예를 제공합니다. MQPUT 샘플 amqsput0를 기반으로 합니다.

분배 목록 샘플 amqsptl0 실행

분배 목록 샘플은 Put 샘플과 유사한 방법으로 실행됩니다.

다음 매개변수를 사용합니다.

- 큐의 이름
- 큐 관리자의 이름

이러한 값은 쌍으로 입력됩니다. 예를 들면, 다음과 같습니다.

```
amqsptl0 queue1 qmanagername1 queue2 qmanagername2
```

큐는 MQOPEN을 사용하여 열리며 MQPUT을 사용하여 메시지를 큐에 넣습니다. 큐 또는 큐 관리자 이름이 인식되지 않으면 이유 코드가 리턴됩니다.

메시지가 큐 관리자 간에 이동할 수 있도록 큐 관리자 간에 채널의 정의해야 합니다. 샘플 프로그램은 사용자를 위해 이를 수행하지 않습니다.

분배 목록 샘플의 설계

메시지 레코드 넣기(MQPMR)는 각 대상에 대한 메시지 속성을 지정합니다. 샘플은 *MsgId* 및 *CorrelId*에 대한 값을 제공하며 MQMD 구조에 지정된 값을 대체합니다.

MQPMO 구조의 *PutMsgRecFields* 필드는 MQPMR에 있는 필드를 나타냅니다.

```
MQLONG PutMsgRecFields=MQPMRF_MSG_ID + MQPMRF_CORREL_ID;
```

다음으로 샘플은 응답 레코드 및 오브젝트 레코드를 할당합니다. 오브젝트 레코드(MQOR)는 *ObjectName* 및 *ObjectQMgrName*과 같이 하나 이상의 이름 쌍과 짝수의 이름이 필요합니다.

다음 단계는 MQCONN을 사용하여 큐 관리자에 연결하는 작업이 포함됩니다. 샘플은 MQOR의 첫 번째 큐와 연결된 큐 관리자에 연결하려고 시도합니다. 실패하는 경우 차례로 오브젝트 레코드를 통과합니다. 큐 관리자에 연결할 수 없는 경우 사용자에게 공지되고 프로그램이 종료됩니다.

MQOPEN을 사용하여 대상 큐가 열리며 MQPUT을 사용하여 메시지를 큐에 넣습니다. 문제점이 발생하거나 실패하는 경우 이유 레코드(MQRR)에 보고됩니다.

마지막으로 대상 큐는 MQCLOSE를 사용하여 닫히고 MQDISC를 사용하여 프로그램은 큐 관리자의 연결을 끊습니다. 동일한 응답 레코드가 *CompCode* 및 *Reason*을 명시하며 각 호출에 사용됩니다.

Echo 샘플 프로그램

Echo 샘플 프로그램은 메시지 큐의 메시지를 응답 큐로 에코합니다.

다음 플랫폼별 프로그램 명칭은 해당 항목을 참조하십시오:

- ▶ **Linux** ▶ **AIX** [C 및 COBOL MQI 샘플에 표시된 기능 AIX and Linux](#)
- ▶ **Windows** [C 및 COBOL MQI 샘플에 표시된 기능 Windows](#)
- ▶ **IBM i** [C, COBOL 및 RPG MQI 샘플에 표시된 기능 IBM i](#)

프로그램은 트리거된 프로그램으로 실행하기 위한 것입니다.

IBM i, AIX, Linux, and Windows 시스템에서 유일한 입력은 대상 큐의 이름 및 큐 관리자를 포함하는 MQTMC2(트리거 메시지) 구조입니다. COBOL 버전은 기본 큐 관리자를 사용합니다.

IBM i IBM i에서 작업할 트리거링 프로세스의 경우 사용하려는 Echo 샘플 프로그램이 SYSTEM.SAMPLE.ECHO 큐에 도착하는 메시지에 의해 트리거됩니다. 이를 수행하려면, 프로세스 정의 SYSTEM.SAMPLE.ECHOPROCESS의 *ApplId* 필드에 사용하려는 Echo 샘플 프로그램의 이름을 지정하십시오. (이를 위해 CHGMQMPRC 명령을 사용할 수 있습니다. 자세한 내용은 [MQ 프로세스 변경\(CHGMQMPRC\)](#)을 참조하십시오.) 샘플 큐에는 트리거 유형 FIRST가 있으므로 이미 큐에 메시지가 있는 경우 요청 샘플을 실행하기 전에 Echo 샘플은 송신하는 메시지에 의해 트리거되지 않습니다.

정의를 올바르게 설정한 경우 먼저 하나의 작업에서 AMQSERV4를 시작한 후 다른 작업에서 AMQSREQ4를 시작하십시오. AMQSERV4 대신에 AMQSTRG4를 사용할 수 있지만, 잠재적인 작업 제출 지연으로 인해 발생하는 상황에 쉽게 대응하지 못할 수 있습니다.

요청 샘플 프로그램을 사용하여 메시지를 SYSTEM.SAMPLE.ECHO 큐에 보냅니다. Echo 샘플 프로그램은 요청 메시지에 데이터를 포함하는 응답 메시지를 요청 메시지에 지정된 응답 큐로 보냅니다.

Echo 샘플 프로그램의 디자인

프로그램은 시작했을 때 전달된 트리거 메시지 구조에서 이름 지정된 큐를 엽니다. (명확성을 위해 이를 요청 큐라고 합니다.) 프로그램은 MQOPEN 호출을 사용하여 공유 입력에 대한 이 큐를 엽니다.

프로그램은 MQGET 호출을 사용하여 이 큐에서 메시지를 제거합니다. 이 호출은 5초의 대기 간격으로 MQGMO_ACCEPT_TRUNCATED_MSG, MQGMO_CONVERT 및 MQGMO_WAIT 옵션을 사용합니다. 프로그램은 요청 메시지가 있는지 확인하기 위해 각 메시지의 디스크립터를 테스트합니다. 요청 메시지가 아닌 경우 프로그램은 메시지를 제거하고 경고 메시지를 표시합니다.

각 입력 행의 경우 프로그램은 텍스트를 버퍼로 해석하고 MQPUT1 호출을 사용하여 해당 행의 텍스트를 포함하는 요청 메시지를 응답 대상 큐에 넣습니다.

MQGET 호출에 실패하는 경우, 프로그램은 응답 대상 큐에 보고 메시지를 넣고 메시지 디스크립터의 *Feedback* 필드를 MQGET에서 리턴한 이유 코드로 설정합니다.

요청 큐에 남아 있는 메시지가 없는 경우 프로그램은 해당 큐를 닫고 큐 관리자의 연결을 끊습니다.

IBM i IBM i에서 이 상황에 대한 샘플이 제공되지 않아도 프로그램은 IBM MQ for IBM i 이외의 플랫폼에서 큐에 송신된 메시지에 응답합니다. ECHO 프로그램이 작동하게 하려면,

- 텍스트 요청 메시지를 전송하기 위해 **Format, Encoding** 및 **CCSID** 매개변수를 올바르게 지정하여 프로그램을 작성하십시오.

ECHO 프로그램은 필요한 경우 큐 관리자에게 메시지 데이터 변환을 수행하도록 요청합니다.

- 작성한 프로그램이 응답에 대해 유사한 변환을 제공하지 않는 경우 IBM MQ for IBM i 송신 채널에 CONVERT(*YES)를 지정하십시오.

관련 참조

1070 페이지의 『에코 RPG 샘플 프로그램 IBM i』

에코 샘플 프로그램은 응답 큐로 송신된 메시지를 리턴합니다. 프로그램의 이름은 AMQ3ECH4입니다.

Get 샘플 프로그램

Get 샘플 프로그램은 MQGET 호출을 사용하여 큐에서 메시지를 가져옵니다.

다음 플랫폼별 프로그램 명칭은 해당 항목을 참조하십시오:

- **Linux** **AIX** C 및 COBOL MQI 샘플에 표시된 기능 [AIX and Linux](#)
- **Windows** C 및 COBOL MQI 샘플에 표시된 기능 [Windows](#)
- **IBM i** C, COBOL 및 RPG MQI 샘플에 표시된 기능 [IBM i](#)

Get 샘플 프로그램의 디자인

프로그램은 MQOO_INPUT_AS_Q_DEF 옵션과 함께 MQOPEN 호출을 사용하여 대상 큐를 엽니다. 큐를 열 수 없는 경우 프로그램은 MQOPEN 호출에서 리턴한 이유 코드가 포함된 오류 메시지를 표시합니다.

큐의 각 메시지의 경우 프로그램은 큐에서 메시지를 제거하기 위해 MQGET 호출을 사용한 후 메시지에 포함된 데이터를 표시합니다. MQGET 호출은 큐에 메시지가 없는 경우 프로그램에서 이 기간 동안 대기하도록 15초의 *WaitInterval*을 지정하여 MQGMO_WAIT 옵션을 사용합니다. 이 간격이 끝나기 전에 메시지가 도착하지 않는 경우 호출은 실패하고 MQRC_NO_MSG_AVAILABLE 이유 코드를 리턴합니다.

호출은 이러한 필드를 검색한 메시지에 포함된 값으로 설정하기 때문에 프로그램은 각 MQGET 호출 후에 MQMD 구조의 *MsgId* 및 *CorrelId* 필드를 어떻게 지워야 하는지 방법을 보여줍니다. 이러한 필드를 지운다는 것은 연속적인 MQGET 호출이 메시지가 큐에 보유되는 순서대로 메시지를 검색한다는 것을 의미합니다.

MQGET 호출은 고정된 크기의 버퍼를 지정합니다. 메시지가 이 버퍼보다 더 긴 경우 호출에 실패하고 프로그램은 중지됩니다.

프로그램은 MQGET 호출에서 MQRC_NO_MSG_AVAILABLE 이유 코드를 리턴하거나 MQGET 호출에 실패할 때까지 계속됩니다. 호출에 실패하는 경우 프로그램은 이유 코드를 포함하는 오류 메시지를 표시합니다.

그런 다음 프로그램은 MQCLOSE 호출을 사용하여 큐를 닫습니다.

관련 참조

1071 페이지의 『Get RPG 샘플 프로그램 IBM i』

Get 샘플 프로그램, AMQ3GET4는 MQGET 호출을 사용하여 큐에서 메시지를 가져옵니다.

amqsget 및 amqsgetc 샘플 실행

이러한 프로그램 각각 다음 위치 매개변수를 사용합니다.

1. 소스 큐의 이름(필수)

2. 큐 관리자의 이름(선택사항)

큐 관리자가 지정되지 않은 경우, 기본 큐 **amqsget** 관리자에 연결하며 *MQSERVER* 환경 변수 또는 클라이언트 **amqsgetc** 채널 정의 파일로 식별된 큐 관리자에 연결합니다.

3. 열기 옵션(선택사항)

열기 옵션이 지정되지 않은 경우 샘플은 이러한 두 가지 옵션의 결합인 8193 값을 사용합니다.

- MQOO_INPUT_AS_Q_DEF
- MQOO_FAIL_IF_QUIESCING

4. 닫기 옵션(선택사항)

닫기 옵션이 지정되지 않은 경우 샘플은 값이 0인 MQCO_NONE을 사용합니다.

다음 환경 변수를 사용하여 큐 관리자와의 인증에 사용되는 자격 증명을 제공하십시오:

MQSAMP_사용자_ID

큐 매니저에 사용자 ID와 비밀번호를 사용하여 인증하려면 연결 인증에 사용할 사용자 ID를 설정하십시오. 프로그램은 사용자 ID에 해당하는 비밀번호를 입력하도록 요청합니다.

Linux AIX MQSAMP_TOKEN

큐 관리자와 인증하기 위해 인증 토큰을 제공하려면 비어 있지 않은 값으로 설정하십시오. 프로그램이 인증 토큰을 요청합니다. 인증 토큰은 클라이언트 바인딩을 사용하는 **amqsgetc** 샘플에서만 사용할 수 있습니다.

이러한 프로그램을 실행하려면 다음 중 하나를 입력하십시오.

- **amqsget** *myqueue* *qmanagername*
- **amqsgetc** *myqueue* *qmanagername*

여기서 *myqueue*는 프로그램이 메시지를 가져오는 큐의 이름이며 *qmanagername*은 *myqueue*를 소유하는 큐 관리자입니다.

amqsget 및 amqsgetc 사용

amqsget는 큐 관리자에 접속하기 위해 공유 메모리를 사용하여 큐 관리자에 대한 로컬 연결을 수행하고 이는 큐 관리자가 상주하는 시스템에서만 실행할 수 있지만 **amqsgetc**는 동일한 시스템에서 큐 관리자에 연결하는 경우에도 클라이언트 스타일 연결을 수행함을 참고하십시오.

amqsgetc를 사용할 때 큐 관리자 호스트 또는 IP 주소 및 큐 관리자 리스너 포트에 관련하여 큐 관리자에 실제로 도달하는 방법에 대한 애플리케이션 세부사항을 제공해야 합니다.

일반적으로 이는 환경 **MQSERVER** 변수를 사용하거나 클라이언트 채널 정의 테이블을 통해 연결 세부 정보를 정의함으로써 수행됩니다. 이 테이블 역시 환경 변수를 통해 **amqsgetc** 제공될 수 있습니다. 예를 들어 **MQCCDTURL**을 참조하십시오.

로컬에서 큐 매니저에 연결하는 예시입니다 **MQSERVER**. 해당 큐 매니저는 포트 1414에서 리스너를 실행 중이며 기본 서버 연결 채널을 사용합니다:

```
export MQSERVER="SYSTEM.DEF.SVRCONN/TCP/ localhost(1414)"
```

고가용성 샘플 프로그램

amqsghac, **amqsphac** 및 **amqsmhac** 고가용성 샘플 프로그램은 자동화된 클라이언트 다시 연결을 사용하여 큐 관리자 실패에 따른 복구를 보여줍니다. **amqsfhac**는 네트워크 스토리지를 사용하는 큐 관리자가 실패에 따른 데이터 무결성을 유지보수하는지 확인합니다.

amqsghac, **amqsphac** 및 **amqsmhac** 프로그램은 명령행에서 시작되며 여러 인스턴스 큐 관리자의 한 인스턴스가 실패한 후 재연결을 입증하기 위해 결합에 사용될 수 있습니다.

또는 **amqsghac**, **amqsphac** 및 **amqsmhac** 샘플을 사용하여 일반적으로 큐 관리자 그룹으로 구성된 단일 인스턴스 큐 관리자에 대한 클라이언트 재연결을 보여줄 수 있습니다.

구성하기 쉽도록 예를 단순하게 유지하기 위해 시작, 중지 후 다시 시작되는 단일 인스턴스 큐 관리자에 재연결하는 샘플 프로그램을 표시합니다. 1005 페이지의 『큐 관리자 설정 및 제어』의 내용을 참조하십시오.

amqmfscck와 병렬로 **amqsfhac**를 사용하여 파일 시스템 무결성을 확인합니다. 자세한 정보는 **amqmfscck**(파일 시스템 검사) 및 공유 파일 시스템 작동 확인을 참조하십시오.

amqsphac queueName [qMgrName]

- **amqsphac**는 IBM MQ MQI client 애플리케이션입니다. 이는 일련의 메시지를 각 메시지 간의 2초 지연 간격으로 큐에 넣으며 해당 이벤트 핸들러로 보낸 이벤트를 표시합니다.
- 큐에 메시지를 넣는 데 사용되는 동기점이 없습니다.
- 동일한 큐 관리자 그룹에 있는 모든 큐 관리자로 재연결이 작성될 수 있습니다.

amqsghac queueName [qMgrName]

- **amqsghac**는 IBM MQ MQI client 애플리케이션입니다. 이는 큐에서 메시지를 가져오고 해당 이벤트 핸들러로 보낸 이벤트를 표시합니다.
- 큐에서 메시지를 가져오는 데 사용되는 동기점이 없습니다.
- 동일한 큐 관리자 그룹에 있는 모든 큐 관리자로 재연결이 작성될 수 있습니다.

amqsmhac -s sourceQueueName -t targetQueueName [-m qMgrName] [-w waitInterval]

- **amqsmhac**는 IBM MQ MQI client 애플리케이션입니다. 이는 한 큐에서 다른 큐로 메시지를 복사하고, 프로그램이 완료되기 전에 마지막 메시지가 수신된 후 15분의 기본 대기 간격을 사용합니다.
- 메시지가 동기점 내에 복사됩니다.
- 재연결은 동일한 큐 관리자로만 작성될 수 있습니다.

amqsfhac QueueManagerName QueueName SideQueueName InTransactionCount RepeatCount (0 | 1 | 2)

- **amqsfhac**는 IBM MQ MQI client 애플리케이션입니다. NAS 또는 클러스터 파일 시스템과 같은 네트워크 스토리지를 사용하는 IBM MQ 다중 인스턴스 큐 관리자가 데이터 무결성을 유지보수하는지 확인합니다. 공유 파일 시스템 작동 확인에서 **amqsfhac**를 실행하는 단계를 수행하십시오.
- **QueueManagerName**에 연결할 때 **MQCNO_RECONNECT_Q_MGR** 옵션을 사용합니다. 큐 관리자가 실패하면 자동으로 다시 연결합니다.

- 큐 관리자가 몇 번이고 장애 복구하게 하는 시간 동안 *InTransactionCount*RepeatCount* 지속 메시지를 *QueueName*에 넣습니다. **amqsfhac**는 매번 큐 관리자에 다시 연결하여 계속합니다. 이는 메시지가 유실되지 않았는지 확인하기 위한 테스트입니다.
- *InTransactionCount* 메시지를 각 트랜잭션에 넣습니다. 트랜잭션은 *RepeatCount*번 만큼 반복됩니다. 트랜잭션 내에 실패가 발생하는 경우 **amqsfhac**가 큐 관리자에 다시 연결할 때 **amqsfhac**는 트랜잭션을 롤백하고 다시 제출합니다.
- 또한 메시지를 *SideQueueName*에 넣습니다. *SideQueueName*을 사용하여 모든 메시지가 커밋되는지 또는 *QueueName*에서 성공적으로 롤백되는지 확인합니다. 불일치를 감지하는 경우 오류 메시지를 작성합니다.
- 마지막 매개변수를 (0|1|2)로 설정하여 **amqsfhac**에서 추적하는 출력의 양을 변경합니다.

0

최소 출력.

1

중간 출력.

2

최대 출력.

클라이언트 연결 구성

샘플을 실행하려면 클라이언트 및 서버 연결 채널을 구성해야 합니다. 클라이언트 확인 프로시저는 클라이언트 테스트 환경 설정 방법을 설명합니다.

또는 다음 예에 제공된 구성을 사용하십시오.

amqsghac, amqsphac 및 amqsmhac의 사용 예

이 예는 단일 인스턴스 큐 관리자를 사용하는 다시 연결 가능한 클라이언트를 보여줍니다.

메시지는 **amqsphac**에 의해 SOURCE 큐에 위치되고, **amqsmhac**에 의해 TARGET에 전송되며, **amqsghac**에 의해 TARGET에서 검색됩니다. [1004 페이지의 그림 145](#)의 내용을 참조하십시오.

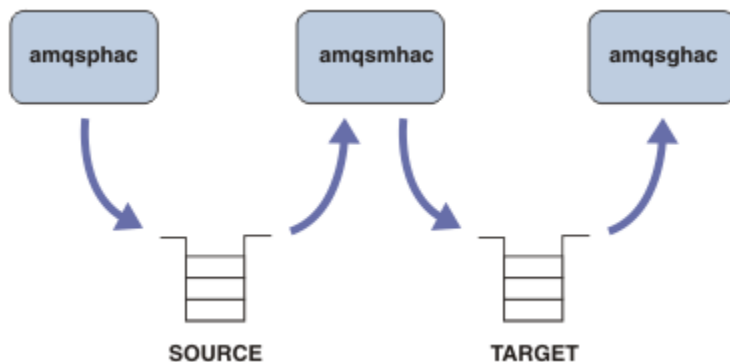


그림 145. 다시 연결 가능한 클라이언트 샘플

샘플을 실행하려면 다음 단계를 따르십시오.

1. 다음 명령을 포함한 `hasamples.tst` 파일을 작성하십시오.

```

DEFINE QLOCAL(SOURCE) REPLACE
DEFINE QLOCAL(TARGET) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
MCAUSER(MUSR_MQADMIN) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNNAME('LOCALHOST(2345)') QMNAME(QM1) REPLACE
ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) +
PORT(2345)
START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
START CHANNEL(CHANNEL1)
  
```

2. 명령 프롬프트에서 다음 명령을 입력하십시오.

- a. crtmmq QM1
- b. strmmq QM1
- c. runmqsc QM1 < hasamples.tst

3. 환경 변수 **MQCHLLIB**을(를) AMQCLCHL.TAB 클라이언트 채널 정의 파일의 경로로 설정하십시오(예: SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\mqgrs\QM1\@ipcc).

4. **MQCHLLIB**이 설정된 세 개의 새 창을 여십시오. 예를 들어, Windows의 경우 이전 명령 프롬프트에 **start**를 세 번 입력하여 각 창에서 각 프로그램을 시작하십시오. [1005 페이지의 『큐 관리자 설정 및 제어』의 1006 페이지의 『5』 단계를 참조하십시오.](#)

5. endmqm -r -p QM1 명령을 입력하여 큐 관리자를 중지한 다음 클라이언트가 다시 연결할 수 있도록 하십시오.

6. strmqm QM1 명령을 입력하여 큐 관리자를 재시작하십시오.

amqsgshac, **amqspshac** 및 **amqsmhac** 샘플을 Windows에서 실행한 결과가 다음 예에 표시됩니다.

큐 관리자 설정 및 제어

1. 큐 관리자를 작성하십시오.

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

나중에 **MQCHLLIB** 변수를 설정하려면 데이터 디렉토리를 기억하십시오.

2. 큐 관리자를 시작하십시오.

```
C:\> strmmq QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

3. 큐 및 채널을 작성하고 리스너 포트를 수정한 후 리스너 및 채널을 시작하십시오.

```
C:\> runmqsc QM1 < hasamples.tst

5724-H72 (C) Copyright IBM Corp. 1994, 2026. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.

      1 : DEFINE QLOCAL(SOURCE) REPLACE
AMQ8006: IBM MQ queue created.
      2 : DEFINE QLOCAL(TARGET) REPLACE
AMQ8006: IBM MQ queue created.
      3 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(MUSR_MQADMIN)
REPLACE
AMQ8014: IBM MQ channel created.
      4 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN) TRPTYPE(TCP) CONNAME('LOCALHOST(2345)')
QMNAME(QM1) REPLACE
AMQ8014: IBM MQ channel created.
      5 : ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) PORT(2345)
AMQ8623: IBM MQ listener changed.
      6 : START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
AMQ8021: Request to start IBM MQ Listener accepted.
      7 : START CHANNEL(CHANNEL1)
AMQ8018: Start IBM MQ channel accepted.
      7 MQSC commands read.
```

```
No commands have a syntax error.  
All valid MQSC commands were processed.
```

4. 클라이언트에 인식된 클라이언트 채널 테이블을 작성하십시오.

1005 페이지의 『1』 단계의 **crtmqm** 명령에서 리턴된 데이터 디렉토리를 사용하고 이것에 @ipcc 디렉토리를 추가하여 **MQCHLLIB** 변수를 설정하십시오.

```
C:\> SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc
```

5. 다른 창에서 샘플 프로그램을 시작하십시오.

```
C:\> start amqspvac SOURCE QM1  
C:\> start amqsmvac -s SOURCE -t TARGET -m QM1  
C:\> start amqsgvac TARGET QM1
```

6. 큐 관리자를 종료하고 다시 재시작하십시오.

```
C:\> endmqm -r -p QM1  
  
Waiting for queue manager 'QM1' to end.  
IBM MQ queue manager 'QM1' ending.  
IBM MQ queue manager 'QM1' ended.  
  
C:\> strmqm QM1  
  
IBM MQ queue manager 'QM1' starting.  
5 log records accessed on queue manager 'QM1' during the log replay phase.  
Log replay for queue manager 'QM1' complete.  
Transaction manager state recovered for queue manager 'QM1'.  
IBM MQ queue manager 'QM1' started.
```

amqspvac

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

amqsmvac

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

amqsgvac

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


message Message 4
message Message 5

관련 태스크

공유 파일 시스템 작동 확인

관련 참조

[amqmfscck](#)(파일 시스템 검사)

조회 샘플 프로그램

조회 샘플 프로그램은 MQINQ 호출을 사용하여 큐의 일부 속성에 대해 조회합니다.

다음 플랫폼별 프로그램 명칭은 해당 항목을 참조하십시오:

- ▶ **Linux** ▶ **AIX** [C 및 COBOL MQI 샘플에 표시된 기능 AIX and Linux](#)
- ▶ **Windows** [C 및 COBOL MQI 샘플에 표시된 기능 Windows](#)
- ▶ **IBM i** [C, COBOL 및 RPG MQI 샘플에 표시된 기능 IBM i](#)

이러한 프로그램은 트리거링된 프로그램으로서 실행하기 위한 것이므로 이에 대한 유일한 입력은 IBM MQ for Multiplatforms의 MQTMC2(트리거 메시지) 구조입니다. 이 구조에는 조회될 속성이 있는 대상 큐의 이름이 포함되어 있습니다. C 버전도 큐 관리자 이름을 사용합니다. COBOL 버전은 기본 큐 관리자를 사용합니다.

트리거링 프로세스가 작업하기 위해, 사용하려는 조회 샘플 프로그램이 SYSTEM.SAMPLE.INQ 큐에 도착하는 메시지에 의해 트리거되는지 확인하십시오. 이를 수행하려면, 프로세스 정의 SYSTEM.SAMPLE.INQPROCESS의 *ApplicId* 필드에 사용하려는 조회 샘플 프로그램의 이름을 지정하십시오. ▶ **IBM i** IBM i의 경우 CHGMQMPRC 명령을 사용할 수 있습니다. 세부사항은 [MQ 프로세스 변경\(CHGMQMPRC\)](#)을 참조하십시오. 샘플 큐에는 FIRST 트리거 유형이 있습니다. 이미 큐에 메시지가 있는 경우 요청 샘플을 실행하기 전에 조회 샘플이 송신된 메시지에 의해 트리거되지 않습니다.

올바르게 정의가 설정된 경우

- ▶ **ALW** AIX, Linux, and Windows의 경우 하나의 세션에서는 **runmqtrm** 프로그램을 시작하고 다른 세션에서는 **amqsreq** 프로그램을 시작하십시오.
- ▶ **IBM i** IBM i의 경우 하나의 세션에서 AMQSERV4 프로그램을 시작한 후 다른 세션에서 AMQSREQ4 프로그램을 시작하십시오. AMQSERV4 대신에 AMQSTRG4를 사용할 수 있지만, 잠재적인 작업 제출 지연으로 인해 발생하는 상황에 쉽게 대응하지 못할 수 있습니다.

요청 샘플 프로그램을 사용하여 각각 큐 이름만 포함하는 요청 메시지를 SYSTEM.SAMPLE.INQ 큐에 송신합니다. 각 요청 메시지의 경우 조회 샘플 프로그램은 요청 메시지에 지정된 큐에 대한 정보가 포함되어 있는 응답 메시지를 송신합니다. 응답은 요청 메시지에 지정된 응답 대상 큐로 송신됩니다.

▶ **IBM i** IBM i에서 샘플 입력 파일 멤버 QMQMSAMP.AMQSDATA(INQ)가 사용되며 이름이 지정된 마지막 큐가 존재하지 않으므로 샘플은 실패에 대한 이유 코드와 함께 보고 메시지를 리턴합니다.

조회 샘플 프로그램의 디자인

프로그램은 시작했을 때 전달된 트리거 메시지 구조에서 이름 지정된 큐를 엽니다. (명확성을 위해 이를 요청 큐라고 합니다.) 프로그램은 MQOPEN 호출을 사용하여 공유 입력에 대한 이 큐를 엽니다.

프로그램은 MQGET 호출을 사용하여 이 큐에서 메시지를 제거합니다. 이 호출은 5초의 대기 간격으로 MQGMO_ACCEPT_TRUNCATED_MSG 및 MQGMO_WAIT 옵션을 사용합니다. 프로그램은 요청 메시지인지 확인하기 위해 각 메시지의 디스크립터를 테스트합니다. 요청 메시지가 아닌 경우 프로그램은 메시지를 제거하고 경고 메시지를 표시합니다.

요청 큐에서 제거된 각 요청 메시지의 경우 프로그램은 데이터에 포함된 큐의 이름을 읽고(대상 큐라고 함) MQOO_INQ 옵션이 포함된 MQOPEN 호출을 사용하여 해당 큐를 엽니다. 그런 다음 프로그램은 MQINQ 호출을 사용하여 대상 큐의 *InhibitGet*, *CurrentQDepth* 및 *OpenInputCount* 속성 값을 조회합니다.

MQINQ 호출에 성공하는 경우 프로그램은 MQPUT1 호출을 사용하여 응답 메시지를 응답 대상 큐에 넣습니다. 이 메시지는 세 가지 속성 값을 포함합니다.

MQOPEN 또는 MQINQ 호출이 성공하지 못한 경우 프로그램은 MQPUT1 호출을 사용하여 보고 메시지를 응답 대상 큐에 넣습니다. 이 보고 메시지에 있는 메시지 디스크립터의 *Feedback* 필드는 어떤 호출이 실패했는지에 따라 MQOPEN 또는 MQINQ 호출에서 리턴된 이유 코드입니다.

MQINQ 호출 후 프로그램은 MQCLOSE 호출을 사용하여 대상 큐를 닫습니다.

요청 큐에 남아 있는 메시지가 없는 경우 프로그램은 해당 큐를 닫고 큐 관리자의 연결을 끊습니다.

관련 참조

[1072 페이지의 『Inquire RPG 샘플 프로그램 IBM i』](#)

조회 샘플 프로그램, AMQ3INQ4는 MQINQ 호출을 사용하여 큐의 몇 가지 속성을 조회합니다.

메시지 핸들 샘플 프로그램의 조회 특성

AMQSIQMA는 메시지 큐에서 메시지 핸들의 특성을 조회하는 샘플 C 프로그램이며, MQINQMP API 호출을 사용하는 예입니다.

이 샘플은 메시지 핸들을 작성하여 MQGMO 구조의 *MsgHandle* 필드에 넣습니다. 그런 다음 샘플은 하나의 메시지를 가져와서 메시지 핸들이 채워지는 모든 특성을 조회하고 출력합니다.

```
C:\Program Files\IBM\MQ\tools\c\Samples\Bin >amqsiqm Q QM1
Sample AMQSIQMA start
property name MyProp value MyValue
message text Hello world!
Sample AMQSIQMA end
```

발행/구독 샘플 프로그램

발행/구독 샘플 프로그램은 IBM MQ의 발행 및 구독 기능의 사용을 보여줍니다.

IBM MQ 발행/구독 인터페이스에 프로그래밍하는 방법을 설명하는 세 가지 C 언어 샘플 프로그램이 있습니다. 이전 인터페이스를 사용하는 일부 C 샘플과 Java 샘플이 있습니다. Java 샘플은 com.ibm.mq.jar의 IBM MQ 발행/구독 인터페이스와 com.ibm.mqjms의 JMS 발행/구독 인터페이스를 사용합니다. JMS 샘플은 이 주제에서 다루지 않습니다.

C

C 샘플 폴더에서 발행자 샘플 amqspub을(를) 찾으십시오. 첫 번째 매개변수로 원하는 토픽 이름과 그 뒤에 선택적 큐 관리자 이름을 사용하여 샘플을 실행하십시오. 예를 들어, amqspub mytopic QM3.amqsubc(이)라는 클라이언트 버전도 있습니다. 클라이언트 버전을 실행하려는 경우 먼저 [979 페이지의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』](#)의 세부사항을 참조하십시오.

발행자는 기본 큐 관리자에 연결되고 target topic is mytopic 출력으로 응답합니다. 지금부터 이 창에 입력하는 모든 행은 mytopic에 발행됩니다.

동일한 디렉토리에서 다른 명령 창을 열고 동일한 토픽 이름 및 선택적 큐 관리자 이름을 제공하여 구독자 프로그램, amqssub을(를) 실행하십시오. 예: amqssub mytopic QM3.

구독자는 Calling MQGET : 30 seconds wait time 출력으로 응답합니다. 이제부터 발행자에 입력하는 행은 구독자의 출력에 표시됩니다.

다른 명령 창에서 다른 구독자를 시작하고 두 구독자가 발행물을 수신하는지 확인하십시오.

옵션 설정을 포함하여 매개변수에 대한 전체 문서는 샘플 소스 코드를 참조하십시오. 구독자 옵션 필드에 대한 값은 다음 주제 [옵션\(MQLONG\)](#)에서 설명됩니다.

추가 구독 옵션을 명령행 스위치로 제공하는 다른 구독자 샘플 amqssbx이(가) 있습니다.

구독자가 종료된 후 보유되는 지속 가능 구독을 사용하여 구독자를 호출하려면 amqssbx -d mysub -t mytopic -k를 입력하십시오.

발행자로 다른 항목을 발행하여 구독을 테스트하십시오. 구독자가 종료할 때까지 30초 동안 대기하십시오. 동일한 주제의 더 많은 항목을 발행하십시오. 구독자를 재시작하십시오. 구독자가 재시작되면 구독자는 실행되지 않는 동안 발행된 마지막 항목을 즉시 표시합니다.

C 레거시

큐에 있는 명령을 보여주는 추가 C 샘플 세트가 있습니다. 이들 샘플의 일부는 원래 MQOC Supportpac의 일부로 제공되었습니다. 호환성을 위해 샘플이 표시하는 기능을 전체 지원합니다.

큐된 명령 인터페이스를 사용하지 않는 것이 좋습니다. 발행/구독 API보다 훨씬 더 복잡합니다. 그리고 복잡한 큐 대기 명령을 프로그래밍할 설득력 있는 기능적 이유가 없습니다. 그러나 이미 인터페이스를 사용하고 있기 때문에 또는 프로그래밍 환경이 MQSUB에 대해 다른 호출을 구성하는 대신 일반 MQPUT를 호출하고 복잡한 메시지를 빌드하기 쉽게 만들기 때문에 큐 대기 방법이 더 적당하다는 것을 발견할 수 있습니다.

추가 샘플은 samples 폴더의 pubsub 서브디렉토리에 있습니다.

1009 페이지의 표 167에 나열된 샘플 유형에는 6가지가 있습니다.

표 167. 레거시 발행/구독 샘플 C 프로그램의 범주		
범주	프로그램	주석
RFH1	amqssr1a.c amqspr1a.c	RFH1 형식 메시지를 사용하여 빌드된 단순 발행/구독 예
RFH2	amqssr2a.c amqspr2a.c	RFH2 형식 메시지를 사용하여 빌드된 단순 발행/구독 예
MQAI 샘플	amqsppca.c amqsspca.c	PCF 명령 및 MQAI 명령행 인터페이스를 사용하여 빌드된 단순 발행/구독 예
RFH1을 사용하는 MAOC 결과 서비스	amqsgama.c amqsresa.c	RFH1 헤더를 사용하여 빌드된 결과 서비스 1. amqsgama.tst 및 amqsresa.tst에 정의된 큐가 필요합니다. 2. amqsresa은(는) amqsgama 이전에 시작되어야 합니다.
RFH2를 사용하는 MAOC 결과 서비스	amqsgr2a.c amqsrr2a.c	RFH2 헤더를 사용하여 빌드된 결과 서비스 1. amqsgama.tst 및 amqsresa.tst에 정의된 큐가 필요합니다. 2. amqsresa은(는) amqsgama 이전에 시작되어야 합니다.
라우팅 엑시트 발행/구독 샘플	amqspdra.c	라우팅 엑시트의 발행/구독 메시지에 대한 큐 또는 큐 관리자 목적지를 변경하는 방법을 보여줍니다.

Java용 샘플 프로그램

Java 샘플 MQPubSubApiSample.java은(는) 단일 프로그램에 발행자 및 구독자를 결합합니다. 그 소스 및 결합된 클래스 파일은 wmqjava 샘플 폴더에 있습니다.

클라이언트 모드로 실행하려는 경우 먼저 979 페이지의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』의 세부사항을 참조하십시오.

Java 환경이 구성된 경우 Java 명령을 사용하여 명령행에서 샘플을 실행하십시오. 또한 탐색기 EclipseIBM MQ 작업 공간에서 샘플을 실행할 수 있으며, 해당 작업 공간에는 이미 프로그래밍 Java 워크벤치가 설정되어 있습니다.

샘플 프로그램을 실행하려면 샘플 프로그램의 특성 중 일부를 변경해야 하는 경우가 있습니다. JVM에 매개변수를 제공하거나 소스를 편집하여 이를 수행하십시오.

1010 페이지의 『MQPubSubApiSample Java 샘플 실행』의 지시사항은 Eclipse 작업공간에서 샘플을 실행하는 방법을 설명합니다.

MQPubSubApiSample Java 샘플 실행

Eclipse 플랫폼에서 JDT(Java Development Tools)를 사용하여 MQPubSubApiSample을 실행하는 방법.

시작하기 전에

Eclipse 워크벤치를 여십시오. 새 작업공간 디렉토리를 작성한 후 선택하십시오. 시작 창을 닫으십시오.

클라이언트로 실행하기 전에 979 페이지의 『멀티플랫폼에서 클라이언트 연결을 승인하도록 큐 관리자 구성』의 단계를 따르십시오.

이 태스크 정보

Java 발행/구독 샘플 프로그램은 IBM MQ MQI client Java 프로그램입니다. 샘플은 포트 1414에서 청취하는 기본 큐 관리자를 사용하여 수정 없이 실행합니다. 태스크는 이 간단한 경우에 대해 설명하고 일반 용어로 매개변수를 제공하고 다른 IBM MQ 구성에 맞게 샘플을 수정하는 방법을 표시합니다. 예는 Windows에서 실행을 나타냅니다. 파일 경로는 다른 플랫폼과 다를 수 있습니다.

프로시저

1. Java 샘플 프로그램을 가져오십시오.

- a) 워크벤치에서 **창 > 퍼스펙티브 열기 > 기타 > Java**를 클릭하고 **확인**을 클릭하십시오.
- b) **패키지 탐색기** 보기로 전환하십시오.
- c) **패키지 탐색기** 보기의 공간에서 마우스 오른쪽 단추를 클릭하십시오. **새로 작성 > Java 프로젝트**를 클릭하십시오.
- d) '**Project name**' 필드에 'MQ Java Samples'를 입력합니다. **다음**을 클릭하십시오.
- e) '**Java Settings**' 패널에서 **라이브러리** 탭으로 전환합니다.
- f) **외부 JAR** 추가를 클릭하십시오.
- g) `MQ_INSTALLATION_PATH\java\lib`(으)로 이동하고(여기서 `MQ_INSTALLATION_PATH`는(는) IBM MQ 설치 폴더임) `com.ibm.mq.jar` 및 `com.ibm.mq.jmqi.jar`을(를) 선택하십시오.
- h) **열기 > 완료**를 클릭하십시오.
- i) **패키지 탐색기** 보기에서 `src`을(를) 마우스 오른쪽 단추로 클릭하십시오.
- j) **선택 가져오기...** > **일반 > 파일 시스템 > 다음 > 찾아보기...** 를 클릭하고 '`MQ_INSTALLATION_PATH\tools\wmqjava\samples`' 경로를 찾습니다. 여기서 '`MQ_INSTALLATION_PATH`' IBM MQ '설치 디렉터리'입니다.
- k) **가져오기** 패널에서, 1011 페이지의 그림 146, `samples`을(를) 클릭하십시오(선택란을 선택하지 않음).
- l) `MQPubSubApiSample.java`을(를) 선택하십시오. '**Into folder**' 필드에는 'MQ Java Samples/src'가 포함되어야 합니다. **완료**를 클릭하십시오.

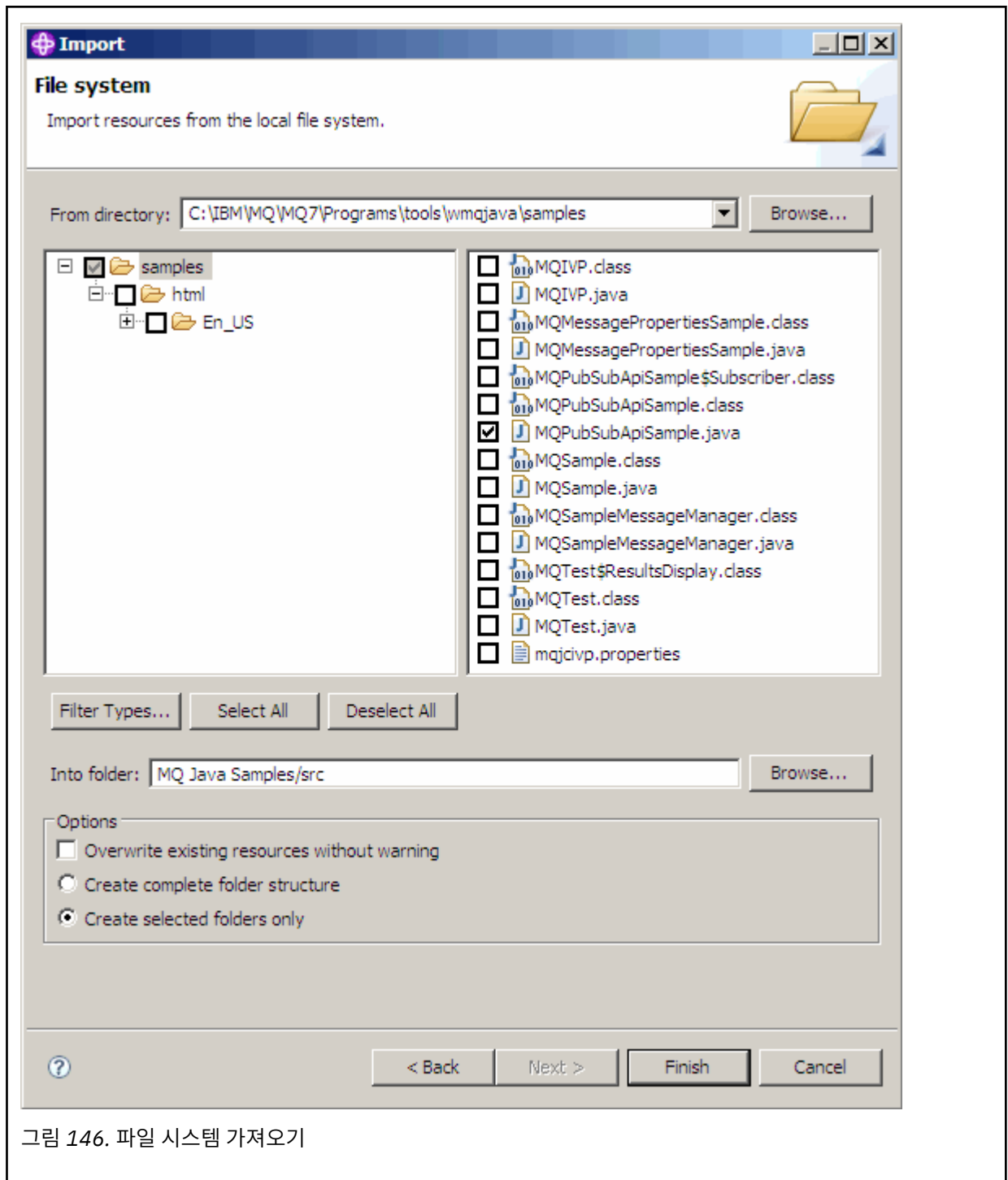


그림 146. 파일 시스템 가져오기

2. 발행/구독 샘플 프로그램을 실행하십시오.

기본 매개변수를 변경해야 하는지에 따라 프로그램을 실행하는 두 가지 방법이 있습니다.

- 첫 번째 선택은 변경하지 않고 프로그램을 실행합니다.
 - 작업공간 기본 메뉴에서 src 폴더를 펼치십시오. 마우스 오른쪽 버튼으로 **MQPubSubApiSample.java Run-as > 1. Java 애플리케이션**
- 두 번째 선택은 매개변수를 사용하여 또는 환경에 대해 수정된 소스 코드를 사용하여 프로그램을 실행합니다.
 - MQPubSubApiSample.java(를) 열고 MQPubSubApiSample 생성자를 검토하십시오.
 - 프로그램의 속성을 수정하십시오.

이러한 속성은 -D JVM 스위치를 사용하거나, 소스 코드를 편집하여 System property에 대한 기본값을 제공함으로써 변경할 수 있습니다.

- topicObject
- queueManagerName
- subscriberCount

이러한 속성은 구성자에서 소스 코드를 편집하는 방법으로만 변경할 수 있습니다.

- 호스트 이름
- 포트
- 채널

System properties를 설정하려면, 액세서에서 기본값을 코딩하십시오. 예를 들어,

```
queueManagerName = System.getProperty("com.ibm.mq.pubSubSample.queueManagerName",  
"QM3");
```

또는 다음 단계에 표시된 대로 -D 옵션을 사용하여 JVM에 매개변수를 제공하십시오.

- 설정하려는 System.Property의 전체 이름을 복사하십시오. 예:
com.ibm.mq.pubSubSample.queueManagerName.
- 작업공간에서 **실행 > 열린 실행 대화 상자**를 마우스 오른쪽 단추로 클릭하십시오. **애플리케이션 만들기, 관리 및 실행에서** 'Java 애플리케이션을 두 번 클릭하고 **(x) = 인수** 탭을 클릭합니다.
- VM 인수:** 분할창에 -D을 입력하고 =QM3이 뒤에 오는 System.property 이름,
com.ibm.mq.pubSubSample.queueManagerName을 붙여넣으십시오. **적용 > 실행**을 누르십시오.
- 선택으로 구분된 목록으로 또는 분할창의 추가 행으로 선택 없이 추가 인수를 추가하십시오.
예: -Dcom.ibm.mq.pubSubSample.queueManagerName=QM3,
-Dcom.ibm.mq.pubSubSample.subscriberCount=6.

발행 엑시트 샘플 프로그램

AMQSPSE0는 구독자에게 전달되기 전에 발행물을 가로채기 위한 엑시트의 샘플 C 프로그램입니다. 그런 다음 엑시트는 예를 들어 메시지 헤더, 페이로드 또는 목적지를 변경하거나 메시지가 구독자에게 발행되지 않도록 방지할 수 있습니다.

샘플을 실행하려면 다음 작업을 수행하십시오.

1. 큐 관리자를 구성하십시오.

-   AIX and Linux 시스템에서 다음과 같이 스탠자를 qm.ini 파일에 추가하십시오.

```
PublishSubscribe:  
PublishExitPath=Module  
PublishExitFunction=EntryPoint
```

모듈이 있는 곳은 `MQ_INSTALLATION_PATH/samp/bin/amqspse`. `MQ_INSTALLATION_PATH`는 IBM MQ가 설치되어 있는 상위 레벨 디렉토리를 나타냅니다.

-  Windows에서 레지스트리에 동일한 속성을 설정하십시오.
2. 모듈이 IBM MQ에 액세스할 수 있는지 확인하십시오.
 3. 큐 관리자를 재시작하여 구성을 선택하십시오.
 4. 추적할 애플리케이션 프로세스에서, 추적 파일이 작성되어야 하는 위치를 설명하십시오. 예를 들면, 다음과 같습니다.

- ▶ **Linux** ▶ **AIX** 시스템에서 AIX and Linux 디렉터리가 /var/mqm/trace 존재하는지 확인하고 환경 **MQPSE_TRACE_LOGFILE** 변수를 내보내십시오:

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

- ▶ **Windows** 날짜에 디렉터리가 C:\temp 존재하는지 확인하고 Windows 환경 변수 **MQPSE_TRACE_LOGFILE** 를 설정하십시오:

```
set MQPSE_TRACE_LOGFILE=C:\temp\PubTrace
```

Put 샘플 프로그램

Put 샘플 프로그램은 MQPUT 호출을 사용하여 큐에 메시지를 넣습니다.

다음 플랫폼별 프로그램 명칭은 해당 항목을 참조하십시오:

- ▶ **Linux** ▶ **AIX** C 및 COBOL MQI 샘플에 표시된 기능 AIX and Linux
- ▶ **Windows** C 및 COBOL MQI 샘플에 표시된 기능 Windows
- ▶ **IBM i** C, COBOL 및 RPG MQI 샘플에 표시된 기능 IBM i

Put 샘플 프로그램의 디자인

프로그램은 메시지를 넣기 위해 대상 큐를 여는 데 MQOO_OUTPUT 옵션과 함께 MQOPEN 호출을 사용합니다.

큐를 열 수 없는 경우 프로그램은 MQOPEN 호출에서 리턴한 이유 코드가 포함된 오류 메시지를 출력합니다. 프로그램을 단순하게 유지하기 위해 이 호출 및 후속 MQI 호출에서 프로그램은 여러 옵션의 기본값을 사용합니다.

입력의 각 행에 대해 프로그램은 텍스트를 버퍼로 해석하고 MQPUT 호출을 사용하여 해당 행의 텍스트를 포함하는 데이터그램 메시지를 작성합니다. 프로그램은 입력의 끝에 도달하거나 MQPUT 호출에 실패할 때까지 계속됩니다. 프로그램이 입력의 끝에 도달하는 경우 MQCLOSE 호출을 사용하여 큐를 닫습니다.

관련 참조

1073 페이지의 『Put RPG 샘플 프로그램 IBM i』

Put 샘플 프로그램, AMQ3PUT4는 MQPUT 호출을 사용하여 메시지를 큐에 넣습니다.

Put 샘플 프로그램 실행

amqsput 및 amqsputc 샘플 실행

▶ **ALW**

샘플은 **amqsput** 로컬 바인딩을 사용하여 메시지를 전송하는 프로그램이며, **amqsputc** 샘플은 클라이언트 바인딩을 사용하여 메시지를 전송하는 프로그램입니다. 이러한 프로그램 각각 다음 위치 매개변수를 사용합니다.

1. 대상 큐의 이름(필수)
2. 큐 관리자의 이름(선택사항)

큐 관리자가 지정되지 않은 경우, 기본 큐 **amqsput** 관리자에 연결하며 **MQSERVER** 환경 변수 또는 클라이언트 **amqsputc** 채널 정의 파일로 식별된 큐 관리자에 연결합니다.

3. 열기 옵션(선택사항)

열기 옵션이 지정되지 않은 경우 샘플은 이러한 두 가지 옵션의 결합인 8208 값을 사용합니다.

- MQOO_OUTPUT
- MQOO_FAIL_IF QUIESCING

4. 닫기 옵션(선택사항)

닫기 옵션이 지정되지 않은 경우 샘플은 값이 0인 MQCO_NONE을 사용합니다.

5. 대상 큐 관리자의 이름(선택사항)

대상 큐 관리자가 지정되지 않은 경우 MQOD의 ObjectQMgrName 필드는 공백으로 남겨 둡니다.

6. 동적 큐의 이름(선택사항)

동적 큐 이름이 지정되지 않은 경우 MQOD의 DynamicQName 필드는 공백으로 남겨 둡니다.

다음 환경 변수를 사용하여 큐 관리자와의 인증에 사용되는 자격 증명을 제공하십시오:

MQSAMP_사용자_ID

큐 매니저에 사용자 ID와 비밀번호를 사용하여 인증하려면 연결 인증에 사용할 사용자 ID를 설정하십시오. 프로그램은 사용자 ID에 해당하는 비밀번호를 입력하도록 요청합니다.

Linux AIX MQSAMP_TOKEN

큐 관리자와 인증하기 위해 인증 토큰을 제공하려면 비어 있지 않은 값으로 설정하십시오. 프로그램이 인증 토큰을 요청합니다. 인증 토큰은 클라이언트 바인딩을 사용하는 **amqspu**tc 샘플에서만 사용할 수 있습니다.

이러한 프로그램을 실행하려면 다음 명령 중 하나를 입력하십시오.

- `amqspu myqueue qmanagername`
- `amqspu myqueue qmanagername`

여기서 *myqueue*는 메시지가 들어가는 큐의 이름이며 *qmanagername*은 *myqueue*를 소유하는 큐 관리자입니다.

amq0put 샘플 실행

ALW

COBOL 버전에는 매개변수가 없습니다. 기본 큐 관리자에 연결하고 실행하는 경우 다음 프롬프트가 표시됩니다.

Please enter the name of the target queue

StdIn에서 입력을 가져와서 입력의 각 행을 대상 큐에 추가합니다. 빈 줄은 더 이상 데이터가 없다는 것을 표시합니다.

AMQSPUT4 C 샘플 실행(IBM i)

IBM i

IBM i 플랫폼에서만 사용 가능한 C 프로그램 AMQSPUT4는 소스 파일의 멤버에서 데이터를 읽어 메시지를 작성합니다.

프로그램을 시작할 때 매개변수로 파일 이름을 지정해야 합니다. 파일의 구조는 다음과 같아야 합니다.

```
queue name
text of message 1
text of message 2
:
text of message n
blank line
```

Put 샘플의 입력 샘플은 QMQMSAMP 라이브러리 AMQSDATA 파일 PUT 멤버에 제공됩니다.

참고: 큐 이름은 대소문자를 구분합니다. 샘플 파일 생성 프로그램이 생성한 모든 대기열의 AMQSAMP4 이름은 대문자로 생성됩니다.

C 프로그램은 파일의 첫 번째 행에서 이름 지정된 큐에 메시지를 넣습니다. 제공된 SYSTEM.SAMPLE.LOCAL 큐를 사용할 수 있습니다. 프로그램은 파일의 다음 각 행의 텍스트를 별도의 데이터 그램 메시지에 넣고 파일의 끝에서 빈 줄을 읽게 되면 중지합니다.

예제 데이터 파일을 사용하려면 명령은 다음과 같습니다.

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

AMQ0PUT4 COBOL 샘플 실행(IBM i)

IBM i

IBM i 플랫폼에서만 사용 가능한 COBOL 프로그램 AMQ0PUT4는 키보드에서 데이터를 승인하여 메시지를 작성합니다.

프로그램을 시작하려면, 프로그램을 호출하고 프로그램 매개변수로 대상 큐의 이름을 지정하십시오. 프로그램은 키보드의 입력을 버퍼에 전달되도록 승인하고 각 텍스트 행에 대한 데이터그램 메시지를 작성합니다. 프로그램은 키보드에 공백 행을 입력하는 경우 중지됩니다.

참조 메시지 샘플 프로그램

참조 메시지 샘플을 사용하면 소스 또는 대상 노드의 IBM MQ 큐에 오브젝트를 저장할 필요 없이 한 노드에서 다른 노드(일반적으로 다른 시스템에 있음)로 대형 오브젝트를 전송할 수 있습니다.

메시지 엑시트에 의해 수신되고 큐에서 가져오는 참조 메시지를 큐에 넣을 수 있는 방법을 보여주기 위해 샘플 프로그램 세트가 제공됩니다. 샘플 프로그램은 파일을 이동하는 데 참조 메시지를 사용합니다. 데이터베이스와 같은 다른 오브젝트를 이동하거나 보안 검사를 수행하려는 경우 amqsxrm 샘플을 기반으로 고유 엑시트를 정의하십시오.

사용할 참조 메시지 엑시트 샘플 프로그램의 버전은 채널이 실행 중인 플랫폼에 따라 달라집니다.

- 모든 플랫폼의 경우 송신 끝에서 amqsxrma를 사용하십시오.
- 수신자가 IBM i를 제외한 플랫폼에서 실행 중인 경우 수신 측에서 amqsxrma를 사용하십시오.
- **IBM i** 수신자가 IBM i에서 실행 중인 경우 amqsxrm4를 사용하십시오.

IBM i

IBM i 사용자를 위한 참고

샘플 메시지 엑시트를 사용하여 참조 메시지를 수신하려면, 스트림 파일을 작성할 수 있도록 IFS의 루트 파일 시스템 또는 임의의 서브디렉토리에 있는 파일을 지정하십시오.

IBM i에서 샘플 메시지 엑시트는 파일을 작성하고, 데이터를 EBCDIC으로 변환하고, 코드 페이지를 시스템 코드 페이지로 설정합니다. 그런 다음 CPYFRMSTMF 명령을 사용하여 이 파일을 QSYS.LIB 파일 시스템에 복사할 수 있습니다. 예를 들면, 다음과 같습니다.

```
CPYFRMSTMF FROMSTMF('JANEP/TEST.TXT')
TOMBR('qsys.lib.janep.lib/test.fie/test.mbr') MBROPT(*REPLACE)
CVTDTA(*NONE)
```

CPYFRMSTMF 명령은 파일을 작성하지 않습니다. 이 명령을 실행하기 전에 작성해야 합니다.

QSYS.LIB에서 파일을 송신하는 경우 샘플을 변경하지 않아도 됩니다. 다른 파일 시스템의 경우 MQRMH 구조의 CodedCharSetId 필드에 지정된 CCSID가 송신하는 벌크 데이터와 일치하는지 확인하십시오.

통합 파일 시스템을 사용하는 경우 SYSIFCOPT(*IFSIO) 옵션을 설정하여 프로그램 모듈을 작성하십시오. 데이터베이스 또는 고정 길이 레코드 파일을 이동하려는 경우 제공된 샘플 AMQXRM4를 기반으로 고유 엑시트를 정의하십시오.

데이터베이스 파일을 전송하기 위한 방법으로는 CPYTOSTMF 명령을 사용하여 IFS 구조로 변환한 후 IFS 파일에 첨부하여 참조 메시지를 송신하는 것이 좋습니다. IFS 구조로 변환하지 않고 IFS 내에서 참조하게 하여 데이터베이스 파일을 전송하도록 선택하는 경우 멤버 이름을 지정해야 합니다. 이 방법을 선택하는 경우 데이터 무결성은 보장되지 않습니다.

Multi

참조 메시지 샘플 실행

이 예제를 사용하여 AIX, Linux, and Windows에서 참조 메시지 샘플 애플리케이션 AMQSPRMIBM i에서는 AMQSPRMA의 실행 방법을 알 수 있습니다. 이 예는 메시지 엑시트에 의해 수신되고 큐에서 가져오는 참조 메시지를 큐에 넣을 수 있는 방법을 설명합니다.

참고 메시지 샘플은 다음과 같이 실행됩니다.

MACHINE1 시스템에서, 큐 관리자 QMGR1

MQSC 구문

```
define chl(chl1) chlname(sdr) trptype(tcp) conname('machine2') xmitq(xmitq)
msgdata(FLATFILE) msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)')

define ql(xmitq) usage(xmitq)

define chl(report) chlname(rcvr) trptype(tcp) replace

define qr(qr) rname(dq) rqmname(qmgr2) xmitq(xmitq) replace
```

▶ IBM i IBM i 명령 구문

참고: 큐 관리자 이름을 지정하지 않는 경우 시스템은 기본 큐 관리자를 사용합니다.

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*SDR) MQMNAME(QMGR1) +
          REPLACE(*YES) TRPTYPE(*TCP) +
          CONNAME('MACHINE2(60501)') TMQNAME(XMITQ) +
          MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMQ QNAME(XMITQ) QTYPE(*LCL) MQMNAME(QMGR1) +
        REPLACE(*YES) USAGE(*TMQ)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*RCVR) +
          MQMNAME(QMGR1) REPLACE(*YES) TRPTYPE(*TCP)

CRTMQMQ QNAME(QR) QTYPE(*RMT) MQMNAME(QMGR1) +
        REPLACE(*YES) RMTQNAME(DQ) +
        RMTMQMNAME(QMGR2) TMQNAME(XMITQ)
```

MACHINE2 시스템에서, 큐 관리자 QMGR2

MQSC 구문

```
define chl(chl1) chlname(rcvr) trptype(tcp)
msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)')
msgdata(flatfile)

define chl(report) chlname(sdr) trptype(tcp) conname('MACHINE1')
xmitq(qmgr1)

define ql(initq)

define ql(qmgr1) usage(xmitq)

define pro(proc) applicid('/usr/lpp/mqm/samp/bin/amqsgrm')

define ql(dq) initq(initq) process(proc) trigger trigttype(first)
```

▶ IBM i IBM i 명령 구문

참고: ▶ IBM i IBM i에서 큐 관리자 이름을 지정하지 않은 경우 시스템에서는 기본 큐 관리자를 사용합니다.

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*RCVR) MQMNAME(QMGR2) +
          REPLACE(*YES) TRPTYPE(*TCP) +
          MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*SDR) MQMNAME(QMGR2) +
          REPLACE(*YES) TRPTYPE(*TCP) +
          CONNAME('MACHINE1(60500)') TMQNAME(QMGR1)

CRTMQMQ QNAME(INITQ) QTYPE(*LCL) MQMNAME(QMGR2) +
        REPLACE(*YES) USAGE(*NORMAL)

CRTMQMQ QNAME(QMGR1) QTYPE(*LCL) MQMNAME(QMGR2) +
        REPLACE(*YES) USAGE(*TMQ)

CRTMQMPRC PRCNAME(PROC) MQMNAME(QMGR2) REPLACE(*YES) +
        APPID('QMQM/AMQSGRM4')

CRTMQMQ QNAME(DQ) QTYPE(*LCL) MQMNAME(QMGR2) +
```

```
REPLACE(*YES) PRCNAME(PROC) TRGENBL(*YES) +  
INITQNAME(INITQ)
```

2. 일단 IBM MQ 오브젝트가 작성되면,

- 플랫폼에 적용 가능한 경우 큐 관리자 송신 및 수신을 위한 리스너를 시작하십시오.
- 채널 CHL1 및 REPORT를 시작하십시오.
- 수신 큐 관리자에서 이니시에이션 큐 INITQ에 대한 트리거 모니터를 시작하십시오.

3. 명령행에서 다음 매개변수를 사용하여 put 참조 메시지 샘플 프로그램 AMQSPRM(IBM i의 AMQSPRMA)을 호출하십시오.

-m

로컬 큐 관리자의 이름. 기본 큐 관리자를 기본값으로 지정합니다.

-i

소스 파일의 이름 및 위치

-o

대상 파일의 이름 및 위치

-q

큐의 이름

-g

-q 매개변수에 정의된 큐가 있는 큐 관리자의 이름입니다. -m 매개변수에 지정된 큐 관리자를 기본값으로 지정합니다.

-t

오브젝트 유형

-w

대기 간격 즉, 수신 큐 관리자에서 예외 및 COA 보고서를 대기하는 시간.

예를 들어, 이전에 정의된 오브젝트와 함께 샘플을 사용하기 위해 다음 매개변수를 사용하게 됩니다.

```
-mQMGR1 -iInput File -oOutput File -qQR -tFLATFILE -w120
```

대기 시간을 늘리면 프로그램이 메시지를 넣는 제한시간이 초과되기 전에 네트워크 전체에서 대형 파일을 송신할 수 있는 시간을 허용합니다.

```
amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
```

IBM i 사용자:  IBM i에서는 다음 단계를 완료하십시오.

a. 다음 명령을 사용하십시오.

```
CALL PGM(QMQM/AMQSPRM4) PARM('-mQMGR1' +  
'-i/refmsgs/rmsg1' +  
'-o/refmsgs/rmsgx' '-qQR' +  
'-gQMGR1' '-tFLATFILE' '-w15')
```

원래 파일 rmsg1이 IFS 디렉토리 /refmsgs에 있으며, 대상 파일을 대상 시스템의 IFS 디렉토리 /refmsgs에 rmsgx하려는 것으로 가정합니다.

- 루트 디렉토리를 사용하기 보다 CRTDIR 명령을 사용하여 고유 디렉토리를 작성하십시오.
- 데이터를 넣는 프로그램을 호출하는 경우 출력 파일 이름이 IFS 이름 지정 규칙을 반영해야 한다는 점을 기억하십시오. 예를 들어, /TEST/FILENAME은 FILENAME이라는 파일을 TEST 디렉토리에 작성합니다.

참고:

 IBM i에서, 매개변수를 지정할 때 슬래시(/) 또는 대시(-)를 사용할 수 있습니다. 예를 들면, 다음과 같습니다.

```
amqsprm /i d:\files\infile.dat /o e:\files\outfile.dat /q QR  
/m QMGR1 /w 30 /t FLATFILE
```


Linux **AIX** AIX and Linux 플랫폼의 경우 대상 파일 디렉토리를 표기하는 데 하나의 백슬래시 대신 백슬래시 두 개(\\)를 사용해야 합니다. 따라서 **amqsprpm** 명령은 다음과 같습니다.

```
amqsprpm -i /files/infile.dat -o e:\\files\\outfile.dat -q QR
-m QMGR1 -w 30 -t FLATFILE
```

put 참조 메시지 프로그램 실행은 다음을 수행합니다.

- 큐 관리자 QMGR1의 QR 큐에 참조 메시지를 넣습니다.
 - 소스 파일 및 경로는 d:\files\infile.dat이며 예제 명령이 발행된 시스템에 있습니다.
 - 큐 QR이 리모트 큐인 경우 참조 메시지는 파일 e:\files\outfile.dat로 작성되는 다른 시스템의 다른 큐 관리자에 송신됩니다. 이 파일의 콘텐츠는 소스 파일과 동일합니다.
 - amqsprpm은 대상 큐 관리자의 COA 보고서를 30초 대기합니다.
 - 오브젝트 유형이 flatfile이므로 QR 큐에서 메시지를 이동하는 데 사용된 채널은 MsgData 필드에 이를 지정해야 합니다.
4. 채널을 정의하는 경우 amqsxrm이 될 송신 및 수신 끝 둘 다에서 메시지 엑시트를 선택하십시오.

Windows Windows에서는 다음과 같이 정의됩니다.

```
msgexit(' pathname\amqsxrm.dll(MsgExit)')
```

Linux **AIX** AIX and Linux에서는 다음과 같이 정의됩니다.

```
msgexit(' pathname/amqsxrm(MsgExit)')
```

경로 이름을 지정하는 경우 전체 이름을 지정하십시오. 경로 이름을 생략하면 qm.ini 파일에 지정된 경로(또는 IBM MQ for Windows의 경우 레지스트리에 지정된 경로)에 프로그램이 있다고 가정합니다.

5. 채널 엑시트는 참조 메시지 헤더를 읽고 헤더가 나타내는 파일을 찾습니다.
6. 그런 다음 채널 엑시트는 헤더와 함께 채널에 송신하기 전에 파일을 세그먼트화할 수 있습니다.

Linux **AIX** AIX and Linux에서는 샘플 메시지 엑시트가 해당 디렉토리에서 파일을 작성할 수 있도록 대상 디렉토리의 그룹 소유권을 'mqm'으로 변경하십시오. 또한 mqm 그룹 멤버가 쓸 수 있도록 대상 디렉토리의 권한을 변경하십시오. 파일 데이터는 IBM MQ큐에 저장되지 않습니다.

7. 파일의 마지막 세그먼트가 수신 메시지 엑시트에 의해 처리되는 경우 amqsprpm에서 지정한 대상 큐에 참조 메시지가 놓여집니다. 이 큐가 트리거되는 경우(즉, 정의가 **Trigger**, **InitQ** 및 **Process** 큐 속성을 지정하는 경우) 대상 큐의 RPOC 매개변수가 지정한 프로그램이 트리거됩니다. 트리거할 프로그램은 '**Process**' 속성의 'AppId' 필드에 정의되어 있어야 합니다.
8. 참조 메시지가 대상 큐(DQ)에 도달하는 경우 COA 보고서가 넣기 애플리케이션(amqsprpm)에 다시 송신됩니다.
9. Get 참조 메시지 샘플 amqsgrm은 입력 트리거 메시지에 지정된 큐에서 메시지를 가져오고 파일이 있는지 확인합니다.

Put 참조 메시지 샘플(amqsprma.c, AMQSPRM4) 디자인

이 주제는 Put 참조 메시지 샘플에 대한 자세한 설명을 제공합니다.

이 샘플은 파일을 참조하는 참조 메시지를 작성하고 지정된 큐에 메시지를 넣습니다.

1. 샘플은 MQCONN을 사용하여 로컬 큐 관리자에 연결합니다.
2. 그런 다음 보고 메시지를 수신하는 데 사용되는 모델 큐를 엽니다(MQOPEN).
3. 샘플은 예를 들어, 소스 및 대상 파일 이름과 오브젝트 유형과 같이 파일을 이동하는 데 필요한 값을 포함하는 참조 메시지를 빌드합니다. 예로서 IBM MQ와 함께 제공되는 샘플은 참조 메시지를 빌드하여 d:\x\file.in 파일을 QMGR1에서 QMGR2로 보내고 다음 매개변수를 사용하여 d:\y\file.out으로 파일을 재작성합니다.

```
amqsprpm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
```

여기서 QR은 QMGR2의 대상 큐를 참조하는 리모트 큐 정의입니다.

참고: AIX and Linux 플랫폼의 경우 대상 파일 디렉토리를 표기하는 데 하나의 백슬래시 대신 백슬래시 두 개 (\\)를 사용하십시오. 따라서 **amqsprpm** 명령은 다음과 같습니다.

```
amqsprpm -q QR -m QMGR1 -i /x/file.in -o d:\\y\\file.out -t FLATFILE
```

4. /q 매개변수에서 지정한 큐에 (파일 데이터 없이) 참조 메시지를 넣습니다. 리모트 큐인 경우 해당하는 전송 큐에 메시지를 넣습니다.
5. (기본값이 15초인) /w 매개변수에 지정된 시간 동안 샘플은 예외 보고서와 함께 COA 보고서를 대기하며 로컬 큐 관리자(QMGR1)에 작성된 동적 큐로 다시 송신됩니다.

참조 메시지 엑시트 샘플(amqsxrma.c, AMQSXRM4) 디자인

이 샘플은 채널 정의의 메시지 엑시트 사용자 데이터 필드에 있는 오브젝트 유형과 일치하는 오브젝트 유형의 참조 메시지를 인식합니다.

이러한 메시지의 경우 다음 상황이 발생합니다.

- 송신자 또는 서버 채널에서 지정된 길이의 데이터가 지정된 파일의 지정된 오프셋에서 에이전트 버퍼에 남아 있는 공간에 참조 메시지 뒤로 복사됩니다. 파일 끝에 도달하지 않으면 *DataLogicalOffset* 필드를 업데이트한 후 참조 메시지를 전송 큐에 다시 둡니다.
- 요청자 또는 수신자 채널에서 *DataLogicalOffset* 필드가 0이고 지정된 파일이 없으면 작성됩니다. 참조 메시지 뒤에 오는 데이터는 지정된 파일의 끝에 추가됩니다. 참조 메시지가 지정된 파일에 대한 마지막 메시지가 아닌 경우 제거됩니다. 그렇지 않으면 데이터 추가 없이 대상 큐에 넣을 채널 엑시트에 리턴됩니다.

송신자 및 서버 채널의 경우, 입력 참조 메시지의 *DataLogicalLength* 필드가 0이면 파일의 나머지 부분 (*DataLogicalOffset*에서 파일의 끝까지)이 채널을 따라 전송됩니다. 0이 아닌 경우에는 지정된 길이만 송신됩니다.

오류가 발생하는 경우(예를 들어, 샘플이 파일을 열 수 없는 경우) MQCXP입니다. *ExitResponse*은(는) 처리되는 메시지를 계속 목적지 큐에 두지 않고 데드-레터 큐에 두도록 MQXCC_SUPPRESS_FUNCTION으로 설정됩니다. 피드백 코드가 MQCXP에 리턴됩니다. *Feedback*은(는) 보고서 메시지의 메시지 디스크립터의 *Feedback* 필드에 메시지를 넣는 애플리케이션으로 리턴됩니다. MQMD의 *Report* 필드에 MQRO_EXCEPTION을 설정하여 넣기 애플리케이션이 예외 보고서를 요청했기 때문입니다.

참조 메시지의 *CodedCharacterSetId*(CCSID) 또는 인코딩이 큐 관리자와 다르면 참조 메시지가 로컬 인코딩 및 CCSID로 변환됩니다. 샘플 amqsprpm에서 오브젝트의 형식이 MQFMT_STRING이므로 데이터가 파일에 작성되기 전에 수신 끝 부분에서 amqsxrma이 오브젝트 데이터를 로컬 CCSID로 변환합니다.

파일에 멀티바이트 문자(예: DBCS 또는 유니코드)가 포함되어 있는 경우 전송되는 파일의 형식을 MQFMT_STRING으로 지정하지 마십시오. 파일이 송신 끝에서 세그먼트되는 경우 멀티바이트 문자가 분할될 수 있기 때문입니다. 이러한 파일을 전송하고 변환하려면, 참조 메시지 엑시트가 변환하지 않고 전송이 완료될 때 수신 끝에서 파일을 변환하도록 MQFMT_STRING 이외의 다른 형식으로 지정하십시오.

참조 메시지 엑시트 샘플 컴파일

참조 메시지 엑시트 샘플을 컴파일하려면 IBM MQ가 설치되어 있는 플랫폼에 대한 명령을 사용하십시오.

필수 라이브러리

사용 가능한 32비트 및 64비트 라이브러리 목록은 [외부 라이브러리](#) 항목을 참조하십시오.

MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

amqsxrma를 컴파일하려면 다음 명령을 사용하십시오.

AIX의 경우

AIX

```
xlc_r -q64 -e MsgExit -bE:amqsxrm.exp -bM:SRE -o amqsxrm_64_r  
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -lmqm_r amqsxrma.c
```

IBM i의 경우

IBM i

```
CRTCMOD MODULE(MYLIB/AMQSRMA) SRCFILE(QMQMSAMP/QCSRC)
TERASPACE(*YES *TSIFC)
```

참고:

1. IFS 파일 시스템을 사용하도록 모듈을 작성하려면 SYSIFCOPT(*IFSI0) 옵션을 추가하십시오.
2. 스레드되지 않은 채널에서 사용할 프로그램을 작성하려면 다음 명령을 사용하십시오. CRTPGM PGM(MYLIB/AMQSRMA) BNDSRVPGM(QMQM/LIBMQM)
3. 스레드된 채널에서 사용할 프로그램을 작성하려면 다음 명령을 사용하십시오. CRTPGM PGM(MYLIB/AMQSRMA) BNDSRVPGM(QMQM/LIBMQM_R)

Linux의 경우

Linux

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsxrma amqsqrma.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-lmqm_r
```

Windows의 경우

Windows

IBM MQ는 이제 서버 패키지뿐만 아니라 클라이언트 패키지를 mqm 라이브러리에 제공하므로 다음 예에서는 mqmvx.lib 대신 mqm.lib을 사용합니다.

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

관련 개념

712 페이지의 『채널 엑시트 프로그램 작성』

다음 정보를 사용하면 채널 엑시트 프로그램 작성에 도움이 됩니다.

Get 참조 메시지 샘플(amqsgрма.c, AMQSGRM4) 디자인

이 주제는 Get 참조 메시지 샘플 디자인에 대해 설명합니다.

프로그램 로직은 다음과 같습니다.

1. 샘플이 트리거되고 큐와 큐 관리자 이름을 입력 트리거 메시지에서 추출합니다.
2. 그런 다음 지정된 큐 관리자에 MQCONN을 사용하여 연결하고 MQOPEN을 사용하여 지정된 큐를 엽니다.
3. 샘플은 큐에서 메시지를 가져오기 위해 루프 내에 15초의 대기 간격을 설정하여 MQGET을 발행합니다.
4. 메시지가 참조 메시지인 경우 샘플은 전송된 파일이 있는지 확인합니다.
5. 그런 다음 큐를 닫고 큐 관리자에서 연결을 끊습니다.

요청 샘플 프로그램

요청 샘플 프로그램에서는 클라이언트/서버 처리를 보여줍니다. 샘플은 서버 프로그램에서 처리하는 대상 서버 큐 위에 요청 메시지를 넣는 클라이언트입니다. 서버 프로그램이 응답 대상 큐에 응답 메시지를 넣을 때까지 대기합니다.

요청 샘플은 MQPUT 호출을 사용하여 일련의 요청 메시지를 대상 서버 큐에 넣습니다. 이러한 메시지는 응답 대상 큐로 SYSTEM.SAMPLE.REPLY 로컬 큐를 지정하며 로컬 또는 리모트 큐가 될 수 있습니다. 프로그램은 응답 메시지를 기다린 후 이를 표시합니다. 대상 서버 큐가 서버 애플리케이션에 의해 처리되고 있거나 애플리케이션이 해당 용도로 트리거되는 경우(조회, 설정 및 Echo 샘플 프로그램이 트리거되도록 설계됨)에만 응답이 송신됩니다. 첫 번째 응답이 도착하기 까지 C 샘플은 1분 대기하고(COBOL 샘플은 5분 대기함) 후속 응답의 경우 15초를 기다리지만 두 샘플 모두 응답 없이 종료할 수 있습니다. 다음 항목에서 다양한 플랫폼별 요청 샘플 프로그램의 이름을 확인하십시오:

-   C 및 COBOL MQI 샘플에 표시된 기능 [AIX and Linux](#)

-  C 및 COBOL MQI 샘플에 표시된 기능 Windows
-  C, COBOL 및 RPG MQI 샘플에 표시된 기능 IBM i

관련 참조

1073 페이지의 『요청 RPG 샘플 프로그램 IBM i』

요청 샘플 프로그램, AMQ3REQ4는 클라이언트/서버 처리를 보여줍니다. 샘플은 서버 프로그램이 처리한 요청 메시지를 큐에 넣는 클라이언트입니다. 서버 프로그램이 응답 메시지를 응답 대상 큐에 넣을 때까지 대기합니다.

요청 샘플 프로그램 실행

amqsreq0.c, amqsreq 및 amqsreqc 샘플 실행

프로그램의 C 버전은 세 개 매개변수를 사용합니다.

1. 대상 서버 큐의 이름(필수)
2. 큐 관리자의 이름(선택사항)
3. 응답 큐(선택사항)

예를 들어, 다음 중 하나를 입력하십시오.

- amqsreq myqueue qmanagername replyqueue
- amqsreqc myqueue qmanagername
- amq0req0 myqueue

여기서 myqueue는 대상 서버 큐의 이름이며 qmanagername은 myqueue를 소유하는 큐 관리자의 이름이며 replyqueue는 응답 큐의 이름입니다.

큐 관리자의 이름을 생략하는 경우 기본 큐 관리자가 큐를 소유하는 것으로 가정합니다. 응답 큐의 이름을 생략하는 경우 기본 응답 큐가 제공됩니다.

amq0req0.cbl 샘플 실행

COBOL 버전에는 매개변수가 없습니다. 기본 큐 관리자에 연결하고 실행하는 경우 다음 프롬프트가 표시됩니다.

```
Please enter the name of the target server queue
```

프로그램은 StdIn에서 입력을 선택하고 각 행을 대상 서버 큐에 추가합니다. 요청 메시지의 콘텐츠로 텍스트의 한 행을 사용합니다. 프로그램은 널 행이 읽히면 종료됩니다.

AMQSREQ4 샘플 실행

C 프로그램은 stdin(키보드)에서 데이터를 사용하여 공백 시간 종료 입력으로 메시지를 작성합니다. 프로그램은 대상 큐 이름(필수), 큐 관리자 이름(선택사항) 및 응답 대상 큐 이름(선택사항)의 최대 세 개의 매개변수를 사용합니다. 큐 관리자 이름을 지정하지 않으면, 기본 큐 관리자가 사용됩니다. 응답 대상 큐가 지정되지 않은 경우 SYSTEM.SAMPLE.REPLY 큐가 사용됩니다.

다음은 응답 대상 큐를 지정하지만 큐 관리자가 기본값을 사용하게 하는 C 샘플 프로그램을 호출하는 방법에 대한 예입니다.

```
CALL PGM(QMQM/AMQSREQ4) PARM('SYSTEM.SAMPLE.LOCAL' ' ' 'SYSTEM.SAMPLE.REPLY')
```

참고: 큐 이름은 대소문자를 구분합니다. 샘플 파일 생성 프로그램이 생성한 모든 대기열의 [AMQSAMP4](#) 이름은 대문자로 생성됩니다.

AMQ0REQ4 샘플 실행

COBOL 프로그램은 키보드에서 데이터를 승인하여 메시지를 작성합니다. 프로그램을 시작하려면, 프로그램을 호출하고 매개변수로 대상 큐의 이름을 지정하십시오. 프로그램은 키보드의 입력을 버퍼에 전달되도록 승인하고 각 텍스트 행에 대해 요청 메시지를 작성합니다. 프로그램은 키보드에 공백 행을 입력하는 경우 중지됩니다.

트리거링을 사용하여 *Request* 샘플 실행

샘플이 트리거링 및 조회, 설정 또는 Echo 샘플 프로그램 중 하나와 함께 사용되는 경우 입력의 행은 트리거된 프로그램에서 액세스할 큐의 큐 이름이어야 합니다.

ALW AIX, Linux, and Windows에서 트리거를 사용하여 요청 샘플 실행

AIX, Linux, and Windows에서는 한 세션에서 트리거 모니터 프로그램 RUNMQTRM을 시작하고, 다른 세션에서는 amqsreq 프로그램을 시작하십시오.

트리거링을 사용하여 샘플을 실행하려면,

1. 한 세션에서 트리거 모니터 프로그램 RUNMQTRM을 시작하십시오(이니시에이션 큐 SYSTEM.SAMPLE.TRIGGER을 사용할 수 있음).
2. 다른 세션에서 amqsreq 프로그램을 시작하십시오.
3. 대상 서버 큐가 정의되었는지 확인하십시오.

메시지를 넣기 위해 요청 샘플에 대한 대상 서버 큐로 사용할 수 있는 샘플 큐는 다음과 같습니다.

- SYSTEM.SAMPLE.INQ - 조회 샘플 프로그램의 경우
- SYSTEM.SAMPLE.SET - 설정 샘플 프로그램의 경우
- SYSTEM.SAMPLE.ECHO - Echo 샘플 프로그램의 경우

이러한 큐에는 FIRST 트리거 유형이 있으므로 요청 샘플을 실행하기 전에 큐에 이미 메시지가 있는 경우 서버 애플리케이션은 송신된 메시지에 의해 트리거되지 않습니다.

4. 조회, 설정 또는 Echo 샘플 프로그램에서 사용할 큐를 정의했는지 확인하십시오.

요청 샘플에서 메시지를 보내는 경우 트리거 모니터가 준비되었음을 의미합니다.

참고: RUNMQSC 및 amqscos0.tst 파일을 사용하여 작성된 샘플 프로세스 정의는 C 샘플을 트리거합니다. amqscos0.tst의 프로세스 정의를 변경하고 COBOL 버전을 사용하려면 이 업데이트된 파일과 함께 RUNMQSC를 사용하십시오.

[1024 페이지의 그림 148](#)은 요청 및 조회 샘플을 함께 사용하는 방법을 보여줍니다.

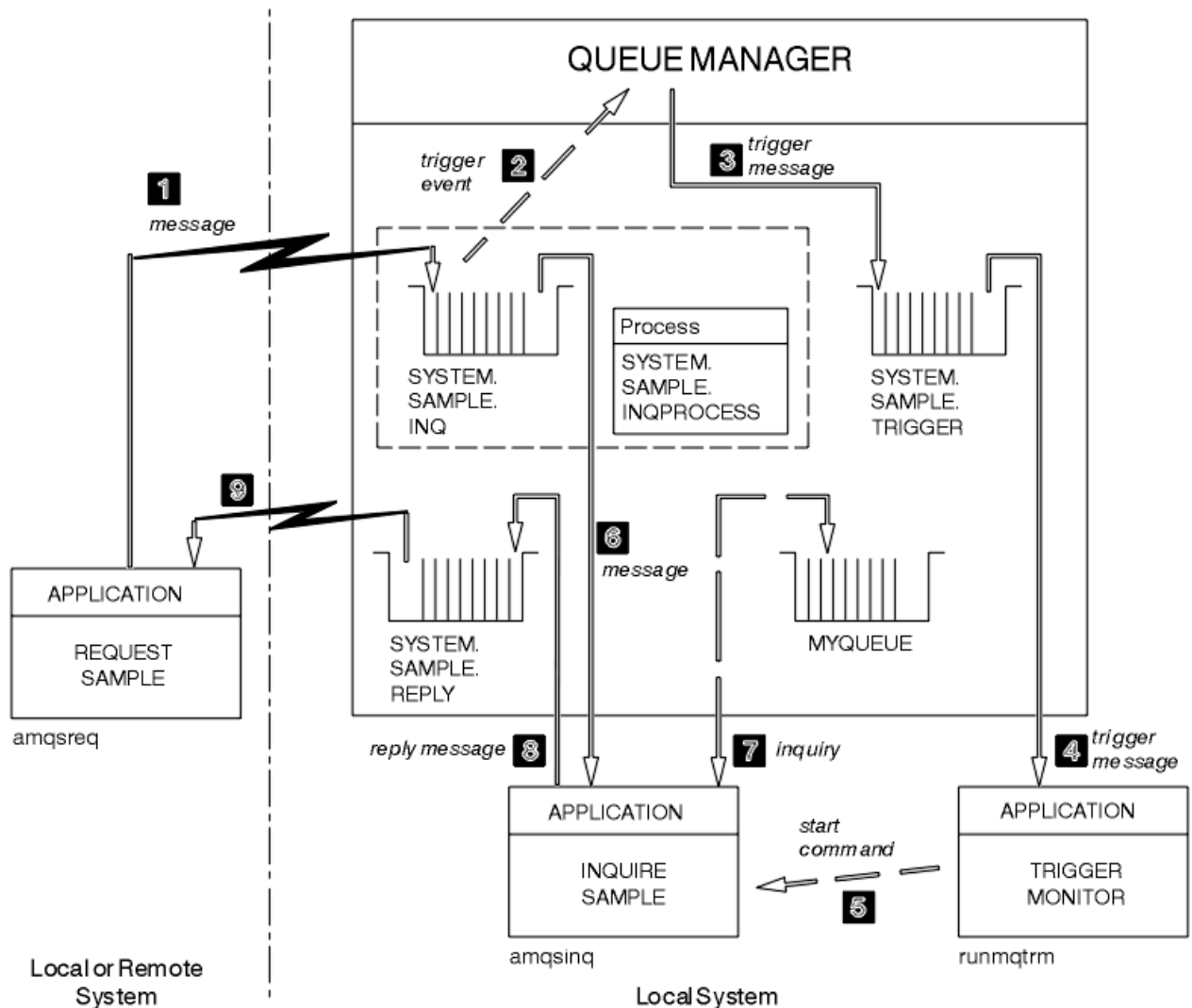


그림 148. 트리거링을 사용하는 요청 및 조회 샘플

1024 페이지의 그림 148에서 요청 샘플은 메시지를 대상 서버 큐 SYSTEM.SAMPLE.INQ에 넣고 조회 샘플은 MYQUEUE 큐를 조회합니다. 또는 amqsreq0.tst를 실행했을 때 정의된 샘플 큐 중 하나를 사용하거나 조회 샘플에 대해 정의한 다른 큐를 사용할 수 있습니다.

참고: 1024 페이지의 그림 148의 숫자는 이벤트 시퀀스를 표시합니다.

트리거링을 사용하여 요청 및 조회 샘플을 실행하려면,

1. 사용하려는 큐가 정의되어 있는지 확인하십시오. 샘플 큐를 정의하려면 amqsreq0.tst를 실행하고, MYQUEUE 큐를 정의하십시오.
2. 트리거 모니터 명령 RUNMQTRM을 실행하십시오.

```
RUNMQTRM -m qmanagername -q SYSTEM.SAMPLE.TRIGGER
```

3. 요청 샘플을 실행하십시오.

```
amqsreq SYSTEM.SAMPLE.INQ
```

참고: 프로세스 오브젝트는 트리거할 내용을 정의합니다. 클라이언트와 서버가 동일한 플랫폼에서 실행되고 있지 않는 경우, 트리거 모니터에서 시작된 프로세스는 ApplType을 정의해야 합니다. 그렇지 않으면 서버는 기본 정의(즉, 일반적으로 서버 시스템과 연관된 애플리케이션 유형)를 사용하므로 오류가 발생합니다.

애플리케이션 유형의 목록은 ApplType의 내용을 참조하십시오.

4. 조회 샘플에서 사용할 큐의 이름을 입력하십시오.

5. (요청 프로그램을 종료하려면) 빈 줄을 입력하십시오.
 6. 그러면 조회 프로그램이 MYQUEUE에서 얻은 데이터를 포함하는 메시지를 요청 샘플에서 표시합니다.
- 둘 이상의 큐를 사용할 수 있습니다. 이런 경우 [1024 페이지의 『4』 단계](#)에서 다른 큐의 이름을 입력하십시오. 트리거링에 대한 자세한 정보는 [873 페이지의 『트리거를 사용한 IBM MQ 애플리케이션 시작』](#)의 내용을 참조하십시오.

IBM i IBM i에서 트리거를 사용하여 요청 샘플 실행

IBM i에서는 하나의 작업에서 샘플 트리거 서버인 AMQSERV4를 시작한 후 다른 작업에서 AMQSREQ4를 시작하십시오. 요청 샘플 프로그램에서 메시지를 보내는 경우 트리거 서버가 준비되었음을 의미합니다.

참고:

1. 프로그램이 AMQSAMP4 생성한 샘플 정의는 해당 샘플의 C 버전을 트리거합니다. COBOL 버전을 트리거하려면 경우 SYSTEM.SAMPLE.ECHOPROCESS, SYSTEM.SAMPLE.INQPROCESS 및 SYSTEM.SAMPLE.SETPROCESS 프로세스 정의를 변경하십시오. CHGMQMPRC 명령을 사용하여(세부사항은 [MQ 프로세스 변경\(CHGMQMPRC\)](#) 참조) 이를 수행하거나, AMQSAMP4의 고유 버전을 편집하고 실행할 수 있습니다.
2. AMQSERV4에 대한 소스 코드는 C 언어에 대해서만 제공됩니다. 그러나 컴파일된 버전(COBOL 샘플과 함께 사용할 수 있는)이 라이브러리 QMQM에 제공됩니다.

다음 샘플 서버 큐에 요청 메시지를 넣을 수 있습니다.

- SYSTEM.SAMPLE.ECHO (Echo 샘플 프로그램의 경우)
- SYSTEM.SAMPLE.INQ (Inquire 샘플 프로그램의 경우)
- SYSTEM.SAMPLE.SET (Set 샘플 프로그램의 경우)

SYSTEM.SAMPLE.ECHO 프로그램에 대한 플로우차트가 [1027 페이지의 그림 149](#)에 표시됩니다. 예제 데이터 파일을 사용하여 이 서버에 대한 C 프로그램 요청을 발행하기 위한 명령은 다음과 같습니다.

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(ECHO)')
```

참고: 이 샘플 큐에는 FIRST 트리거 유형이 있으므로 Request 샘플을 실행하기 전에 큐에 이미 메시지가 있는 경우 서버 애플리케이션은 송신된 메시지에 의해 트리거되지 않습니다.

추가 예를 시도하려는 경우 다음 변형을 시도할 수 있습니다.

- AMQSERV4 대신 AMQSTRG4(또는 해당 명령행과 동등한 STRMQMTRM, 세부사항은 [MQ 트리거 모니터 시작\(STRMQMTRM\)](#) 참조)를 사용하여 대신 작업을 제출하지만, 잠재 작업 제출 지연으로 발생하는 상황을 따르기 쉽지 않을 수 있습니다.
- SYSTEM.SAMPLE.INQUIRE and SYSTEM.SAMPLE.SET 샘플 프로그램을 실행하십시오. 예제 데이터 파일을 사용하여 이러한 서버에 대한 C 프로그램 요청을 발행하기 위한 명령은 다음과 같습니다.

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(INQ)')
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(SET)')
```

이러한 샘플 큐에도 FIRST 트리거 유형이 있습니다.

요청 샘플 프로그램의 디자인

메시지를 넣을 수 있도록 프로그램은 대상 서버 큐를 엽니다. MQOO_OUTPUT 옵션과 함께 MQOPEN 호출을 사용합니다. 큐를 열 수 없는 경우 프로그램은 MQOPEN 호출이 리턴한 이유 코드를 포함하는 오류 메시지를 표시합니다.

그런 다음 응답 메시지를 가져올 수 있도록 프로그램은 SYSTEM.SAMPLE.REPLY라는 응답 대상 큐를 엽니다. 이를 위해 프로그램은 MQOO_INPUT_EXCLUSIVE 옵션과 함께 MQOPEN 호출을 사용합니다. 큐를 열 수 없는 경우 프로그램은 MQOPEN 호출에서 리턴한 이유 코드가 포함된 오류 메시지를 표시합니다.

입력의 각 행에 대해 프로그램은 텍스트를 버퍼로 해석하고 MQPUT 호출을 사용하여 해당 행의 텍스트를 포함하는 요청 메시지를 작성합니다. 이 호출에서 프로그램은 MQRO_EXCEPTION_WITH_DATA 보고서 옵션을 사용하

여 요청 메시지에 대한 전송된 보고 메시지에 메시지 데이터의 100바이트가 포함되도록 요청합니다. 프로그램은 입력의 끝에 도달하거나 MQPUT 호출에 실패할 때까지 계속됩니다.

그런 다음 프로그램은 MQGET 호출을 사용하여 큐에서 응답 메시지를 제거하고 응답에 포함된 데이터를 표시합니다. MQGET 호출은 MQGMO_WAIT, MQGMO_CONVERT 및 MQGMO_ACCEPT_TRUNCATED 옵션을 사용합니다. 첫 번째 응답에 대해 (서버 애플리케이션이 트리거될 시간을 허용하기 위해) *WaitInterval*은 COBOL 버전에서 5분이고 C 버전의 경우 1분이며 후속 응답의 경우 15초입니다. 큐에 메시지가 없는 경우 프로그램은 이 기간 동안 대기합니다. 이 간격이 끝나기 전에 메시지가 도착하지 않는 경우 호출은 실패하고 MQRC_NO_MSG_AVAILABLE 이유 코드를 리턴합니다. 또한 선언된 버퍼 크기보다 긴 메시지는 잘리도록 호출은 MQGMO_ACCEPT_TRUNCATED_MSG 옵션을 사용합니다.

프로그램은 호출에서 이러한 필드를 검색하는 메시지에 포함된 값으로 설정하기 때문에 각 MQGET 호출 후에 MQMD 구조의 *MsgId* 및 *CorrelId* 필드를 지우는 방법을 보여줍니다. 이러한 필드를 지운다는 것은 연속적인 MQGET 호출이 메시지가 큐에 보유되는 순서대로 메시지를 검색한다는 것을 의미합니다.

프로그램은 MQGET 호출에서 MQRC_NO_MSG_AVAILABLE 이유 코드를 리턴하거나 MQGET 호출에 실패할 때까지 계속됩니다. 호출에 실패하는 경우 프로그램은 이유 코드를 포함하는 오류 메시지를 표시합니다.

그런 다음 프로그램은 MQCLOSE 호출을 사용하여 대상 서버 큐와 응답 대상 큐를 둘 다 닫습니다.

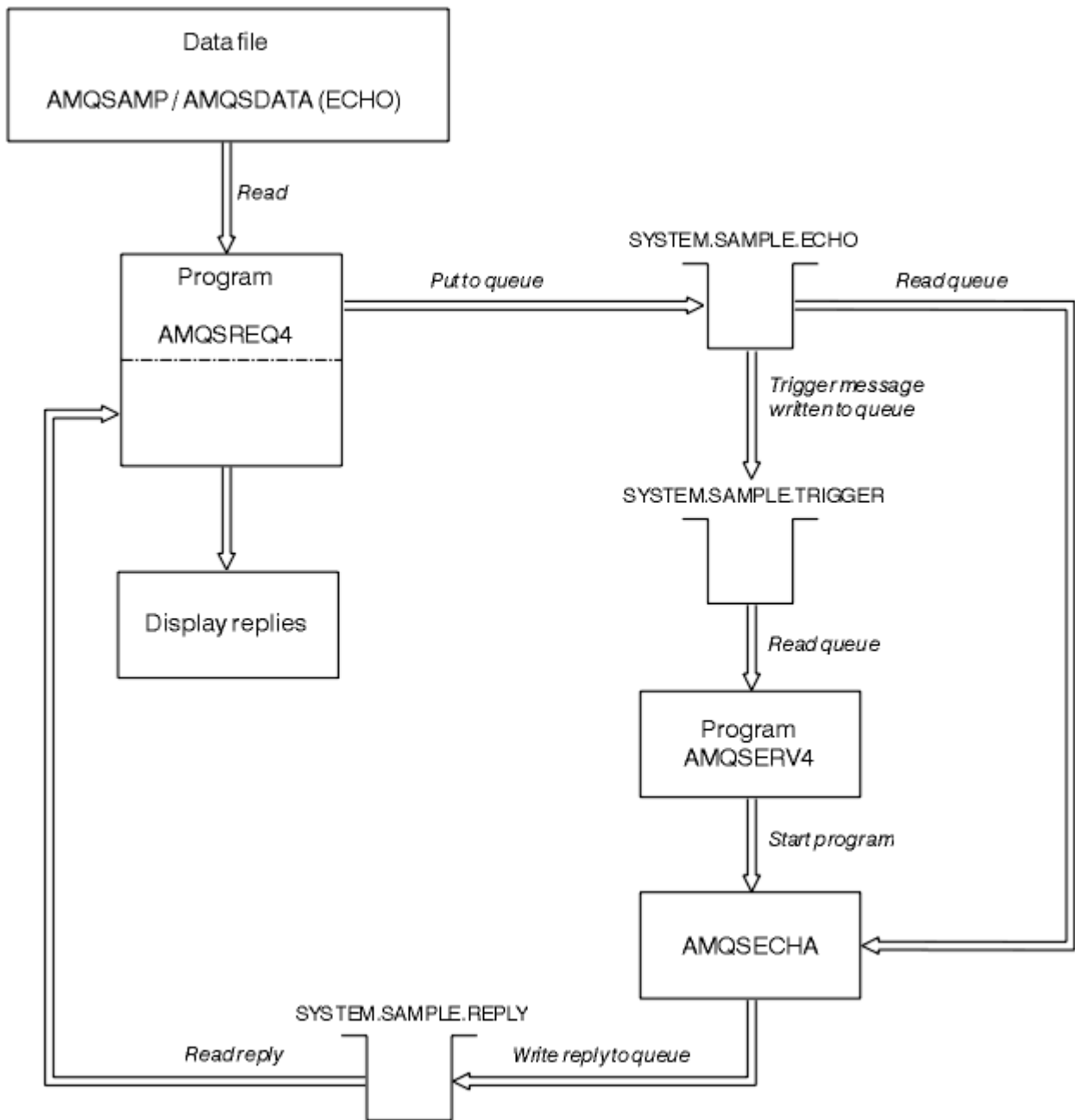


그림 149. 샘플 IBM i 클라이언트/서버(Echo) 프로그램 플로우차트

설정 샘플 프로그램

설정 샘플 프로그램은 큐의 **InhibitPut** 속성을 변경하기 위해 MQSET 호출을 사용하여 큐에서 put 조작을 금지합니다. 또한 설정 샘플 프로그램의 디자인에 대해 알아보십시오.

다음 플랫폼별 프로그램 명칭은 해당 항목을 참조하십시오:

- ▶ **Linux** ▶ **AIX** C 및 COBOL MQI 샘플에 표시된 기능 AIX and Linux
- ▶ **Windows** C 및 COBOL MQI 샘플에 표시된 기능 Windows
- ▶ **IBM i** C, COBOL 및 RPG MQI 샘플에 표시된 기능 IBM i

프로그램은 트리거된 프로그램으로 실행하기 위한 것이므로 유일한 입력은 조회될 속성과 함께 대상 큐의 이름을 포함하는 MQTMC2(트리거 메시지) 구조입니다. C 버전도 큐 관리자 이름을 사용합니다. COBOL 버전은 기본 큐 관리자를 사용합니다.

작업할 트리거링 프로세스의 경우 사용하려는 설정 샘플 프로그램이 SYSTEM.SAMPLE.SET 큐에 도착하는 메시지에 의해 트리거되는지 확인하십시오. 이를 수행하려면 프로세스 정의 SYSTEM.SAMPLE.SETPROCESS의 *ApplicId* 필드에서 사용하려는 설정 샘플 프로그램의 이름을 지정하십시오. 샘플 큐에는 FIRST 트리거 유형이 있습니다. 요청 샘플을 실행하기 전에 큐에 이미 메시지가 있는 경우 설정 샘플은 송신하는 메시지에 의해 트리거되지 않습니다.

올바르게 정의가 설정된 경우

- ▶ **ALW** AIX, Linux, and Windows 시스템의 경우 한 세션에서는 **runmqtrm** 프로그램을 시작하고 다른 세션에서는 **amqsreq** 프로그램을 시작하십시오.
- ▶ **IBM i** IBM i의 경우 하나의 세션에서 AMQSERV4 프로그램을 시작한 후 다른 세션에서 AMQSREQ4 프로그램을 시작하십시오. AMQSERV4 대신에 AMQSTRG4를 사용할 수 있지만, 잠재적인 작업 제출 지연으로 인해 발생하는 상황에 쉽게 대응하지 못할 수 있습니다.

요청 샘플 프로그램을 사용하여 각각 큐 이름만 포함하는 요청 메시지를 SYSTEM.SAMPLE.SET 큐에 송신합니다. 각 요청 메시지의 경우 설정 샘플 프로그램은 Put 조작이 지정된 큐에서 금지되었다는 확인을 포함하는 응답 메시지를 송신합니다. 응답은 요청 메시지에 지정된 응답 대상 큐로 송신됩니다.

설정 샘플 프로그램의 디자인

프로그램은 시작했을 때 전달된 트리거 메시지 구조에서 이름 지정된 큐를 엽니다. (명확성을 위해 이를 요청 큐라고 합니다.) 프로그램은 MQOPEN 호출을 사용하여 공유 입력에 대한 이 큐를 엽니다.

프로그램은 MQGET 호출을 사용하여 이 큐에서 메시지를 제거합니다. 이 호출은 5초의 대기 간격으로 MQGMO_ACCEPT_TRUNCATED_MSG 및 MQGMO_WAIT 옵션을 사용합니다. 프로그램은 요청 메시지인지를 확인하기 위해 각 메시지의 디스크립터를 테스트합니다. 아닌 경우, 프로그램은 메시지를 제거하고, 경고 메시지를 표시합니다.

요청 큐에서 제거된 각 요청 메시지의 경우 프로그램은 데이터에 포함된 큐의 이름을 읽고(대상 큐라고 함) MQOO_SET 옵션이 포함된 MQOPEN 호출을 사용하여 해당 큐를 엽니다. 그런 다음 프로그램은 MQSET 호출을 사용하여 대상 큐의 **InhibitPut** 속성 값을 MQQA_PUT_INHIBITED로 설정합니다.

MQSET 호출에 성공하는 경우 프로그램은 MQPUT1 호출을 사용하여 응답 메시지를 응답 대상 큐에 넣습니다. 이 메시지에는 PUT inhibited 문자열이 포함됩니다.

MQOPEN 또는 MQSET 호출이 성공하지 못한 경우 프로그램은 MQPUT1 호출을 사용하여 report 메시지를 응답 대상 큐에 넣습니다. 이 보고 메시지에 있는 메시지 디스크립터의 *Feedback* 필드는 어떤 호출이 실패했는지에 따라 MQOPEN 또는 MQSET 호출에서 리턴된 이유 코드입니다.

MQSET 호출 후 프로그램은 MQCLOSE 호출을 사용하여 대상 큐를 닫습니다.

요청 큐에 남아 있는 메시지가 없는 경우 프로그램은 해당 큐를 닫고 큐 관리자의 연결을 끊습니다.

관련 참조

1076 페이지의 『Set RPG 샘플 프로그램 IBM i』

설정 샘플 프로그램, AMQ3SET4는 MQSET 호출을 사용하여 큐에서 Put 조작이 큐의 **InhibitPut** 속성을 변경하지 못하도록 금지합니다.

TLS 샘플 프로그램

AMQSSSLC는 MQCONNX 호출에 TLS 클라이언트 연결 정보를 제공하기 위해 MQCNO 및 MQSCO 구조를 사용하는 방법을 보여주는 샘플 C 프로그램입니다. 이를 통해 클라이언트 MQI 애플리케이션은 해당 클라이언트 연결 채널의 정의 및 TLS 설정을 런타임 시 클라이언트 채널 정의 테이블(CCDT) 없이 제공할 수 있습니다.

연결 이름이 제공되면 프로그램은 MQCD 구조에서 클라이언트 연결 채널 정의를 구성합니다.

키 저장소 파일의 스템 이름이 제공되는 경우 프로그램은 MQSCO 구조를 구성합니다. OCSP 응답자 URL도 제공되는 경우에는 프로그램은 인증 정보 레코드 MQAIR 구조를 구성합니다.

그런 다음 프로그램은 MQCONNX를 사용하여 큐 관리자에 연결합니다. 연결된 큐 관리자의 이름을 조회하여 출력합니다.

이 프로그램은 MQI 클라이언트 애플리케이션으로 링크되기 위한 것입니다. 하지만 일반적인 MQI 애플리케이션으로 연결할 수도 있으며, 이 경우 단순히 로컬 큐 매니저에 연결하고 클라이언트 연결 정보는 무시합니다.

키 저장소에 접근하기 위한 암호가 파일에 저장되어 있지 않은 경우, 애플리케이션 실행 시 **amqssslc** 해당 암호를 입력해야 합니다. 암호는 다음 방법 중 하나로 입력할 수 있습니다:

- 암호를 입력하라는 메시지를 표시하도록 **amqssslc** 요청하거나
- **MQKEYRPWD** 환경 변수를 사용하거나,
- 클라이언트 구성 파일에서 속성 **SSLKeyRepositoryPassword** 사용

애플리케이션에 키 저장소 암호를 제공하는 IBM MQ MQI client 방법에 대한 자세한 내용은 다음을 참조하십시오. [예 IBM MQ MQI client 대한 키 저장소 AIX, Linux, and Windows 비밀번호를 제공합니다.](#)

amqssslc 다음 매개변수를 받으며, 모두 선택 사항입니다:

-m QmgrName

연결할 큐 관리자의 이름

-c ChannelName

사용할 채널의 이름

-x ConnName

서버 연결 이름

TLS 매개변수

-k KeyReposFileName

키 저장소 파일의 이름. 파일 확장자가 지정되지 않은 경우, 기본 확장자로 .kdb. 예를 들면, 다음과 같습니다.

```
/home/user/client.kdb  
C:\User\client.p12
```

-s CipherSpec

TLS 채널 CipherSpec 큐 매니저의 SVRCONN 채널 정의에 **SSLCIPH** 해당하는 문자열.

-f

인증된 FIPS 140-2 알고리즘만 사용해야 함을 명시합니다.

-b VALUE1[,VALUE2...]

Suite B 준수 알고리즘만 사용해야 함을 지정합니다. 이 매개변수는 하나 이상의 다음 값이 쉼표로 구분된 목록입니다. NONE,128_BIT,192_BIT. 이러한 값들은 환경 **MQSUITEB** 변수의 값들과 동일한 의미를 가지며, 클라이언트 구성 파일의 SSL 구절에 있는 동등한 **EncryptionPolicySuiteB** 설정과도 동일합니다.

-p Policy

사용될 인증서 유효성 검증 정책을 지정합니다. 다음 값 중 하나일 수 있습니다.

ANY

보안 소켓 라이브러리에서 지원하는 인증서 유효성 검증 정책을 각각 적용하고 정책이 인증서 체인이 유효하다고 판단하는 경우 인증서를 승인합니다. 이 설정은 최신 인증서 표준을 준수하지 않는 오래된 디지털 인증서와의 최대 역호환성을 위해 사용할 수 있습니다.

RFC5280

RFC 5280 준수 인증서 유효성 검증 정책만 적용합니다. 이 설정은 임의(ANY) 설정보다 엄격한 유효성 검증을 제공하지만 일부 오래된 디지털 인증서는 거부합니다.

기본값은 ANY입니다.

-l CertLabel

보안 연결에 대해 사용할 인증서 레이블입니다.

참고: 소문자를 사용하여 값을 지정해야 합니다.

-w

키 저장소 암호 문구를 입력하도록 프롬프트를 **amqssslc** 표시하도록 지정합니다.

-i

초기 키 저장소 암호화 암호에 사용할 초기 키를 입력하라는 프롬프트를 **amqssslc** 표시하도록 지정합니다.

키 저장소 암호화 시 초기 키 파일을 지정했다면 **runmqicred** 이 옵션을 명시하십시오.

OCSP 인증서 폐기 매개변수:

-o URL

OCSP 응답자 URL

다음 환경 변수 중 하나를 설정하여 큐 관리자와의 인증에 사용되는 자격 증명을 제공할 수도 있습니다:

MQSAMP_사용자_ID

큐 매니저에 사용자 ID와 비밀번호를 사용하여 인증하려면 연결 인증에 사용할 사용자 ID를 설정하십시오. 프로그램은 사용자 ID에 해당하는 비밀번호를 입력하도록 요청합니다.

Linux AIX MQSAMP_TOKEN

큐 관리자와 인증하기 위해 인증 토큰을 제공하려면 비어 있지 않은 값으로 설정하십시오. 프로그램이 인증 토큰을 요청합니다.

TLS 샘플 프로그램 실행

TLS 샘플 프로그램을 실행하려면 먼저 TLS 환경을 설정해야 합니다. 그런 다음 여러 매개변수를 제공하면서 명령행에서 샘플을 실행합니다.

이 태스크 정보

다음 명령은 개인 인증서를 사용하여 샘플 프로그램을 실행합니다. 명령을 변경하여 예를 들어, CA 인증서를 사용하고 OCSP 응답자를 사용하여 해당 상태를 확인할 수 있습니다. 샘플 내의 명령을 참조하십시오.

프로시저

1. QM1이라는 이름의 큐 관리자를 작성하십시오. 자세한 정보는 `crtmqm`의 내용을 참조하십시오.
2. 큐 관리자에 대한 키 저장소를 작성하십시오. 자세한 정보는 [AIX, Linux, and Windows에서 키 저장소 설정](#)을 참조하십시오.
3. 클라이언트에 대한 키 저장소를 작성하십시오. `clientkey.kdb`라고 합니다.
키 저장소를 생성할 때 키 저장소 비밀번호를 파일에 저장하십시오.
4. 큐 관리자에 대한 개인 인증서를 작성하십시오. 자세한 정보는 [AIX, Linux, and Windows에서 자체 서명된 개인 인증서 작성](#)을 참조하십시오.
5. 클라이언트에 대한 개인 인증서를 작성하십시오.
6. 서버 키 저장소에서 개인 인증서를 추출하고 클라이언트 저장소에 추가하십시오. 자세한 정보는 [AIX, Linux, and Windows의 키 저장소에서 자체 서명 인증서의 공용 부분 추출 및 AIX, Linux, and Windows 시스템의 키 저장소에 CA 인증서\(또는 자체 서명 인증서의 공용 부분\) 추가](#)를 참조하십시오.
7. 클라이언트 키 저장소에서 개인 인증서를 추출하고 서버 키 저장소에 추가하십시오.
8. MQSC 명령을 사용하여 서버 연결 채널을 작성하십시오.

```
DEFINE CHANNEL(QM1SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256)
```

자세한 정보는 [서버 연결 채널](#)을 참조하십시오.

9. 큐 관리자에서 채널 리스너를 정의하고 시작하십시오. 자세한 정보는 [DEFINE LISTENER](#) 및 [START LISTENER](#)를 참조하십시오.
10. 다음 명령을 사용하여 샘플 프로그램을 실행하십시오.

```
AMQSSSLC -m QM1 -c QM1SVRCONN -x localhost
-k "C:\Program Files\IBM\MQ\clientkey.kdb" -s TLS_RSA_WITH_AES_128_CBC_SHA256
-o http://dummy.OCSP.responder
```

결과

샘플 프로그램은 다음 조치를 수행합니다.

1. 지정된 옵션을 사용하여 지정된 큐 관리자 또는 기본 큐 관리자에 연결합니다.

2. 큐 관리자를 열고 해당 이름을 조회합니다.
3. 큐 관리자를 닫습니다.
4. 큐 관리자로부터 연결을 끊습니다.

샘플 프로그램이 성공적으로 실행되면 다음 예와 유사한 출력이 표시됩니다.

```
Sample AMQSSSLC start
Connecting to queue manager QM1
Using the server connection channel QM1SVRCONN
on connection name localhost.
Using TLS CipherSpec TLS_RSA_WITH_AES_128_CBC_SHA256
Using TLS key repository stem C:\Program Files\IBM\MQ\clientkey
Using OCSP responder URL http://dummy.OCSP.responder
Connection established to queue manager QM1
```

Sample AMQSSSLC end

샘플 프로그램에 문제가 발생하는 경우 적합한 오류 메시지를 표시합니다. 예를 들어, 올바르지 않은 OCSP 응답자 URL을 지정하는 경우 다음 메시지를 수신합니다.

MQCONN ended with reason code 2553

이유 코드 목록은 [API 완료 및 이유 코드를 참조하십시오](#).

트리거링 샘플 프로그램

트리거링 샘플에서 제공된 기능은 **runmqtrm** 프로그램의 트리거 모니터에 제공된 기능의 서브세트입니다.

다음 플랫폼별 프로그램 명칭은 해당 항목을 참조하십시오:

-   [C 및 COBOL MQI 샘플에 표시된 기능 AIX and Linux](#)
-  [C 및 COBOL MQI 샘플에 표시된 기능 Windows](#)
-  [C, COBOL 및 RPG MQI 샘플에 표시된 기능 IBM i](#)

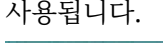
트리거링 샘플의 디자인

트리거링 샘플 프로그램은 MQOO_INPUT_AS_Q_DEF 옵션과 함께 MQOPEN 호출을 사용하여 이니시에이션 큐를 엽니다. 무제한 대기 간격을 지정하는 MQGMO_ACCEPT_TRUNCATED_MSG 및 MQGMO_WAIT 옵션과 함께 MQGET 호출을 사용하여 이니시에이션 큐에서 메시지를 가져옵니다. 프로그램은 차례대로 메시지를 가져오기 위해 각 MQGET 호출 전에 *MsgId* 및 *CorrelId* 필드를 지웁니다.

이니시에이션 큐에서 메시지를 검색한 경우 MQTM 구조와 동일한 크기인지 확인하기 위해 프로그램은 메시지의 크기를 확인하여 메시지를 테스트합니다. 이 테스트에 실패하면 프로그램은 경고를 표시합니다.

올바른 트리거 메시지의 경우 트리거링 샘플은 다음 필드의 데이터를 복사합니다. *ApplicId*, *EnvrData*, *Version* 및 *ApplType*. 이러한 필드의 마지막 두 개는 숫자이므로 프로그램은 IBM i, AIX, Linux, and Windows 시스템에 대해 MQTMC2 구조에서 사용할 문자열 대체를 작성합니다.

트리거링 샘플은 트리거 메시지의 *ApplicId* 필드에 지정된 애플리케이션에 시작 명령을 발행하고 MQTMC2 또는 MQTMC(트리거 메시지의 문자 버전) 구조를 전달합니다.

-  AIX, Linux, and Windows 시스템에서 *EnvrData* 필드는 명령 문자열을 호출하기 위한 확장으로 사용됩니다.
-  IBM i에서는 작업 제출 매개변수(예: 작업 우선순위 또는 작업 설명)로 사용됩니다.

마지막으로 프로그램은 이니시에이션 큐를 닫습니다.

IBM i의 트리거링 샘플 프로그램 종료

IBM i

sysrequest 옵션 2(ENDRQS)에 의해 또는 트리거 큐에서 가져오기를 금지하여 트리거 모니터 프로그램을 종료할 수 있습니다.

샘플 트리거 큐가 사용되는 경우 명령은 다음과 같습니다.

```
CHGMQM QNAME('SYSTEM.SAMPLE.TRIGGER') MQMNAME GETENBL(*NO)
```

중요사항: 이 큐에서 트리거링을 다시 시작하기 전에 다음 명령을 입력해야 합니다.

```
CHGMQM QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*YES)
```

관련 참조

1077 페이지의 『트리거링 RPG 샘플 프로그램 IBM i』

IBM MQ for IBM i는 ILE/RPG로 작성된 두 가지 트리거 샘플 프로그램을 제공합니다.

트리거링 샘플 프로그램 실행

이 주제에는 트리거링 샘플 프로그램 실행에 대한 정보가 포함되어 있습니다.

amqstrg0.c, amqstrg 및 amqstrgc 샘플 실행

프로그램은 두 개의 매개변수를 사용합니다.

1. 이니시에이션 큐 이름(필수)
2. 큐 관리자의 이름(선택사항)

큐 관리자가 지정되지 않은 경우 기본 관리자에 연결합니다. 샘플 이니시에이션 큐는 amqscos0.tst를 실행했을 때 정의됩니다. 해당 큐의 이름은 SYSTEM.SAMPLE.TRIGGER이며 이 프로그램을 실행할 때 큐를 사용할 수 있습니다.

참고: 이 샘플의 함수는 runmqtrm 프로그램에 제공된 전체 트리거링 함수의 서브세트입니다.

AMQSTRG4 샘플 실행

IBM i

IBM i 환경을 위한 트리거 모니터입니다. 시작될 각 애플리케이션을 위해 하나의 IBM i 작업을 제출합니다. 각 트리거 메시지와 연관된 추가적인 처리가 있다는 것을 의미합니다.

AMQSTRG4(QCSRC)에는 두 개 매개변수를 사용합니다. 제공하기 위한 이니시에이션 큐의 이름 및 큐 관리자의 이름(선택사항)입니다. AMQSAMP4 (QCLSRC)에서 샘플 프로그램 실행 시 사용할 수 있는 샘플 초기화 대기열 (SYSTEM.SAMPLE.TRIGGER)을 정의합니다.

예 트리거 큐를 사용하여 발행할 명령은 다음과 같습니다.

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

또는 CL 등가 STRMQMTRM을 사용할 수 있습니다. 세부사항은 [MQ 트리거 모니터 시작\(STRMQMTRM\)](#)을 참조하십시오.

AMQSERV4 샘플 실행

IBM i

이는 IBM i 환경을 위한 트리거 서버입니다. 각 트리거 메시지의 경우 이 서버는 지정된 애플리케이션을 시작하기 위해 고유 작업에서 시작 명령을 실행합니다. 트리거 서버는 CICS 트랜잭션을 호출할 수 있습니다.

AMQSTRG4는 두 개 매개변수를 사용합니다. 제공하기 위한 이니시에이션 큐의 이름 및 큐 관리자의 이름(선택사항)입니다. [AMQSAMP4](#) 샘플 프로그램 실행 시 사용할 수 있는 샘플 초기화 큐 (SYSTEM.SAMPLE.TRIGGER)를 정의합니다.

예 트리거 큐를 사용하여 발행할 명령은 다음과 같습니다.

```
CALL PGM(QMQM/AMQSERV4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

트리거 서버의 디자인

트리거 서버의 설계는 몇 가지 예외를 제외하고 트리거 모니터의 설계와 유사합니다.

트리거 서버와 트리거 모니터의 차이점은 다음과 같습니다. 트리거 서버는:

- MQAT_OS400 애플리케이션 및 MQAT_CICS 애플리케이션을 허용합니다.
- **IBM i** IBM i 작업을 제출하지 않고 자체 작업에서 IBM i 애플리케이션을 호출합니다 (또는 STRCICSUSR를 사용하여 CICS 애플리케이션 시작).
- 응용 프로그램에서는 CICS STRCICSUSR 명령의 트리거 메시지에서 예를 들어 지역을 CICS 지정하기 위해 를 대체합니다 *EnvData*.
- 동시에 여러 트리거 서버가 실행될 수 있도록 공유 입력에 대한 이니시에이션 큐를 엽니다.

참고: AMQSERV4에서 시작한 프로그램은 트리거 서버를 중지하기 때문에 MQDISC 호출을 사용하면 안됩니다. AMQSERV4에서 시작한 프로그램이 MQCONN 호출을 사용하는 경우 MQRC_ALREADY_CONNECTED 이유 코드를 받습니다.

관련 개념

873 페이지의 『트리거를 사용한 IBM MQ 애플리케이션 시작』

트리거와 트리거를 사용하여 IBM MQ 애플리케이션을 시작하는 방법에 대해 학습하십시오.

ALW AIX, Linux, and Windows에서 TUXEDO 샘플 사용

TUXEDO에 대한 Put 및 Get 샘플 프로그램 및 TUXEDO에서 서버 환경 빌드에 대해 알아봅니다.

시작하기 전에

이러한 샘플을 실행하려면 먼저 서버 환경을 빌드해야 합니다.

이 태스크 정보

참고: 이 섹션에서는 백슬래시(\) 문자를 사용하여 둘 이상의 행에 걸쳐 있는 긴 명령을 분할합니다. 이 문자를 입력하지 마십시오. 각 명령을 단일 행으로 입력하십시오.

ALW 서버 환경 빌드

다른 플랫폼에 대한 IBM MQ용 서버 환경 빌드에 대한 정보입니다.

시작하기 전에

작동 중인 TUXEDO 환경이 있다고 가정합니다.

AIX AIX용 서버 환경 빌드(32비트)

IBM MQ for AIX용 서버 환경을 빌드하는 방법입니다(32비트).

프로시저

1. 서버 환경이 빌드되는 디렉토리(예: APPDIR)를 작성하고 이 디렉토리의 모든 명령을 실행하십시오.
2. 다음 환경 변수를 내보내십시오. 여기서 TUXDIR은(는) TUXEDO의 루트 디렉토리이고 MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstvx.flds
$ export VIEWFILES=/APPDIR/amqstvx.v
$ export LIBPATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib:/lib
```

3. TUXEDO 파일 udataobj/RM에 다음 행을 추가하십시오.

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmx -lmqm
```

4. 명령을 실행하십시오.

```
$ mkfldhdr MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
-f MQ_INSTALLATION_PATH/lib/libmqm.a \
-r MQSeries_XA_RMI -s MPUT1:MPUT \
-s MGET1:MGET \
-v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
-f MQ_INSTALLATION_PATH/lib/libmqm.a \
-r MQSeries_XA_RMI -s MPUT2:MPUT \
-s MGET2:MGET \
-v -bshm
$ buildclient -o doputs -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
-f MQ_INSTALLATION_PATH/lib/libmqm.a
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
-f MQ_INSTALLATION_PATH/lib/libmqm.a
```

5. ubbstxcx.cfg를 편집하고 필요한 경우 시스템 이름, 작업 디렉토리 및 큐 관리자의 세부사항을 추가하십시오.

```
$ tmloadcf -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. TLOGDEVICE를 작성하십시오.

```
$tmadmin -c
```

프롬프트가 표시됩니다. 이 프롬프트에 다음을 입력하십시오.

```
> crdl -z /APPDIR/TLOG1
```

7. 큐 관리자를 시작하십시오.

```
$ stmqm
```

8. TUXEDO를 시작하십시오.

```
$ tmboot -y
```

다음에 수행할 작업

이제 doputs 및 dogets 프로그램을 사용하여 메시지를 큐에 넣고 큐에서 메시지를 검색할 수 있습니다.

AIX IBM MQ용 서버 환경 빌드(64비트)

IBM MQ for AIX용 서버 환경을 빌드하는 방법입니다(64비트).

프로시저

1. 서버 환경이 빌드되는 디렉토리(예: APPDIR)를 작성하고 이 디렉토리의 모든 명령을 실행하십시오.
2. 다음 환경 변수를 내보내십시오. 여기서 TUXDIR은(는) TUXEDO의 루트 디렉토리이고 MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 상위 레벨 디렉토리를 나타냅니다.

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib64"
$ export LDOPTS="-lmqm"
$ export FIELDTBL= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib64: MQ_INSTALLATION_PATH/lib64:/lib64
```

3. TUXEDO 파일 udataobj/RM에 다음 행을 추가하십시오.

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmx64 -lmqm
```

4. 명령을 실행하십시오.

```
$ mkfldhdr MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
-f MQ_INSTALLATION_PATH/lib64/libmqm.a \
-r MQSeries_XA_RMI -s MPUT1:MPUT \
-s MGET1:MGET \
-v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
-f MQ_INSTALLATION_PATH/lib64/libmqm.a \
-r MQSeries_XA_RMI -s MPUT2:MPUT \
-s MGET2:MGET \
-v -bshm
$ buildclient -o doputs -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
-f MQ_INSTALLATION_PATH/lib64/libmqm.a
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
-f MQ_INSTALLATION_PATH/lib64/libmqm.a
```

5. ubbstxcx.cfg를 편집하고 필요한 경우 시스템 이름, 작업 디렉토리 및 큐 관리자의 세부사항을 추가하십시오.

```
$ tmloadcf -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. TLOGDEVICE를 작성하십시오.

```
$tmadmin -c
```

프롬프트가 표시됩니다. 이 프롬프트에 다음을 입력하십시오.

```
> crdl -z /APPDIR/TLOG1
```

7. 큐 관리자를 시작하십시오.

```
$ stimqm
```

8. TUXEDO를 시작하십시오.

```
$ tmboot -y
```

다음에 수행할 작업

이제 doputs 및 dogets 프로그램을 사용하여 메시지를 큐에 넣고 큐에서 메시지를 검색할 수 있습니다.

Windows Windows용 서버 환경 빌드(32비트)
IBM MQ for Windows에 대한 서버 환경을 빌드합니다(32비트).

이 태스크 정보

참고: 다음에서 *VARIABLES*로 식별되는 필드를 디렉토리 경로로 변경하십시오.

표 168. 디렉토리 경로로 변경할 필드	
필드	디렉토리 경로
<i>MQMDIR</i>	IBM MQ이(가) 설치될 때 지정된 디렉토리 경로(예: g:\Program Files\IBM\MQ).
<i>TUXDIR</i>	TUXEDO가 설치될 때 지정된 디렉토리 경로(예: f:\tuxedo).
<i>APPDIR</i>	샘플 애플리케이션에 사용할 디렉토리 경로(예: f:\tuxedo\apps\mqapp).

```

*RESOURCES
IPCKEY      99999
UID         0
GID         0
MAXACCESSERS 20
MAXSERVERS  20
MAXSERVICES 50
MASTER      SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Program Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULLOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPN0=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2

```

그림 150. IBM MQ for Windows에 대한 ubbstxcn.cfg 파일의 예

참고: 시스템 이름 *MachineName* 및 디렉토리 경로를 사용자 설치에 일치하도록 변경하십시오. 또한 큐 관리자 이름 *MYQUEUEMANAGER*를 연결하려는 큐 관리자의 이름으로 변경하십시오.

IBM MQ for Windows의 샘플 ubbconfig 파일은 [1036 페이지의 그림 150](#)에 나열됩니다. 이는 IBM MQ 샘플 디렉토리에 ubbstxcn.cfg(으)로 제공됩니다.

IBM MQ for Windows에 제공된 샘플 makefile([1037 페이지의 그림 151 참조](#))은 ubbstxmn.mak(이)라고 하며 IBM MQ 샘플 디렉토리에 있습니다.


```

TUXDIR = f:\tuxedo
MQMDIR = g:\Program Files\IBM\WebSphere MQ
APPDIR = f:\tuxedo\apps\mqapp
MQMLIB = $(MQMDIR)\tools\lib
MQMINC = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
$(TUXDIR)\bin\mkfldhdr -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
$(TUXDIR)\bin\viewc -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
$(TUXDIR)\bin\buildtms -o MQXA -r MQSERIES_XA_RMI
$(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
-f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
-r MQSERIES_XA_RMI \
-s MPUT1:MPUT -s MGET1:MGET
$(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
-f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
-r MQSERIES_XA_RMI \
-s MPUT2:MPUT -s MGET2:MGET
$(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
-f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
$(TUXDIR)\bin\buildclient -o dogets -f $(MQMSAMP)\amqstxgx.c \
-f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
$(TUXDIR)\bin\tmloadcf -y $(APPDIR)\ubbstxcn.cfg

```

그림 151. IBM MQ for Windows용 샘플 TUXEDO make 파일

서버 환경 및 샘플을 빌드하려면 다음 단계를 완료하십시오.

프로시저

1. 샘플 애플리케이션을 빌드할 애플리케이션 디렉토리를 작성하십시오. 예를 들어,

```
f:\tuxedo\apps\mqapp
```

2. IBM MQ 샘플 디렉토리의 다음 샘플 파일을 애플리케이션 디렉토리에 복사하십시오.

- amqstxmn.mak
- amqstxen.env
- ubbstxcn.cfg

3. 이러한 각 파일을 편집하여 설치에 사용된 디렉토리 경로 및 디렉토리 이름을 설정하십시오.
4. 연결하려는 시스템 이름 및 큐 관리자의 세부사항을 추가하려면 ubbstxcn.cfg을(를) 편집하십시오([1036 페이지의 그림 150](#) 참조).
5. TUXEDO 파일 TUXDIR\dataobj\rm에 다음 행을 추가하십시오.

```
MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib\mqmxa.lib MQMDIR\tools\lib\mqm.lib
```

새 항목은 파일에서 한 행에 있어야 합니다.

6. 다음 환경 변수를 설정하십시오.

```

TUXDIR=TUXDIR
TUXCONFIG=APPDIR\tuxconfig
FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
LANG=C

```

7. TUXEDO용 TLOG 디바이스를 작성하십시오.

이를 수행하려면 tadmin -c를 호출한 후 다음 명령을 입력하십시오.

```
crdl -z APPDIR\TLOG
```

8. 현재 디렉토리를 APPDIR로 설정하고 샘플 makefile amqstxmn.mak을(를) 외부 프로젝트 makefile로 호출하십시오. 예를 들어 Microsoft Visual C++에서는 다음 명령을 실행하십시오.

```
msvc amqstxmn.mak
```

모든 샘플 프로그램을 빌드하려면 **빌드**를 선택하십시오.

Windows Windows용 서버 환경 빌드(64비트)
IBM MQ for Windows용 서버 환경을 빌드하는 방법입니다(64비트).

이 태스크 정보

참고: 다음에서 *VARIABLES*로 식별되는 필드를 디렉토리 경로로 변경하십시오.

표 169. 디렉토리 경로로 변경할 필드	
필드	디렉토리 경로
<i>MQMDIR</i>	IBM MQ이(가) 설치될 때 지정된 디렉토리 경로(예: g:\Program Files\IBM\MQ).
<i>TUXDIR</i>	TUXEDO가 설치될 때 지정된 디렉토리 경로(예: f:\tuxedo).
<i>APPDIR</i>	샘플 애플리케이션에 사용할 디렉토리 경로(예: f:\tuxedo\apps\mqapp).

```
*RESOURCES
IPCKEY      99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS  20
MAXSERVICES  50
MASTER      SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
             TUXDIR="f:\tuxedo"
             APPDIR="f:\tuxedo\apps\mqapp;g:\Program Files\IBM\WebSphere MQ\bin"
             ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
             TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
             ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
             TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
             TLOGNAME=TLOG
             TYPE="i386NT"
             UID=0
             GID=0

*GROUPS
GROUP1
      LMID=SITE1 GRPNO=1
      TMSNAME=MQXA
      OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1  SRVGRP=GROUP1 SRVID=1
MQSERV2  SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

그림 152. IBM MQ for Windows에 대한 *ubbstxcn.cfg* 파일의 예

참고: 시스템 이름 *MachineName* 및 디렉토리 경로를 사용자 설치에 일치하도록 변경하십시오. 또한 큐 관리자 이름 *MYQUEUEMANAGER*를 연결하려는 큐 관리자의 이름으로 변경하십시오.

IBM MQ for Windows에 대한 샘플 *ubbconfig* 파일은 [1038 페이지의 그림 152](#)에 나열됩니다. 이는 IBM MQ 샘플 디렉토리에 *ubbstxcn.cfg*(으)로 제공됩니다.

IBM MQ for Windows에 제공된 샘플 makefile(1039 페이지의 [그림 153](#) 참조)은 ubbstxmn.mak(이)라고 하며 IBM MQ 샘플 디렉토리에 보유됩니다.

```
TUXDIR = f:\tuxedo
MQMDIR = g:\Program Files\IBM\WebSphere MQ
APPDIR = f:\tuxedo\apps\mqapp
MQMLIB = $(MQMDIR)\tools\lib64
MQMINC = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
$(TUXDIR)\bin\mkfldhdr -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
$(TUXDIR)\bin\viewc -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
$(TUXDIR)\bin\buildtms -o MQXA -r MQSERIES_XA_RMI
$(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
-f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
-r MQSERIES_XA_RMI \
-s MPUT1:MPUT -s MGET1:MGET
$(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
-f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
-r MQSERIES_XA_RMI \
-s MPUT2:MPUT -s MGET2:MGET
$(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
-f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
$(TUXDIR)\bin\buildclient -o dogets -f $(MQMSAMP)\amqstxgx.c \
-f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
$(TUXDIR)\bin\tmloadcf -y $(APPDIR)\ubbstxcn.cfg
```

그림 153. IBM MQ for Windows용 샘플 TUXEDO make 파일

서버 환경 및 샘플을 빌드하려면 다음 단계를 완료하십시오.

프로시저

1. 샘플 애플리케이션을 빌드할 애플리케이션 디렉토리를 작성하십시오. 예를 들어,

```
f:\tuxedo\apps\mqapp
```

2. IBM MQ 샘플 디렉토리의 다음 샘플 파일을 애플리케이션 디렉토리에 복사하십시오.

- amqstxmn.mak
- amqstxen.env
- ubbstxcn.cfg

3. 이러한 각 파일을 편집하여 설치에 사용된 디렉토리 경로 및 디렉토리 이름을 설정하십시오.
4. 연결하려는 시스템 이름 및 큐 관리자의 세부사항을 추가하려면 ubbstxcn.cfg(를) 편집하십시오([1038 페이지의 그림 152](#) 참조).
5. TUXEDO 파일 TUXDIRudataobj\rm에 다음 행을 추가하십시오.

```
MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib64\mqmxa64.lib
MQMDIR\tools\lib64\mqm.lib
```

새 항목은 파일에서 한 행에 있어야 합니다.

6. 다음 환경 변수를 설정하십시오.

```
TUXDIR=TUXDIR
TUXCONFIG=APPDIR\tuxconfig
FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
LANG=C
```

7. TUXEDO용 TLOG 디바이스를 작성하십시오. 이를 수행하려면, tadmin -c를 호출하고 명령을 입력하십시오.

```
crdl -z APPDIR\TLOG
```

8. 현재 디렉토리를 *APPDIR*로 설정하고 샘플 makefile *amqstxm.n.mak*을(를) 외부 프로젝트 makefile로 호출하십시오. 예를 들어 Microsoft Visual C++에서는 다음 명령을 실행하십시오.

```
msvc amqstxm.n.mak
```

모든 샘플 프로그램을 빌드하려면 **빌드**를 선택하십시오.

ALW TUXEDO용 샘플 서버 프로그램

샘플 서버 프로그램(*amqstxsx*)은 *Put(amqstxpx.c)* 및 *Get(amqstxgx.c)* 샘플 프로그램을 사용하여 실행하도록 설계되었습니다. TUXEDO가 시작되면 샘플 서버 프로그램이 자동으로 실행됩니다.

참고: TUXEDO를 시작하기 전에 큐 관리자를 시작해야 합니다.

샘플 서버는 MPUT1 및 MGET1이라는 두 가지 TUXEDO 서비스를 제공합니다.

- MPUT1 서비스는 PUT 샘플에 의해 구동되며 TUXEDO에서 제어하는 작업 단위에 메시지를 넣기 위해 동기점에서 MQPUT1을 사용합니다. PUT 샘플에서 제공하는 매개변수 *QName* 및 메시지 텍스트를 사용합니다.
- MGET1 서비스는 메시지를 가져올 때마다 큐를 열고 닫습니다. GET 샘플에서 제공하는 매개변수 *QName* 및 메시지 텍스트를 사용합니다.

오류 메시지, 이유 코드 및 상태 메시지가 TUXEDO 로그 파일에 작성됩니다.

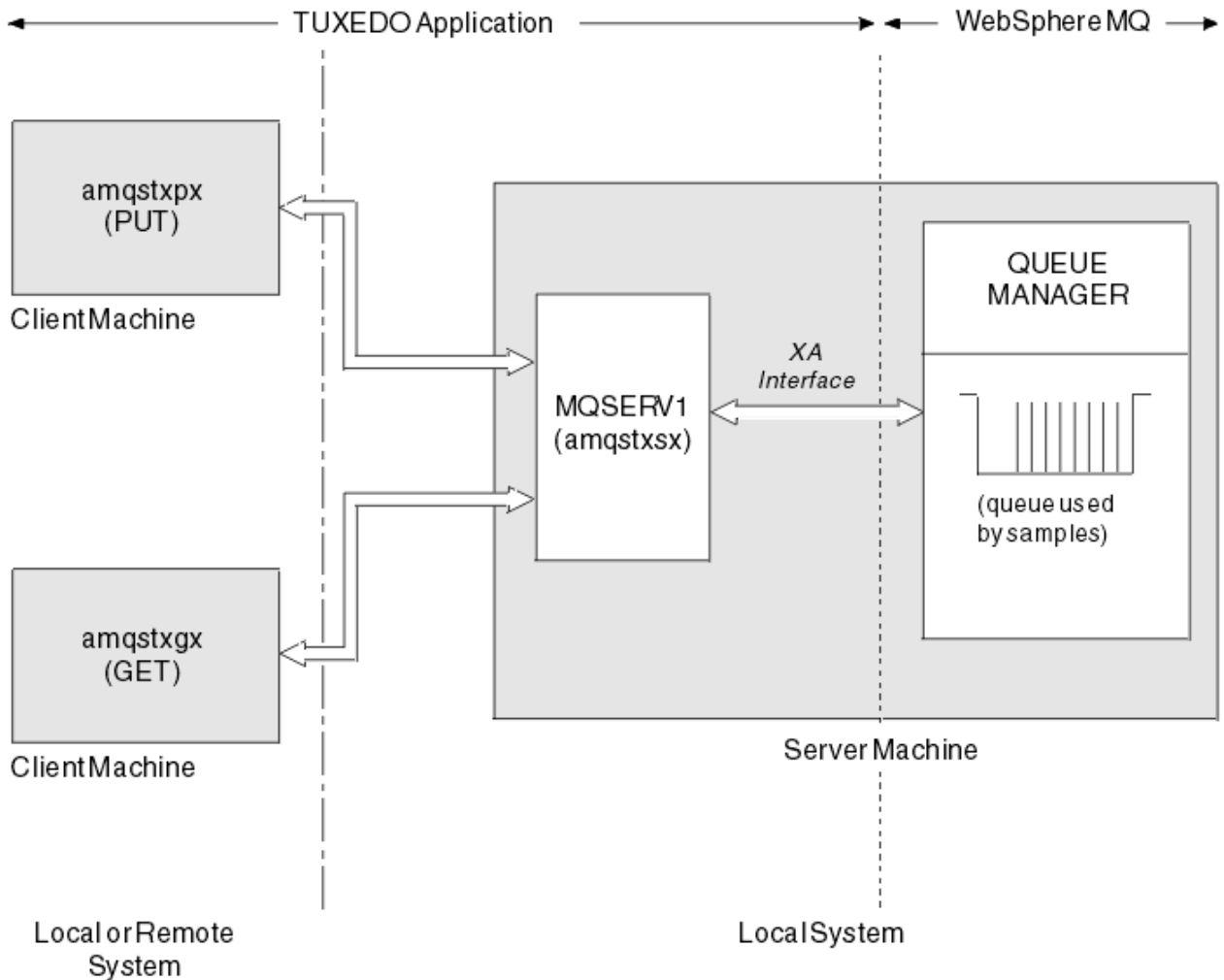


그림 154. TUXEDO 샘플이 함께 작동하는 방법

ALW TUXEDO용 Put 샘플 프로그램

이 샘플을 사용하면 큐에 메시지를 일괄처리로 여러 번 넣을 수 있으며, TUXEDO를 자원 관리자로 사용하여 동기점을 보여줍니다.

샘플 서버 프로그램 amqstxsx는 put 샘플이 성공하기 위해 실행 중이어야 합니다. 서버 샘플 프로그램은 큐 관리자에 연결하고 XA 인터페이스를 사용합니다. 샘플을 실행하려면 다음을 입력하십시오.

- `doputs -n queueName -b batchSize -c tranccount -t message`

예를 들면, 다음과 같습니다.

- `doputs -n myqueue -b 5 -c 6 -t "Hello World"`

이 명령은 다섯 개의 메시지가 각각 포함되어 있는 6개 배치에서 myqueue라는 큐에 30개의 메시지를 넣습니다. 문제가 발생하는 경우 메시지 배치를 취소하고 그렇지 않은 경우 메시지를 커밋합니다.

오류 메시지가 TUXEDO 로그 파일 및 stderr에 작성됩니다. 이유 코드는 stderr에 작성됩니다.

TUXEDO용 샘플 가져오기

이 샘플을 사용하면 큐에서 메시지를 일괄처리로 가져올 수 있습니다.

Get 샘플이 성공하기 위해 amqstxsx 샘플 서버 프로그램이 실행 중이어야 합니다. 샘플을 실행하려면 다음 명령을 입력하십시오.

- `dogets -n queueName -b batchSize -c tranccount`

예를 들면, 다음과 같습니다.

- `dogets -n myqueue -b 6 -c 4`

이 명령은 네 개의 메시지가 각각 포함되어 있는 6개 배치의 myqueue라는 큐에서 24개의 메시지를 선택합니다. myqueue에 30개의 메시지를 넣는 Put 예 후에 이를 실행하는 경우 myqueue에 6개의 메시지만 갖습니다. 배치 수와 배치 크기는 메시지를 넣고 가져오기 간에 다를 수 있습니다.

오류 메시지가 TUXEDO 로그 파일 및 stderr에 작성됩니다. 이유 코드는 stderr에 작성됩니다.

Windows에서 SSPI 보안 엑시트 사용

이 주제에서는 Windows 시스템에서 SSPI 채널 엑시트 프로그램을 사용하는 방법에 대해 설명합니다. 제공된 엑시트 코드는 오브젝트 및 소스의 두 가지 형식입니다.

오브젝트 코드

오브젝트 코드 파일은 amqrspin.dll(이)라 부릅니다. 클라이언트 및 서버 모두에서 이는 `MQ_INSTALLATION_PATH/exits/INSTALLATION_NAME` 폴더에 IBM MQ for Windows의 표준 파트로 설치됩니다. 예를 들어, `C:\Program Files\IBM\MQ\exits\installation2`입니다. 표준 사용자 엑시트로 로드됩니다. 제공된 보안 채널 엑시트를 실행하고 채널 정의에서 인증 서비스를 사용할 수 있습니다.

이를 수행하려면 다음 중 하나를 지정하십시오.

```
SCYEXIT('amqrspin(SCY_KERBEROS)')
```

```
SCYEXIT('amqrspin(SCY_NTLM)')
```

제한된 채널에 대한 지원을 제공하려면 SVRCONN 채널에 다음을 지정하십시오.

```
SCYDATA('remote_principal_name')
```

여기서 `remote_principal_name`은 `DOMAIN\user` 양식입니다. 보안 채널은 원격 프린시펄의 이름이 `remote_principal_name`과 일치하는 경우에만 설정됩니다.

Kerberos 보안 도메인 내에서 조작하는 시스템 사이에서 제공된 채널 엑시트 프로그램을 사용하려면 큐 관리자의 **servicePrincipalName**을 작성하십시오.

소스 코드

엑시트 소스 코드 파일은 amqsspin.c(이)라 부릅니다. 이 파일은 `C:\Program Files\IBM\MQ\Tools\c\Samples`에 있습니다.

소스 코드를 수정하는 경우 수정된 소스를 재컴파일해야 합니다.

컴파일 시간에 SSPI 헤더에 액세스해야 하고 링크 시간에는 권장되는 연관된 라이브러리와 함께 SSPI 보안 라이브러리에 액세스해야 하는 것을 제외하고는 관련된 플랫폼의 다른 채널 엑시트와 같은 방법으로 소스를 컴파일 및 링크합니다.

다음 명령을 실행하기 전에 `c1.exe` 및 Visual C++ 라이브러리와 `include` 폴더가 경로에서 사용 가능한지 확인하십시오. 예를 들면, 다음과 같습니다.

```
c1 /VERBOSE /LD /MT /Ipath_to_Microsoft_platform_SDK\include
/Ipath_to_IBM_MQ\tools\c\include amqssp.c /DSECURITY_WIN32
-link /DLL /EXPORT:SCY_KERBEROS /EXPORT:SCY_NTLM STACK:8192
```

참고: 소스 코드는 추적 또는 오류 핸들링에 대한 프로비저닝을 포함하지 않습니다. 소스 코드를 수정하고 사용하는 경우, 사용자 전용 추적 및 오류 핸들링 루틴을 추가하십시오.

리모트 큐를 사용하여 샘플 실행

연결된 큐 관리자의 샘플을 실행하여 리모트 큐잉을 보여줄 수 있습니다.

`amqscos0.tst` 프로그램은 `OTHER`이라는 이름의 리모트 큐 관리자를 사용하는 리모트 큐의 로컬 정의(`SYSTEM.SAMPLE.REMOTE`)를 제공합니다. 이 샘플 정의를 사용하려면, `OTHER`을 사용하려는 두 번째 큐 관리자의 이름으로 변경하십시오. 또한 두 큐 관리자 간에 메시지 채널을 설정해야 합니다. 이를 수행하는 방법에 대한 정보는 [채널 정의](#)의 내용을 참조하십시오.

요청 샘플 프로그램은 로컬 큐 관리자 이름을 송신하는 메시지의 `ReplyToQMGr` 필드에 넣습니다. 조회 및 설정 샘플은 처리하는 요청 메시지의 `ReplyToQ` 및 `ReplyToQMGr` 필드에 이름 지정된 큐 및 큐 관리자에 응답 메시지를 보냅니다.

관련 참조

1078 페이지의 『[원격 큐를 사용하여 RPG 샘플 실행 IBM i](#)』

연결된 메시지 큐 관리자에서 샘플을 실행하여 리모트 큐잉을 보여줄 수 있습니다.

클러스터 큐 모니터링 샘플 프로그램(AMQSCLM)

이 샘플은 내장 IBM MQ 클러스터 워크로드 밸런싱 기능을 사용하여 이용하는 애플리케이션이 있는 첨부된 큐의 인스턴스에 메시지를 전달합니다. 이러한 자동 전달을 통해 이용하는 애플리케이션이 첨부되지 않은 클러스터 큐의 인스턴스에 메시지가 쌓이지 않도록 합니다.

개요

다른 큐 관리자의 동일한 큐에 대해 둘 이상의 정의가 있는 클러스터를 설정할 수 있습니다. 이 구성은 항상된 가용성과 워크로드 밸런스의 이점을 제공합니다. 그러나 첨부된 애플리케이션의 상태에 따라 클러스터에서 메시지의 분배를 동적으로 수정하는 IBM MQ에 내장된 기능은 없습니다. 따라서 메시지를 처리하려면 이용하는 애플리케이션은 항상 큐의 모든 인스턴스에 첨부되어야 합니다.

클러스터 큐 모니터링 샘플 프로그램은 첨부된 애플리케이션의 상태를 모니터링합니다. 프로그램은 동적으로 내장 워크로드 밸런스 구성을 조정하여 이용 중인 애플리케이션이 첨부된 클러스터된 큐의 인스턴스로 메시지를 전달합니다. 특정 상황에서 이 프로그램은 이용 중인 애플리케이션이 큐의 모든 인스턴스에 항상 연결해야 하는 요구를 완화하는 데 사용될 수 있습니다. 또한 이용 중인 애플리케이션을 첨부하지 않은 큐의 인스턴스에서 큐된 메시지를 다시 송신합니다. 메시지를 다시 송신하면 임시로 종료된 이용 중인 애플리케이션 근처로 메시지를 라우트할 수 있습니다.

프로그램은 자주 첨부 및 첨부 해제되는 애플리케이션이 아닌, 이용 중인 애플리케이션이 장기 실행 애플리케이션인 경우에 사용하도록 설계되었습니다.

클러스터 큐 모니터링 샘플 프로그램은 C 샘플 파일 `amqsc1ma.c`의 컴파일된 실행 가능 프로그램입니다.

클러스터 및 워크로드에 대한 추가 정보는 [워크로드 관리에 대해 클러스터 사용](#)에서 찾을 수 있습니다.

AMQSCLM: 샘플 사용에 대한 설계 및 계획

클러스터 큐 모니터링 샘플 프로그램의 작동 방법, 샘플 프로그램을 실행하기 위해 시스템을 설정할 때 고려할 사항 및 샘플 소스 코드에 작성할 수 있는 수정사항에 대한 정보입니다.

설계

클러스터 큐 모니터링 샘플 프로그램은 이용하는 애플리케이션이 연결되어 있는 로컬 클러스터 큐를 모니터링합니다. 프로그램은 사용자가 지정한 큐를 모니터링합니다. 큐의 이름은 APP.TEST01과 같이 특정하거나 일반적일 수 있습니다. 일반 이름은 PCF(프로그래밍가능한 명령 형식)를 준수하는 형식이어야 합니다. 일반 이름의 예는 APP.TEST* 또는 APP*입니다.

모니터링될 로컬 큐의 인스턴스를 소유하는 클러스터에서 각 큐 관리자는 연결할 클러스터 큐 모니터링 샘플 프로그램의 인스턴스가 필요합니다.

동적 메시지 라우팅

클러스터 큐 모니터링 샘플 프로그램은 큐의 **IPPROCS**(입력 프로세스 수를 위해 열린) 값을 사용하여 큐에 이용자가 있는지 판별합니다. 0보다 큰 값은 큐에 적어도 하나 이상의 이용 애플리케이션이 연결되어 있다는 것을 표시합니다. 이러한 큐는 활성입니다. 0 값은 큐에 연결되어 있는 이용 프로그램이 없음을 나타냅니다. 이러한 큐는 비활성입니다.

클러스터의 여러 인스턴스로 클러스터된 큐의 경우, IBM MQ는 어떤 인스턴스로 메시지를 송신하는지 판별하기 위해 각 큐 인스턴스의 **CLWLPRTY** 클러스터 워크로드 우선순위 특성을 사용합니다. IBM MQ는 최상의 **CLWLPRTY** 값을 사용하여 사용 가능한 큐 인스턴스에 메시지를 송신합니다.

클러스터 큐 모니터링 샘플 프로그램은 로컬 **CLWLPRTY** 값을 1로 설정하여 클러스터 큐를 활성화합니다. 프로그램은 **CLWLPRTY** 값을 0으로 설정하여 클러스터 큐를 비활성화합니다.

IBM MQ 클러스터링 기술은 클러스터된 큐의 업데이트된 **CLWLPRTY** 특성을 클러스터에 있는 모든 관련 큐 관리자에 전파합니다. 예:

- 메시지를 큐에 넣는 애플리케이션이 연결되어 있는 큐 관리자.
- 동일한 클러스터에서 동일한 이름의 로컬 큐를 소유하는 큐 관리자.

클러스터의 전체 저장소 큐 관리자를 사용하여 전파됩니다. 클러스터 큐에 대한 새 메시지는 클러스터 안에 있는 최고의 **CLWLPRTY** 값을 가진 인스턴스로 전달됩니다.

큐 대기 메시지 전송

CLWLPRTY 값을 동적으로 수정하면 새 메시지의 라우팅에 영향을 미칩니다. 이 동적 수정은 연결된 사용자 없이 큐 인스턴스의 큐에 이미 넣어진 메시지 또는 수정된 **CLWLPRTY** 값이 클러스터 전체에서 전파되기 전에 워크로드 밸런싱 메커니즘을 통과한 메시지에는 영향을 주지 않습니다. 결과적으로 비활성 큐에 메시지가 남게 되고 이용 애플리케이션에서 처리하지 않습니다. 이를 해결하기 위해, 클러스터 큐 모니터링 샘플 프로그램은 이용자 없는 로컬 큐에서 메시지를 가져오고 이용자가 연결되어 있는 동일한 큐의 리모트 인스턴스로 이러한 메시지를 송신할 수 있습니다.

클러스터 큐 모니터링 샘플 프로그램은 비활성 로컬 큐의 메시지를 메시지 가져오기(**MQGET** 사용) 및 클러스터된 동일한 큐에 메시지 넣기(**MQPUT** 사용)에 의해 하나 이상의 활성 리모트 큐로 전송합니다. 이 전송으로 인해 IBM MQ 클러스터 워크로드 관리에서 로컬 큐 인스턴스의 값보다 더 높은 **CLWLPRTY** 값을 기반으로 다른 대상 인스턴스를 선택하게 됩니다. 메시지 지속성 및 컨텍스트는 메시지가 전송되는 동안 보존됩니다. 메시지 순서 및 바인딩 옵션은 보존되지 않습니다.

계획

이용 애플리케이션의 연결성에 변경사항이 발생하는 경우 클러스터 큐 모니터링 샘플 프로그램은 클러스터 구성을 수정합니다. 수정사항은 클러스터 큐 모니터링 샘플 프로그램이 큐를 모니터링하는 큐 관리자로부터 클러스터에 있는 전체 저장소 큐 관리자로 전송됩니다. 전체 저장소 큐 관리자가 구성 업데이트를 처리하여 클러스터에 있는 관련된 모든 큐 관리자에 이를 재전송합니다. 관련 큐 관리자에는 동일한 이름의 클러스터된 큐를 소유하는 큐 관리자(클러스터 큐 모니터링 샘플 프로그램의 인스턴스가 실행 중인) 및 지난 30일 동안 메시지를 넣기 위해 애플리케이션에서 클러스터 큐를 열었던 큐 관리자가 포함됩니다.

변경사항은 클러스터에서 비동기적으로 처리됩니다. 따라서 각 변경이 있는 후 클러스터에 있는 다른 큐 관리자에는 일정 기간 동안 구성의 다른 보기가 있을 수 있습니다.

클러스터 큐 모니터링 샘플 프로그램은 이용 애플리케이션을 자주 연결 또는 분리하지 않는 시스템에만 적합합니다(예: 장기 실행 이용 애플리케이션). 이용 애플리케이션이 단기간 동안만 연결되어 있는 시스템을 모니터링하는 데 사용되면 구성 업데이트를 분배한 결과로 클러스터의 큐 관리자에 이용자가 연결된 큐의 잘못된 보기가 포함될 수 있는 경우 대기 시간이 발생합니다. 이 대기 시간으로 인해 올바르게 보내지 않은 메시지가 라우팅될 수 있습니다.

여러 큐를 모니터링하는 경우 모든 큐에서 연결된 이용자의 변경 비율이 상대적으로 낮으면 클러스터에서 클러스터 구성 트래픽이 증가할 수 있습니다. 증가된 클러스터 구성 트래픽은 하나 이상의 다음 큐 관리자에서 과도한로드가 발생할 수 있습니다.

- 클러스터 큐 모니터링 샘플 프로그램을 실행 중인 큐 관리자
- 전체 저장소 큐 관리자
- 메시지를 큐에 넣는 애플리케이션이 연결되어 있는 큐 관리자
- 동일한 클러스터에서 동일한 이름의 로컬 큐를 소유하는 큐 관리자

전체 저장소 큐 관리자의 프로세서 사용을 평가해야 합니다. 추가 프로세서 사용은 전체 저장소 큐 `SYSTEM.CLUSTER.COMMAND.QUEUE`에 메시지 트래픽으로 표시됩니다. 메시지가 해당 큐에서 빌드되는 경우 전체 저장소 큐 관리자가 시스템에서 클러스터 구성 변경의 비율을 잘 알 수 없다는 것을 표시합니다.

클러스터 큐 모니터링 샘플 프로그램에서 여러 큐를 모니터링하는 경우 샘플 프로그램 및 큐 관리자가 상당량의 작업을 수행합니다. 연결된 이용자에 대한 변경사항이 없는 경우에도 이 작업이 수행됩니다. 모니터링 주기의 빈도를 줄여 로컬 시스템에서 샘플 프로그램의 프로세서 사용을 감소시키도록 `-i` 인수를 수정할 수 있습니다.

과도한 활동을 감지하는 데 도움을 주기 위해, 클러스터 큐 모니터링 샘플 프로그램은 폴링 간격당 평균 처리 시간, 경과된 처리 시간 및 구성 변경사항 수를 보고합니다. 보고서는 정보 메시지 `CLM0045I`로 30분 마다 또는 600번의 폴링 간격마다 더 빠른 시간 간격으로 전달됩니다.

사용 요구사항을 모니터링하는 클러스터 큐

클러스터 큐 모니터링 샘플 프로그램에는 요구사항 및 제한사항이 있습니다. 사용할 수 있는 방법에서 이러한 일부 제한사항을 변경하기 위해 샘플 소스 코드를 수정할 수 있습니다. 이 섹션에 나열된 예는 작성 가능한 변경사항에 대해 자세히 설명합니다.

- 클러스터 큐 모니터링 샘플 프로그램은 이용 애플리케이션이 연결되거나 연결되지 않는 큐를 모니터링하는 데 사용하도록 설계되었습니다. 자주 연결 및 분리되는 이용 애플리케이션이 시스템에 있는 경우 샘플 프로그램은 전체 클러스터에서 과도한 클러스터 구성 활동을 생성할 수 있습니다. 이는 클러스터에 있는 큐 관리자의 성능에 영향을 미칠 수 있습니다.
- 클러스터 큐 모니터링 샘플 프로그램은 기본 IBM MQ 시스템 및 클러스터 기술에 따라 달라집니다. 모니터링되는 큐의 수, 모니터링 빈도 및 각 큐의 상태를 변경하는 빈도는 전반적인 시스템의 로드와 영향을 줍니다. 모니터링된 큐 및 모니터링의 폴 간격을 선택할 때 이러한 요소를 고려해야 합니다.
- 클러스터 큐 모니터링 샘플 프로그램의 인스턴스는 모니터링할 큐의 인스턴스를 소유하는 클러스터의 모든 큐 관리자에 연결되어야 합니다. 큐를 소유하지 않는 클러스터의 큐 관리자에 샘플 프로그램을 연결할 필요는 없습니다.
- 필요한 모든 IBM MQ 자원에 액세스하려면 적절한 권한으로 클러스터 큐 모니터링 샘플 프로그램을 실행해야 합니다. 예:
 - 연결할 큐 관리자
 - `SYSTEM.ADMIN.COMMAND.QUEUE`
 - 메시지 전송이 수행되는 경우 모니터링할 모든 큐
- 클러스터 큐 모니터링 샘플 프로그램을 연결하여 각 큐 관리자에 대해 명령 서버를 실행해야 합니다.
- 클러스터 큐 모니터링 샘플 프로그램의 각 인스턴스는 연결되어 있는 큐 관리자의 로컬(클러스터되지 않은) 큐를 독점적으로 사용해야 합니다. 이 로컬 큐는 샘플 프로그램을 제어하는 데 사용되며 큐 관리자의 명령 서버에 대해 수행된 조회로부터 응답 메시지를 수신합니다.
- 클러스터 큐 모니터링 샘플 프로그램의 단일 인스턴스에서 모니터링할 모든 큐는 동일한 클러스터에 있어야 합니다. 큐 관리자에 모니터링이 필요한 여러 클러스터의 큐가 있는 경우 동일한 프로그램의 여러 인스턴스가 필요합니다. 각 인스턴스는 제어 및 응답 메시지에 대한 로컬 큐가 필요합니다.

- 모니터링될 모든 큐는 단일 클러스터에 있어야 합니다. 클러스터 이름 목록을 사용하도록 구성된 큐는 모니터하지 않습니다.
- 비활성 큐로부터 메시지 전송을 사용하는 것은 선택사항입니다. 클러스터 큐 모니터링 샘플 프로그램의 인스턴스에서 모니터링하는 모든 큐에 적용됩니다. 모니터되고 있는 큐의 서브세트만 메시지 전송을 사용해야 하는 경우 클러스터 큐 모니터링 샘플 프로그램의 두 개 인스턴스가 필요합니다. 하나의 샘플 프로그램은 메시지 전송을 사용하고 다른 프로그램은 메시지 전송을 사용하지 않습니다. 샘플 프로그램의 각 인스턴스는 제어 및 응답 메시지에 대한 로컬 큐가 필요합니다.
- IBM MQ 클러스터 워크로드 밸런싱은 기본적으로, 넣기 애플리케이션이 연결되어 있는 동일한 큐 관리자에 상주하는 클러스터 큐의 인스턴스로 메시지를 송신합니다. 다음 환경에서 로컬 큐가 비활성 상태인 동안은 사용 안함으로 설정해야 합니다.
 - 넣기 애플리케이션은 모니터링되는 비활성 큐의 인스턴스를 소유하는 큐 관리자에 연결됩니다.
 - 큐 처리된 메시지가 비활성 큐에서 활성 큐로 전송 중입니다.

CLWLUSEQ 값을 ANY로 설정하여 큐의 로컬 워크로드 밸런싱 환경 설정을 사용 안함으로 정적으로 설정할 수 있습니다. 이 구성에서, 로컬 큐에 넣어진 메시지는 로컬 이용 애플리케이션이 있더라도 워크로드 밸런싱을 위해 로컬 및 리모트 큐 인스턴스로 분배됩니다. 또는 클러스터 큐 모니터링 샘플 프로그램을 큐에 연결된 이용자가 없는 동안 **CLWLUSEQ** 값을 ANY로 임시로 설정하도록 구성할 수 있으며 그 결과 큐가 활성인 동안은 큐의 로컬 인스턴스로만 로컬 메시지가 이동됩니다.
- IBM MQ 시스템 및 애플리케이션은 모니터링할 큐 또는 사용되고 있는 채널에 **CLWLPRTY**를 사용하지 않아야 합니다. 그렇지 않으면, **CLWLPRTY** 큐 속성에 대한 클러스터 큐 모니터링 샘플 프로그램의 조치에 원하지 않은 영향이 있을 수 있습니다.
- 클러스터 큐 모니터링 샘플 프로그램은 일련의 보고서 파일에 런타임 정보를 기록합니다. 이러한 보고서를 저장할 디렉토리가 필요하며 클러스터 큐 모니터링 샘플 프로그램에는 작성할 권한이 있어야 합니다.

AMQSCLM: 샘플 준비 및 실행

클러스터 큐 모니터링 샘플은 큐 관리자에 로컬로 연결되어 실행되거나 채널을 통해 연결된 클라이언트로서 실행될 수 있습니다. 샘플은 큐 관리자가 실행될 때마다 실행되어야 하며 로컬로 실행되는 경우 큐 관리자를 통해 샘플이 자동으로 시작되고 중지되도록 샘플을 큐 관리자 서비스로 구성할 수 있습니다.

시작하기 전에

클러스터 큐 모니터링 샘플을 실행하기 전에 다음 단계를 완료해야 합니다.

1. 샘플의 내부 사용을 위해 각 큐 관리자에 작업 중인 큐를 작성하십시오.

샘플의 각 인스턴스는 독점 내부 사용을 위한 로컬 비클러스터가 필요합니다. 큐의 이름을 선택할 수 있습니다. 예는 AMQSCLM.CONTROL.QUEUE 이름을 사용합니다. 예를 들어 Windows에서는 다음 **MQSC** 명령을 사용하여 이 큐를 작성할 수 있습니다.

```
DEFINE QLOCAL (AMQSCLM.CONTROL.QUEUE)
```

MAXDEPTH 및 **MAXMSGL**의 값을 기본값으로 남겨둘 수 있습니다.

2. 오류 및 정보 메시지 로그를 위한 디렉토리를 작성하십시오.

샘플은 보고서 파일에 진단 메시지를 작성합니다. 파일을 저장할 디렉토리를 선택해야 합니다. 예를 들어, Windows에서 다음 명령을 사용하여 디렉토리를 작성할 수 있습니다.

```
mkdir C:\AMQSCLM\rpts
```

샘플에서 작성한 보고서 파일에는 다음 이름 지정 규칙이 있습니다.

```
QmgrName.ClusterName.RPTOn.LOG
```

3. (선택사항) IBM MQ 서비스로서 클러스터 큐 모니터링 샘플을 정의하십시오.

큐를 모니터링하려면 샘플이 항상 실행 중이어야 합니다. 클러스터 큐 모니터링 샘플이 항상 실행 중인지 확인하기 위해 샘플을 큐 관리자 서비스로 정의할 수 있습니다. 샘플을 서비스로서 정의하는 것은 큐 관리자를 시

작하는 경우 AMQSCLM이 시작되는 것을 의미합니다. 다음 예를 사용하여 클러스터 큐 모니터링 샘플을 IBM MQ 서비스로서 정의할 수 있습니다.

```
define service(AMQSCLM) +
  descr('Active Cluster Queue Message Distribution Monitor - AMQSCLM') +
  control(qmgr) +
  servtype(server) +
  startcmd('MQ_INSTALLATION_PATH\tools\c\samples\Bin\AMQSCLM.exe') +
  startarg('-m +QMNAME+ -c CLUSTER1 -q ABC* -r AMQSCLM.CONTROL.QUEUE -l
c:\AMQSCLM\rpts') +
  stdout('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stdout.log') +
  stderr('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stderr.log')
```

정의	설명
service	서비스 이름을 지정합니다. 서비스 이름을 선택할 수 있습니다.
descr	서비스의 텍스트 설명을 지정합니다.
control	큐 관리자로서 동시에 서비스를 시작하고 중지함을 나타냅니다.
servtype	이 큐 관리자에 대해 단 하나의 인스턴스를 의미하는 서버 서비스 오브젝트를 한 번에 실행할 수 있다는 것을 표시합니다.
startcmd	프로그램의 위치와 이름을 지정합니다.
startarg	샘플의 인수를 지정합니다. +QMNAME+의 사용에 유의하십시오. 큐 관리자의 이름은 자동으로 대체됩니다.
stdout	표준 출력이 경로 재지정되는 완전한 파일 이름. 샘플은 샘플이 종료되었음을 확인하는 메시지만 이 파일에 기록합니다. 샘플 종료 프로세스의 이전 단계에서 표준 오류 파일이 이미 닫혔기 때문에 샘플이 이를 수행합니다.
stderr	표준 오류 출력이 경로 재지정되는 완전한 파일 이름. 샘플은 샘플 종료 전에 오류 메시지를 표준 오류 파일에 기록합니다.

이 태스크 정보

이 태스크는 다른 방식으로 클러스터 큐 모니터링 샘플을 시작하고 중지할 수 있습니다. 또한 모니터링되는 큐에 대한 통계 정보를 포함하는 보고서 파일을 생성하는 모드에서 샘플을 실행할 수 있습니다.

다음 명령을 사용하여 샘플 프로그램을 실행할 수 있습니다.

```
AMQSCLM -m QMgrName -c ClusterName (-q QNameMask| -f QListFile) -r MonitorQName
[-i ReportDir] [-t] [-u ActiveVal] [-i Interval] [-d] [-s] [-v]
```

표는 클러스터 큐 모니터링 샘플에 각 샘플에 대한 추가 정보와 함께 사용할 수 있는 인수를 나열합니다.

인수	변수	추가 정보
-m	QMGrName	모니터할 큐 관리자.
-c	ClusterName	모니터할 큐를 포함하는 클러스터.
-q	QNameMask	모니터할 큐. 후미 문자 *는 0개 또는 더 많은 후미 문자와 일치하는 이름의 모든 큐를 모니터링합니다.
-f	QListFile	모니터할 큐 이름 또는 큐 이름 마스크의 목록을 포함하는 파일의 전체 경로 및 파일 이름. 파일은 행당 하나의 큐 이름/마스크를 포함해야 합니다. -q 또는 -f 를 지정할 수 있지만 둘 다를 지정할 수는 없습니다.
-r	MonitorQName	샘플에서 독점적으로 사용하는 로컬 큐.
-l	ReportDir	로깅된 정보 메시지를 다음의 세트에 저장할 디렉토리 경로입니다. 램핑 ⁹ report files.
-t		(선택사항) 비활성 로컬 큐에서 활성 큐로 큐된 메시지를 전송할 수 있습니다. 사용으로 설정되지 않은 경우 클러스터에 들어오는 새 메시지만 큐의 활성 인스턴스에 동적으로 라우팅됩니다.
-u	ActiveVal	(선택사항) 모니터링되는 큐 인스턴스의 CLWLUSEQ 특성이 비활성 상태인 경우 ANY로 자동 전환되고, 활성인 경우 ActiveVal 값으로 전환됩니다. ActiveVal 은 LOCAL 또는 QMGR일 수 있습니다. 넣기 애플리케이션이 동일한 큐 관리자에 연결되어 있는 시스템 또는 메시지 전송이 사용으로 설정된 시스템에서 이 인수가 설정되지 않은 경우, 모니터링되는 큐는 CLWLUSEQ 값이 ANY이거나, ANY 값을 가진 큐 관리자의 경우 QMGR이어야 합니다.
-i	Interval	(선택사항) 모니터가 큐를 확인하는 시간 간격(초). 기본값은 300초(5분)입니다.
-d		(선택사항) 추가 진단 출력을 사용합니다. 처음에 시스템을 구성할 때, 또는 샘플 코드에 대한 작업을 수행할 때 디버그 출력이 유용할 수 있습니다.
-s		(선택사항) 간격당 최소 통계 출력을 사용합니다.
-v		(선택사항) 보고서 파일 외에, standard out에 보고서 정보를 로깅합니다.

인수 목록 예:

```
-m QMGR1 -c CLUS1 -f c:\QList.txt -r CLMQ -l c:\amqscml\rpts -s
-m QMGR2 -c CLUS1 -q ABC* -r CLMQ -l c:\amqscml\rpts -i 600
-m QMGR1 -c CLUSDEV -q QUEUE.* -r CLMQ -l c:\amqscml\rpts -t -u QMGR -d
```

예 큐 목록 파일:

```
Q1
QUEUE.*
ABC
ABD
```

프로시저

1. 클러스터 큐 모니터링 샘플을 시작하십시오. 다음 방법 중 하나로 샘플을 시작할 수 있습니다.

- 적절한 사용자 권한으로 명령 프롬프트를 사용하십시오.
- 샘플이 서비스로 IBM MQ 구성된 경우 MQSC **START SERVICE** 명령을 사용하십시오.

인수 목록은 두 경우 모두 동일합니다.

⁹ 각 큐 관리자와 큐 조합에 대해 가득 차면 겹쳐쓰는 고정 크기 로그 파일이 생성됩니다. 로거는 항상 동일한 파일에 기록하며 이전 두 개 버전의 파일을 보유합니다.

샘플은 프로그램이 초기화된 후 10초 동안 큐 모니터링을 시작하지 않습니다. 이 지연을 통해 이용 애플리케이션은 모니터링되는 큐에 먼저 연결하여 큐의 활성 단계에 불필요한 변경을 막을 수 있습니다.

- 클러스터 큐 모니터링 샘플을 중지하십시오. 샘플은 큐 관리자가 중지되었거나, 중지 중이거나, 정지 중인 경우 또는 큐 관리자에 대한 연결이 끊어진 경우 자동으로 중지됩니다. 큐 관리자를 종료하지 않고 샘플을 중지할 수 있는 방법이 있습니다.
- Get 함수를 사용 안함으로 설정하기 위해 샘플에서 독점적으로 사용한 로컬 큐를 구성하십시오.
 - "STOP CLUSTER MONITOR\0\0\0\0"의 **CorrelId**를 사용하여 샘플에서 독점적으로 사용하는 로컬 큐로 메시지를 송신하십시오.
 - 샘플 프로세스를 종료하십시오. 이에 따라 활성 큐로 전송 중인 비지속 메시지가 유실될 수 있습니다. 또한 종료 후에 샘플에서 사용하는 로컬 큐가 몇 초 동안 열려있을 수 있습니다. 이 상황은 클러스터 큐 모니터링 샘플의 새 인스턴스가 즉시 시작되지 않게 합니다.

샘플이 IBM MQ 서비스로 시작된 경우 **STOP SERVICE**는 효과가 없습니다. 큐 관리자에서 구성된 **STOP SERVICE** 메커니즘으로 설명된 종료 방법 중 하나를 사용할 수 있습니다.

다음에 수행할 작업

샘플 상태 확인

보고를 사용하는 경우 상태에 대해 보고서 파일을 검토할 수 있습니다. 다음 명령을 사용하여 가장 최근 보고서 파일을 검토하십시오.

```
QMgrName.ClusterName.RPT01.LOG
```

이전 보고서 파일을 검토하려면 다음 명령을 사용하십시오.

```
QMgrName.ClusterName.RPT02.LOG
QMgrName.ClusterName.RPT03.LOG
```

보고서 파일이 약 1MB의 최대 크기로 늘어납니다. RPT01 파일이 가득 차면 새 RPT01 파일이 작성됩니다. 이전 RPT01 파일이 RPT02로 이름이 바뀝니다. RPT02는 RPT03으로 이름이 변경됩니다. 이전 RPT03은(는) 버려집니다.

샘플은 다음의 경우에 정보 메시지를 작성합니다.

- 시작 시
- 종료 시
- 큐 **ACTIVE** 또는 **INACTIVE**를 표시하는 경우
- 비활성 큐의 메시지를 활성 인스턴스 또는 인스턴스로 리큐하는 경우

샘플은 주의가 필요한 문제점을 보고하기 위해 **CLMnnnnE** 오류 메시지를 작성합니다.

30분 마다 샘플은 폴링 간격당 평균 처리 시간 및 경과된 처리 시간을 보고합니다. 이 정보는 **CLM0045I** 메시지에 보관됩니다.

통계 메시지가 **-s**를 사용하는 경우 샘플은 각 큐 검사에 대해 다음의 통계 정보를 보고합니다.

- 큐를 처리하는 데 사용된 시간(밀리초 단위)
- 검사한 큐의 수
- 작성된 활성/비활성 변경사항의 수
- 전송된 메시지 수

이 정보는 **CLM0048I** 메시지에 보고됩니다.

보고서 파일은 디버그 모드에서 급속하게 증가하고 빠르게 줄 바꿈기됩니다. 이 상황에서 개별 파일에 대한 1MB 크기 한계가 초과될 수 있습니다.

AMQSCLM: 문제점 해결

다음 섹션은 샘플을 사용하는 동안 발생할 수 있는 시나리오에 대한 정보를 포함합니다. 시나리오의 잠재적 이유에 대한 정보 및 이를 해결하는 방법에 대한 옵션이 제공됩니다.

시나리오: AMQSCLM이 시작되지 않음

잠재적 이유: 올바르지 않은 구문.

조치: 올바른 구문에 대한 표준 오류 출력을 확인하십시오.

잠재적 이유: 큐 관리자가 사용 불가능함.

조치: 메시지 ID CLM0010E에 대한 보고서 파일을 확인하십시오.

잠재적 이유: 보고서 파일을 열거나 작성할 수 없습니다.

조치: 초기화 동안 오류 메시지에 대한 표준 오류 출력을 확인하십시오.

시나리오: AMQSCLM이 큐를 ACTIVE 또는 INACTIVE로 변경하지 않음

잠재적 이유: 큐가 모니터할 큐 목록에 없음

조치: -q 및 -f 매개변수 값을 확인하십시오.

잠재적 이유: 큐가 올바른 클러스터에 있는 로컬 큐가 아닙니다.

조치: 큐가 로컬이며 올바른 클러스터에 있는지 확인하십시오.

잠재적 이유: AMQSCLM이 이 큐 관리자 및 클러스터에 대해 실행 중이지 않습니다.

조치: 관련 큐 관리자 및 클러스터에 대한 AMQSCLM을 시작하십시오.

잠재적 이유: 이용자가 없기 때문에 큐는 INACTIVE, **CLWLPRTY** =0으로 남습니다. 또는 1명 이상의 이용자가 있으므로 ACTIVE **CLWLPRTY** >=1로 남습니다.

조치: 이용하는 애플리케이션이 큐에 연결되었는지 확인하십시오.

잠재적 이유: 큐 관리자의 명령 서버가 실행 중이지 않습니다.

조치: 오류에 대한 보고서 파일을 확인하십시오.

시나리오: 메시지가 INACTIVE 큐로 라우팅되지 않습니다.

잠재적 이유: 비활성 큐를 소유하는 큐 관리자로 직접 메시지가 놓여지고, 큐의 **CLWLUSEQ** 값이 ANY가 아니며 -u 인수가 AMQSCLM에 사용되고 있지 않습니다.

조치: 관련 큐 관리자의 **CLWLUSEQ** 값을 확인하거나 -u 인수가 AMQSCLM에 사용되는지 확인하십시오.

잠재적 이유: 큐 관리자에 활성 큐가 없습니다. 큐의 상태가 활성이 될 때까지 모든 비활성 큐에서 메시지의 워크로드가 고르게 조정됩니다.

조치: 모든 큐 관리자에서 큐 상태를 확인하십시오.

잠재적 이유: 비활성 큐를 소유하는 큐 관리자에 대해 클러스터에 있는 다른 큐 관리자에 메시지를 넣고 업데이트된 **CLWLPRTY** 값 0은 넣기 애플리케이션의 큐 관리자에 전파되지 않습니다.

조치: 모니터링된 큐 관리자와 전체 저장소 큐 관리자 간의 클러스터 채널이 실행 중인지 확인하십시오. 넣기 큐 관리자와 전체 저장소 큐 관리자 간의 채널이 실행 중인지 확인하십시오. 모니터링된 저장소 큐 관리자, 넣기 저장소 큐 관리자 및 전체 저장소 큐 관리자의 오류 로그를 확인하십시오.

잠재적 이유: 리모트 큐 인스턴스가 활성(**CLWLPRTY**=1)이지만, 로컬 큐 관리자로부터의 클러스터 송신자 채널이 실행 중이지 않기 때문에 메시지를 이러한 큐 인스턴스에 라우팅할 수 없습니다.

조치: 로컬 큐 관리자에서 리모트 큐 관리자 또는 큐의 활성 인스턴스가 있는 관리자까지 클러스터 송신자 채널의 상태를 확인하십시오.

시나리오: AMQSCLM이 비활성 큐에서 메시지를 전송하지 않음

잠재적 이유: 메시지 전송이 사용으로 설정되어 있지 않습니다(-t).

조치: 메시지 전송이 사용으로 설정되어 있는지 확인하십시오(-t).

잠재적 이유: 큐가 모니터할 큐 목록에 없습니다.

조치: **-q** 및 **-f** 매개변수 값을 확인하십시오.

잠재적 이유: AMQSCLM이 이 큐 관리자 또는 동일한 큐의 인스턴스를 소유하는 클러스터에 있는 다른 큐 관리자에 대해 실행 중이지 않습니다.

조치: AMQSCLM을 시작하십시오.

잠재적 이유: 큐는 **CLWLUSEQ** = LOCAL 또는 **CLWLUSEQ** = QMGR이 있으며 **-u** 인수가 설정되어 있지 않습니다.

조치: **-u** 매개변수를 설정하거나 큐 또는 큐 관리자의 구성을 ANY로 변경하십시오.

잠재적 이유: 클러스터에 큐의 활성 인스턴스가 없습니다.

조치: **CLWLPRTY** 값이 1 이상인 큐의 인스턴스를 확인하십시오.

잠재적 이유: 리모트 큐 인스턴스는 사용자(**IPPROCS** >=1)가 있지만 AMQSCLM이 이러한 리모트 인스턴스를 모니터링하지 않기 때문에 이러한 큐 관리자에서 비활성(**CLWLPRTY** =0)입니다.

조치: AMQSCLM이 이러한 큐 관리자에서 실행 중인지 및/또는 **-q** 및 **-f** 매개변수 값을 확인하여 모니터링할 큐 목록에 큐가 있는지 확인하십시오.

잠재적 이유: 리모트 큐 인스턴스가 활성(**CLWLPRTY** =1)이지만 로컬 큐 관리자에 비활성으로 표시됩니다(**CLWLPRTY** =0). 이 상황은 업데이트된 **CLWLPRTY** 값이 이 큐 관리자에 전파되지 않기 때문입니다.

조치: 리모트 큐 관리자가 클러스터에서 최소한 하나 이상의 전체 저장소 큐 관리자에 연결되어 있는지 확인하십시오. 전체 저장소 큐 관리자가 올바르게 작동하는지 확인하십시오. 전체 저장소 큐 관리자 및 모니터링된 큐 관리자 간의 채널이 실행 중인지 확인하십시오.

잠재적 이유: 메시지가 커밋되지 않으므로 검색 가능하지 않습니다.

조치: 송신 애플리케이션이 올바르게 기능하고 있는지 확인하십시오.

잠재적 이유: AMQSCLM에는 메시지가 큐에 넣어진 로컬 큐에 대한 액세스가 없습니다.

조치: 큐에 액세스할 수 있는 충분한 권한을 가진 사용자로 AMQSCLM이 실행 중인지 확인하십시오.

잠재적 이유: 큐 관리자의 명령 서버가 실행 중이지 않습니다.

조치: 큐 관리자의 명령 서버를 시작하십시오.

잠재적 이유: AMQSCLM에 오류가 발생했습니다.

조치: 오류에 대한 보고서 파일을 확인하십시오.

잠재적 이유: 리모트 큐 인스턴스가 활성(**CLWLPRTY**=1)이지만 로컬 큐 관리자의 클러스터 송신 채널이 실행 중이지 않기 때문에 메시지를 이러한 큐에 전송할 수 없습니다. 종종 amqsclm 보고서 로그에 CLM0030W 경고가 수반됩니다.

조치: 로컬 큐 관리자에서 리모트 큐 관리자 또는 큐의 활성 인스턴스가 있는 관리자까지 클러스터 송신자 채널의 상태를 확인하십시오.

CEPL(Connection Endpoint Lookup)을 위한 샘플 프로그램

IBM MQ 연결 엔드포인트 검색 샘플은 IBM MQ 사용자에게 Tivoli Directory Server와 같은 LDAP 저장소에서 연결 정의를 검색하는 방법을 제공하는 간단하지만 강력한 엑시트 모듈을 제공합니다.

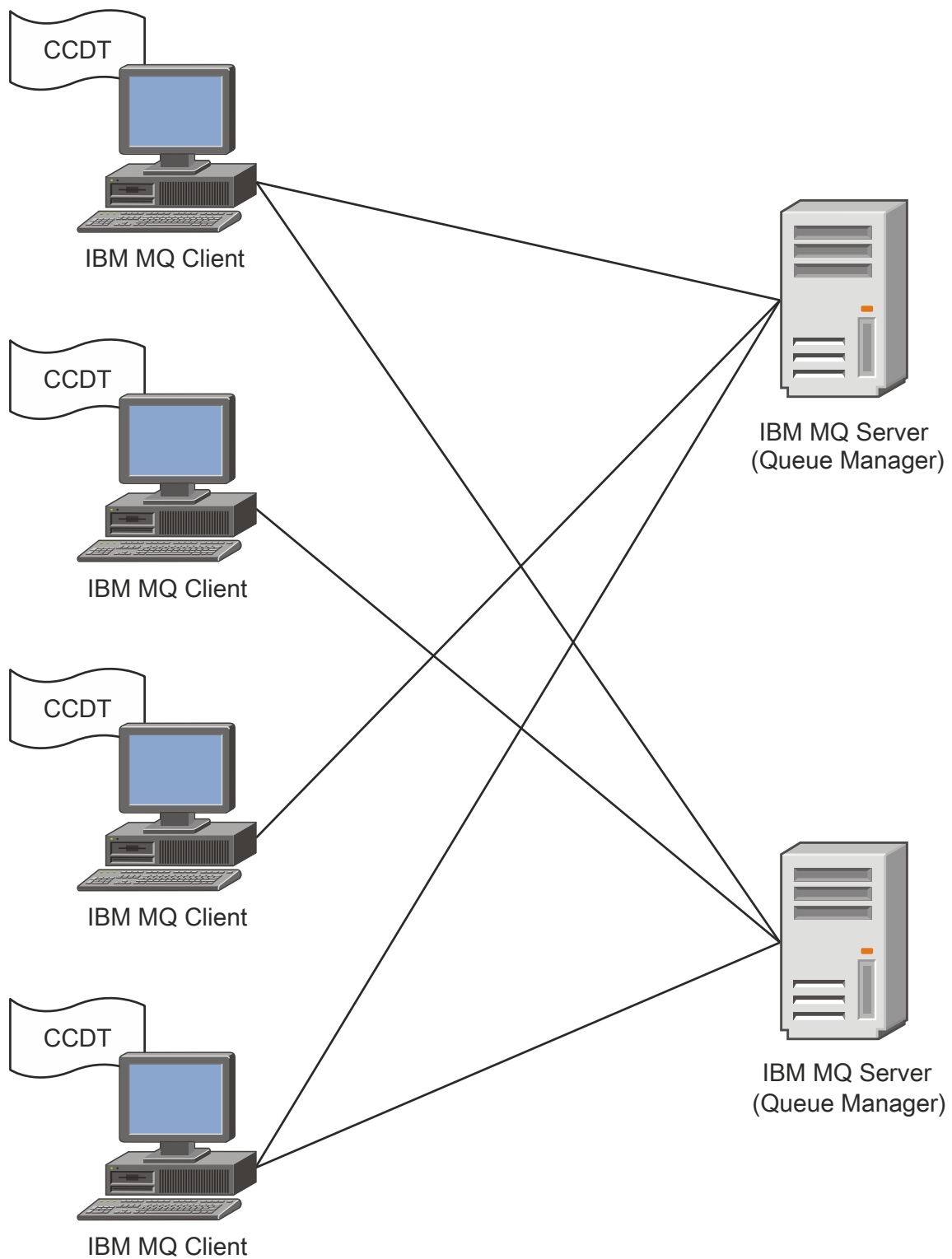
CEPL을 사용하려면 Tivoli Directory Server v6.3 클라이언트를 설치해야 합니다.

이 샘플을 사용하려면 지원되는 플랫폼에서 IBM MQ 관리에 대한 실용적인 지식이 필요합니다.

소개

유지보수 및 관리를 지원하기 위해 클라이언트 연결 정의를 저장하도록 글로벌 저장소 예를 들어, LDAP(Lightweight Directory Access Protocol) 디렉토리를 구성하십시오. 클라이언트 IBM MQ 애플리케이션을 사용하여 클라이언트 연결 정의 테이블(CCDT)을 통해 큐 관리자와 연결을 설정하십시오.

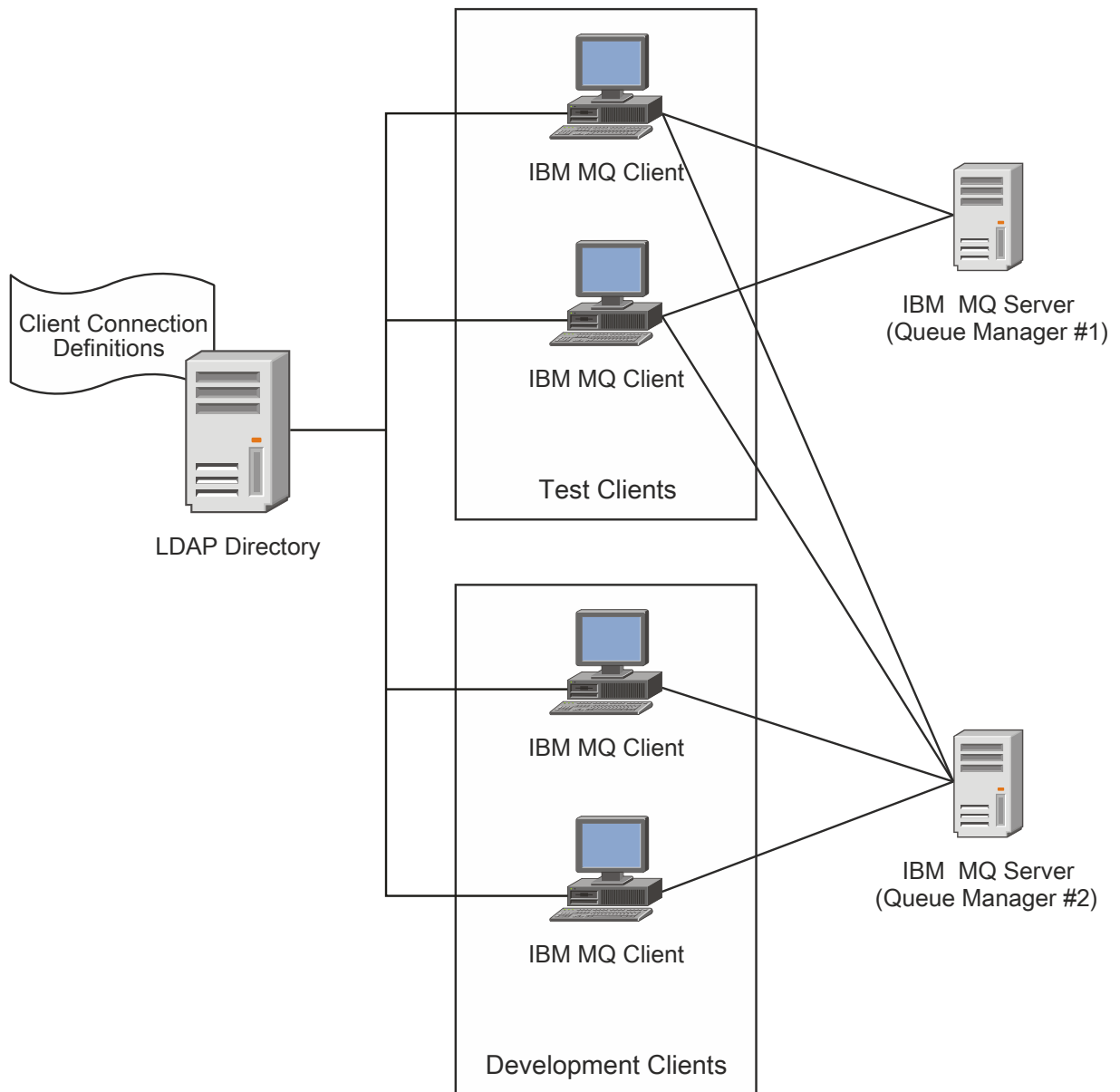
CCDT는 표준 IBM MQ MQSC 관리 인터페이스를 통해 작성됩니다. 정의 내에 포함된 데이터가 큐 관리자로 제한되지 않더라도, 사용자는 클라이언트 연결 정의를 작성하기 위해 큐 관리자에 연결해야 합니다. 생성된 CCDT 파일은 클라이언트 머신과 애플리케이션에 수동으로 배포해야 합니다.



CCDT 파일을 각 IBM MQ 클라이언트에 분배해야 합니다. 수천 개의 클라이언트가 로컬로 또는 글로벌로 존재할 수 있는 경우 유지보수 및 관리가 어렵게 됩니다. 각 클라이언트가 사용 가능한 올바른 클라이언트 정의를 가지고 있는지 확인하도록 돕기 위해 보다 유연한 접근법이 필요합니다.

한 가지 접근 방법은 LDAP(Lightweight Directory Access Protocol) 디렉토리와 같은 글로벌 저장소에서 클라이언트 연결 정의를 저장하는 것입니다. 또한 LDAP 디렉토리는 추가적인 보안, 색인화 및 검색 기능을 제공할 수 있으며 그렇게 함으로써 각 클라이언트에서 관련된 연결 정의에만 액세스할 수 있습니다.

특정 정의만 특정 사용자 그룹에서 사용할 수 있도록 LDAP 디렉토리를 구성할 수 있습니다. 예를 들어, 개발 클라이언트는 큐 관리자 #2만 연결할 수 있는 반면 테스트 클라이언트는 큐 관리자 #1 및 #2 모두 액세스할 수 있습니다.



엑시트 모듈은 채널 정의를 검색하기 위해 LDAP 저장소 예를 들어, IBM Tivoli Directory Server를 검색할 수 있습니다. 이러한 연결 정의를 사용하여 IBM MQ 클라이언트 애플리케이션은 큐 관리자에 대한 연결을 설정할 수 있습니다.

엑시트 모듈은 MQCONN/MQCONNX 호출 중에 LDAP 저장소에서 채널 정의를 획득할 수 있는 사전 연결 엑시트 모듈입니다.

엑시트 모듈 및 스키마를 구현할 수 있는 대상:

- 기술을 기반으로 기존 DDCT 파일을 사용하여 이미 스킴 기반을 빌드했으며 관리 및 분배 비용을 절감하려는 고객.
- 클라이언트 연결 정의를 분배하기 위해 이미 고유의 적절한 기술을 이용하는 기존 고객.
- 현재 클라이언트 연결 솔루션 유형을 이용하고 있지 않으며 IBM MQ에서 제공한 기능을 사용하려는 신규 또는 기존 고객.
- 최신 LDAP 비즈니스 아키텍처를 사용하여 직접 메시징 모델 인라인을 사용하거나 조정하려는 신규 또는 기존 고객.

지원되는 환경

연결 엔드포인트 IBM MQ 조회 샘플을 실행하기 전에 지원되는 운영 체제와 관련 소프트웨어가 설치되어 있는지 확인하십시오. IBM MQ 연결 엔드포인트 검색에 대한 샘플 프로그램은 다음 소프트웨어가 필요합니다.

- IBM WebSphere MQ 7.0 이상
- Tivoli Directory Server V6.3 클라이언트 이상

지원되는 운영 체제:

1.  Windows (2008년 7월 8일/2012년)
2.  AIX
3.  Linux
 - System p의 RHEL v4 및 v5
 - System p의 SUSE v9 및 v10
 - RHEL v4 및 v5 x86-64 32비트 및 64비트
 - SUSE v9 및 v10 x86-64 32비트 및 64비트

참고: 샘플은 다음 플랫폼에서 사용할 수 없습니다.

-  z/OS
-  IBM i

설치 및 구성

엑시트 모듈 및 연결 엔드포인트 스키마의 설치 및 구성

엑시트 모듈 설치

엑시트 모듈은 IBM MQ 설치 중 tools/samples/c/preconnexit/bin에 설치됩니다. 32비트 플랫폼의 경우 엑시트 모듈을 사용하려면 먼저 exit/installation_name/에 복사해야 합니다. 64비트 플랫폼의 경우 엑시트 모듈은 사용하기 전에 exit64/installation_name/로 복사해야 합니다.

연결 엔드 포인트 스키마 설치

엑시트는 연결 엔드포인트 스키마, ibm-amq.schema을(를) 사용합니다. 엑시트를 사용할 수 있으려면 스키마 파일을 LDAP 서버로 가져와야 합니다. 스키마를 가져온 후에 속성에 대한 값을 추가해야 합니다.

다음은 연결 엔드 포인트 스키마 가져오기에 대한 예입니다. 예는 IBM Tivoli Directory Server(ITDS)가 사용되고 있다고 가정합니다.

- IBM Tivoli Directory Server가 실행 중인지 확인한 후 `ibm-amq.schema` 파일을 ITDS 서버로 복사하거나 FTP하십시오.
- ITDS 서버에서는 다음 명령을 입력하여 ITDS 저장소에 스키마를 설치하십시오. 여기서 *LDAP ID* 및 *LDAP password*는 LDAP 서버의 루트 DN 및 비밀번호입니다.
`ldapadd -D "LDAP ID" -w "LDAP password" -f ibm-amq.schema`
- 명령 창에서 다음 명령을 입력하거나 써드파티 도구를 사용하여 확인을 위해 스키마를 찾아보십시오.

```
ldapsearch objectclass=ibm-amqClientConnection
```

스키마 파일 가져오기에 대한 세부사항은 LDAP 서버 문서를 참조하십시오.

구성

PreConnect라는 새 섹션을 클라이언트 구성 파일(예: `mqclient.ini`)에 추가해야 합니다. PreConnect 섹션은 다음 키워드를 포함합니다.

모듈

API 엑시트 코드가 포함된 모듈의 이름입니다. 이 필드에 모듈의 전체 경로가 포함되는 경우 이는 있는 그대로 사용됩니다. 그렇지 않으면 IBM MQ 설치의 `exit` 또는 `exit64` 폴더가 검색됩니다.

Function

라이브러리의 종료 `LdapPreConnect` 코드를 포함하는 기능적 진입점의 이름. 함수 정의는 귀사의 함수 프로토타입을 준수합니다.



주의: 실제 종료 진입점을 지정할 때는 함수 선언문에서 따옴표를 제거해야 합니다.

데이터

채널 정의를 포함하는 LDAP 저장소의 URI.

다음 스니펫은 `mqclient.ini` 파일에 필요한 변경사항의 예입니다.

```
PreConnect:
Module=amqlcelp
Function="LdapPreconnectExit"
Data=ldap:dap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
Sequence=1
```

ALW 엑시트 및 스키마 개요

큐 관리자에 연결을 설정하는 데 구문 및 매개변수가 사용됩니다.

IBM MQ에서는 엑시트 모듈에서 시작점에 대한 다음 구문을 정의합니다.

```
void MQENTRY MQ_PRECONNECT_EXIT ( PMQNX pExitParms
                                   , PMQCHAR pQMgrName
                                   , PPMQCNO ppConnectOpts
                                   , PMQLONG pCompCode
                                   , PMQLONG pReason)
```

MQCONN/X 호출을 실행하는 동안 IBM MQ C 클라이언트는 기능 구문의 구현을 포함하는 엑시트 모듈을 로드합니다. 그런 다음 채널 정의를 검색하기 위해 엑시트 기능을 호출합니다. 큐 관리자에 연결을 설정하는 데 검색된 채널 정의가 사용됩니다.

매개변수

pExitParms

유형: `PMQNX` 입력/출력

PreConnection 엑시트 매개변수 구조. 구조는 엑시트의 호출자에 의해 할당되고 유지보수됩니다.

```
struct tagMQNX
{
    MQCHAR4    StrucId;           /* Structure identifier */
    MQLONG     Version;          /* Structure version number */
    MQLONG     ExitId;           /* Type of exit */
}
```

```

MQLONG      ExitReason;          /* Reason for invoking exit */
MQLONG      ExitResponse;        /* Response from exit */
MQLONG      ExitResponse2;       /* Secondary response from exit */
MQLONG      Feedback;           /* Feedback code (reserved) */
MQLONG      ExitDataLength;      /* Exit data length */
PMQCHAR      pExitDataPtr;       /* Exit data */
MQPTR       pExitUserAreaPtr;    /* Exit user area */
PMQCD *      ppMQCDArrayPtr;     /* Array of pointers to MQCDs */
MQLONG      MQCDArrayCount;      /* Number of entries found */
MQLONG      MaxMQCDVersion;      /* Maximum MQCD version */
};

```

pQMgrName

유형: PMQCHAR 입출력(I/O)

큐 관리자의 이름. 입력 시, 이 매개변수는 **QMgrName** 매개변수를 통해 MQCONN API 호출에 제공되는 필터 문자열입니다. 이 필드는 공백이거나 명시적일 수 있으며 특정 와일드카드 문자를 포함할 수 있습니다. 필드는 엑시트에 의해 변경됩니다. MQXR_TERM으로 엑시트를 호출하는 경우 매개변수는 NULL입니다.

ppConnectOpts

유형: ppConnectOpts 입출력(I/O)

MQCONN의 조치를 제어하는 옵션입니다. MQCONN API 호출의 조치를 제어하는 MQCNO 연결 옵션 구조에 대한 포인터입니다. MQXR_TERM으로 엑시트를 호출하는 경우 매개변수는 NULL입니다. MQI 클라이언트는 원래 애플리케이션에서 제공하지 않은 경우에도 항상 MQCNO 구조를 엑시트에 제공합니다. 애플리케이션이 MQCNO 구조를 제공하는 경우, 클라이언트는 수정된 엑시트에 전달하기 위해 복제본을 작성합니다. 이 클라이언트가 MQCNO의 소유권을 보유합니다. MQCNO를 통해 참조되는 MQCD는 배열을 통해 제공된 모든 연결 정의에서 우선순위를 갖습니다. 클라이언트는 MQCNO 구조를 사용하여 큐 관리자에 연결하며 나머지는 무시됩니다.

pCompCode

유형: PMQLONG 입력/출력

완료 코드. 엑시트 완료 코드를 수신하는 MQLONG에 대한 포인터입니다. 다음 값 중 하나여야 합니다.

- MQCC_OK - 성공적인 완료
- MQCC_WARNING - 경고(부분 완료)
- MQCC_FAILED - 호출 실패

pReason

유형: PMQLONG 입력/출력

pCompCode를 규정하는 이유. 엑시트 이유 코드를 수신하는 MQLONG에 대한 포인터입니다. 완료 코드가 MQCC_OK인 경우 유일한 올바른 값은 MQRC_NONE - (0, x'000')입니다. 보고할 이유가 없습니다.

완료 코드가 MQCC_FAILED 또는 MQCC_WARNING인 경우 엑시트 함수는 이유 코드 필드를 올바른 MQRC_* 값으로 설정할 수 있습니다.

ALW MQ LDAP 컨텍스트 정보

엑시트는 컨텍스트 정보에 대해 다음 데이터 구조를 사용합니다.

MQNLDAPECTX

MQNLDAPECTX 구조는 다음의 C 프로토타입을 가지고 있습니다.

```

typedef struct tagMQNLDAPECTX MQNLDAPECTX;
typedef MQNLDAPECTX MQPOINTER PMQNLDAPECTX;

struct tagMQNLDAPECTX
{
    MQCHAR4      StrucId;          /* Structure identifier */
    MQLONG      Version;          /* Structure version number */
    LDAP *      objectDirectory    /* LDAP Instance */
    MQLONG      ldapVersion;       /* Which LDAP version to use? */
    MQLONG      port;             /* Port number for LDAP server */
    MQLONG      sizeLimit;        /* Size limit */
    MQBOOL      ssl;              /* SSL enabled? */
    MQCHAR *      host;           /* Hostname of LDAP server */
    MQCHAR *      password;       /* Password of LDAP server */
    MQCHAR *      searchFilter;    /* LDAP search filter */
    MQCHAR *      baseDN;         /* Base Distinguished Name */
};

```

```
MQCHAR *      charSet;          /* Character set */
};
```

ALW 연결 엔드포인트 검색 엑시트를 빌드하기 위한 샘플 코드

소스 컴파일을 위해 샘플 코드 AIX, Linux, and Windows 조각을 사용할 수 있습니다.

소스 컴파일링

LDAP 클라이언트 라이브러리 예를 들어, IBM Tivoli Directory Server V6.3 클라이언트 라이브러리를 사용하여 소스를 컴파일할 수 있습니다. 이 문서에서는 Tivoli Directory Server V6.3 클라이언트 라이브러리를 사용 중인 것으로 가정합니다.

참고: 사전 연결 엑시트 라이브러리는 다음 LDAP 서버로 지원됩니다.

- IBM Tivoli Directory Server V6.3
- Novell eDirectory V8.2

다음 코드 스니펫에서는 엑시트 컴파일 방법에 대해 설명합니다.

Windows Windows 플랫폼에서 엑시트 컴파일링

엑시트 소스를 컴파일링하기 위해 다음 스니펫을 사용할 수 있습니다.

```
CC=cl.exe
LL=link.exe
CCARGS=/c /I. /DWIN32 /W3 /DNDEBUG /EHsc /D_CRT_SECURE_NO_DEPRECATED /Zl

# The libraries to include
LDLIBS=ws2_32.lib Advapi32.lib libibmldapstatic.lib libibmldapbgstatic.lib \
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib \
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib msvcrt.lib

OBJS=amqlcel0.obj

all: amqlcelp.dll

amqlcelp.dll: $(OBJS)
    $(LL) /OUT:amqlcelp.dll /INCREMENTAL /NOLOGO /DLL /SUBSYSTEM:WINDOWS /MACHINE: X86 \
    /DEF:amqlcelp.def $(OBJS) $(LDLIBS) /NODEFAULTLIB:msvcrt.lib

# The exit source
amqlcel0.obj: amqlcel0.c
    $(CC) $(CCARGS) $*.c
```

참고: Tivoli Directory Server IBM V6.3 클라이언트 라이브러리를 Microsoft Visual Studio 2003 컴파일러로 컴파일한 경우, IBM Tivoli Directory Server V6.3 클라이언트 라이브러리를, 또는 이후 버전 Microsoft Visual Studio 2012 컴파일러로 컴파일할 때 경고가 발생할 수 있습니다.

Linux AIX 출구를 컴파일 중입니다 AIX . Linux

다음 코드 스니펫은 Linux에서 엑시트 소스를 컴파일링하기 위한 것입니다. 일부 컴파일러 옵션은 다를 수 있습니다.

```
#Make file to build exit
CC=gcc

MQML=/opt/mqm/lib
MQMI=/opt/mqm/inc
TDSI=/opt/ibm/ldap/V6.3/include
XFLAG=-m32

TDSL=/opt/ibm/ldap/V6.3/lib
```

IBM Tivoli Directory Server와 함께 정적 및 동적 링크 라이브러리가 모두 제공되지만 한 가지 유형의 라이브러리만 사용할 수 있습니다. 이 스크립트는 정적 라이브러리를 사용하는 것으로 가정합니다.

```
#Use static libraries.
LDLIBS=-L$(TDSL) -libibmldapstatic

CFLAGS=-I. -I$(MQMI) -I$(TDSI)
```

```
all:amqlcepl
amqlcepl: amqlcel0.c
$(CC) -o cepl amqlcel0.c -shared -fPIC $(XFLAG) $(CFLAGS) $(LDLIBS)
```

ALW PreConnect 엑시트 모듈 호출

PreConnect 엑시트 모듈은 세 가지 다른 이유 코드로 호출할 수 있습니다. LDAP 서버에 대한 연결을 초기화하고 설정하기 위한 MQXR_INIT 이유 코드, LDAP 서버에서 채널 정의를 검색하기 위한 MQXR_PRECONNECT 이유 코드, 엑시트가 정리될 경우 MQXR_TERM 이유 코드입니다.

MQXR_INIT

엑시트는 LDAP 서버로 연결을 초기화하고 설정하기 위한 MQXR_INIT 이유 코드로 호출됩니다.

MQXR_INIT 호출 이전에, MQNXP 구조의 pExitDataPtr 필드는 mqclient.ini 파일(즉, LDAP) 내 PreConnect 스탠자의 Data 속성으로 채워집니다.

LDAP URL은 최소한 프로토콜, 호스트 이름, 포트 번호 및 검색에 대한 기본 DN으로 구성됩니다. 엑시트는 pExitDataPtr 필드 안에 포함된 LDAP URL을 구문 분석하고 MQNLDPCTX LDAP 검색 컨텍스트 구조를 할당하며 이를 적절하게 채웁니다. 이 구조의 주소는 pExitUserAreaPtr 필드에 저장됩니다. 올바르게 LDAP URL을 구문 분석하는 데 실패하는 경우 MQCC_FAILED 오류가 발생합니다.

이 시점에서 엑시트는 **MQNLDPCTX** 매개변수를 사용하여 LDAP 서버에 연결하고 바인딩합니다. 결과로 생기는 LDAP API 핸들도 이 구조에 저장됩니다.

MQXR_PRECONNECT

엑시트 모듈은 LDAP 서버에서 채널 정의를 검색하기 위한 MQXR_PRECONNECT 이유 코드로 호출됩니다.

엑시트는 지정된 필터와 일치하는 채널 정의에 대해 LDAP 서버를 검색합니다. **QMgrNameparameter**에 특정 큐 관리자 이름이 있으면 검색은 **ibm-amqQueueManagerName** LDAP 속성 값이 주어진 큐 관리자 이름과 일치하는 모든 채널 정의를 리턴합니다.

QMgrName 매개변수가 '*' 또는 '(공백)'이면 검색에서는 **ibm-amqIsClientDefault Connection** 엔드포인트 속성이 TRUE로 설정된 모든 채널 정의를 리턴합니다.

성공적으로 검색을 수행하고 나면 엑시트는 하나 또는 MQCD 정의 배열을 준비하여 호출자에게 다시 리턴합니다.

MQXR_TERM

엑시트가 정리될 경우 이 이유 코드로 엑시트가 호출됩니다. 이 정리를 수행하는 동안 엑시트는 LDAP 서버에서 연결을 끊고, MQNLDPCTX 구조, 포인터 배열 및 참조하는 모든 MQCD를 포함하여 엑시트에서 할당하고 유지보수하는 모든 메모리를 해제합니다. 다른 필드는 기본값으로 설정됩니다. **pQMgrName** 및

ppConnectOpts 엑시트 매개변수는 MQXR_TERM 이유 코드로 종료하는 동안 사용되지 않으며 NULL일 수 있습니다.

관련 참조

[클라이언트 구성 파일의 PreConnect 스탠자](#)

ALW LDAP 스키마

클라이언트 연결 데이터는 LDAP(Lightweight Directory Access Protocol) 디렉토리라는 글로벌 저장소에 저장됩니다. IBM MQ 클라이언트는 연결 정의를 얻기 위해 LDAP 디렉토리를 사용합니다. LDAP 디렉토리 내의 IBM MQ 클라이언트 연결 정의의 구조가 LDAP 스키마로 알려져 있습니다. LDAP 스키마는 서버에서 필터 또는 속성 값 어설션이 입력 항목의 속성과 일치하는지 그리고, 조작을 허용하고, 추가하며 수정하는지 판별하는 데 사용하는 속성 유형 정의, 오브젝트 클래스 정의 및 기타 정보의 컬렉션입니다.

LDAP 디렉토리에 데이터 저장

클라이언트 연결 정의는 연결 지점으로 알려진 디렉토리 트리 내의 특정 분기 아래에 있습니다. LDAP 디렉토리 안에 있는 다른 모든 노드와 같이, 연결 지점에는 연관된 식별 이름(DN)이 있습니다. 이 노드를 디렉토리에서 작성하는 조회에 대한 시작점으로 사용할 수 있습니다. LDAP 디렉토리를 조회할 때 필터링을 사용하여 클라이언트 연결 정의의 서브셋을 리턴하십시오. 디렉토리 트리의 다른 부분에 예를 들어, 사용자, 부서 또는 그룹에 부여된 권한을 기반으로 서브 트리에 대한 액세스를 제한할 수 있습니다.

고유 속성 및 클래스 정의

LDAP 스키마를 수정하여 클라이언트 채널 정의를 저장하십시오. 모두 LDAP 데이터 정의는 오브젝트와 속성이 필요합니다. 오브젝트 및 속성은 오브젝트 또는 속성을 고유하게 식별하는 오브젝트 ID(OID) 숫자에 의해 식별됩니다. LDAP 스키마 안에 있는 모든 클래스는 상위 오브젝트로부터 직접적으로 또는 간접적으로 상속됩니다. 클라이언트 채널 정의 오브젝트에는 상위 오브젝트의 속성이 포함되어 있습니다. 모두 LDAP 데이터 정의는 오브젝트와 속성이 필요합니다.

- 오브젝트 정의는 LDAP 속성의 컬렉션입니다.
- 속성은 LDAP 데이터 유형입니다.

각 속성 및 일반 IBM MQ 특성을 매핑하는 방법에 대한 설명은 [LDAP 속성](#)에서 설명됩니다.

▶ ALW LDAP 속성

LDAP 정의된 속성은 특정 클라이언트 연결 IBM MQ 속성에 직접 매핑됩니다.

IBM MQ 클라이언트 채널 디렉토리 문자열 속성

IBM MQ 특성에 대한 해당 매핑이 있는 문자열 속성이 다음 표에 나열됩니다. 속성은 directoryString(UTF-8 인코딩된 유니코드 즉, 서브세트 IA5/ASCII를 포함하는 변수 바이트 인코딩 시스템) 구문의 값을 보유할 수 있습니다. 해당 오브젝트 ID(OID)에 의해 구문이 지정됩니다.

표 170. IBM MQ 클라이언트 채널 디렉토리 문자열 속성		
LDAP 속성	설명	IBM MQ 특성
1062 페이지의 『공용 이름』	채널 이름과 정의된 큐 관리자 이름으로 구성되는 공통 이름.	
1062 페이지의 『ibm-amqChannelName』	채널 정의 이름.	CHANNEL
1062 페이지의 『ibm-amqConnectionName』	통신 연결 ID.	CONNNAME
1063 페이지의 『ibm-amqDescription』	채널 설명.	DESCR
1063 페이지의 『ibm-amqLocalAddress』	채널의 로컬 통신 주소.	LOCLADDR
1063 페이지의 『ibm-amqModeName』	LU 6.2 모드 이름.	MODENAME
1063 페이지의 『ibm-amqPassword』	사용할 수 있는 비밀번호.	PASSWORD
1063 페이지의 『ibm-amqQueueManagerName』	IBM MQ 클라이언트 애플리케이션에서 연결을 요청할 수 있는 큐 관리자 또는 큐 관리자 그룹의 이름.	QMNAME
1064 페이지의 『ibm-amqSecurityExitUserData』	보안 엑시트에 전달되는 사용자 데이터.	SCYDATA
1063 페이지의 『ibm-amqSecurityExitName』	채널 보안 엑시트에서 실행할 엑시트 프로그램의 이름.	SCYEXIT
1064 페이지의 『ibm-amqSslCipherSpec』	TLS 연결에 대한 단일 CipherSpec.	SSLCIPH
1064 페이지의 『ibm-amqSslPeerName』	IBM MQ 채널의 다른 끝에 있는 클라이언트 또는 피어 큐 관리자에서 인증서의 식별 이름(DN)을 확인합니다.	SSLPEER
1064 페이지의 『ibm-amqTransactionProgramName』	트랜잭션 프로그램 이름.	TPNAME

표 170. IBM MQ 클라이언트 채널 디렉토리 문자열 속성 (계속)		
LDAP 속성	설명	IBM MQ 특성
1064 페이지의 『 ibm-amqUserID 』	리모트 MCA로 보안 SNA 세션을 시작하려고 시도하는 경우 MCA에서 사용할 사용자 ID.	USERID

IBM MQ 클라이언트 연결 정수 속성

사전 정의된 값(예: 열거된 유형)을 가진 속성은 표준 정수로 저장됩니다. 이러한 값은 정수 값으로 LDAP 디렉토리에 저장되며 연관된 상수 이름을 사용하지 않습니다.

표 171. IBM MQ 클라이언트 채널 디렉토리 정수 속성		
LDAP 속성	설명	IBM MQ 특성
1064 페이지의 『 ibm-amqConnectionAffinity 』	동일한 큐 관리자 이름을 사용하여 여러 번 연결하는 클라이언트 애플리케이션이 동일한 클라이언트 채널을 사용하는지 판별합니다.	AFFINITY
1065 페이지의 『 ibm-amqClientChannelWeight 』	어느 클라이언트 연결 채널 정의가 사용되는지에 영향을 주는 가중치.	CLNTWGHT
1065 페이지의 『 ibm-amqHeartBeatInterval 』	전송 큐에 메시지가 없는 경우 송신 MCA로부터 전달되는 하트비트 플로우 간의 예상 시간.	HBINT
1065 페이지의 『 ibm-amqKeepAliveInterval 』	채널에 대한 제한시간 값.	KAINT
1065 페이지의 『 ibm-amqMaximumMessageLength 』	채널에서 전송할 수 있는 최대 메시지 길이.	MAXMSGL
1065 페이지의 『 ibm-amqSharingConversations 』	각 TCP/IP 채널 인스턴스를 공유하는 최대 대화 수.	SHARECNV
1065 페이지의 『 ibm-amqTransportType 』	사용될 전송 유형.	TRPTYPE

IBM MQ 클라이언트 채널 부울 속성

이 부울 속성은 IBM MQ 특성에 매핑되지 않습니다. 이 속성의 구문은 부울 값을 표시합니다.

표 172. IBM MQ 클라이언트 채널 부울 속성	
LDAP 속성	설명
1065 페이지의 『 IBM MQ 클라이언트 채널 부울 속성 <code>ibm-amqIsClientDefault</code> 』	이 부울 속성은 정의되지 않은 속성을 ibm-amqQueueManagerName 가진 항목을 검색하는 문제를 해결하기 위해 정의되었습니다.

IBM MQ 클라이언트 채널 목록 속성

IBM MQ 특성은 단일 값의 심표로 구분된 목록 속성으로 LDAP 디렉토리 내에 저장됩니다. 속성은 다른 디렉토리 문자열 속성과 동일한 방식으로 정의됩니다. IBM MQ 특성에 대한 해당 매핑과 함께 목록 속성이 다음 표에서 설명됩니다.

표 173. IBM MQ 클라이언트 채널 목록 속성		
LDAP 속성	설명	IBM MQ 특성
1066 페이지의 『 ibm-amqHeaderCompression 』	채널에서 지원하는 헤더 데이터 압축 기법의 목록.	COMPHDR

표 173. IBM MQ 클라이언트 채널 목록 속성 (계속)

LDAP 속성	설명	IBM MQ 특성
1066 페이지의 『ibm-amqMessageCompression』	채널에서 지원하는 메시지 데이터 압축 기법의 목록.	COMPMSG
1066 페이지의 『ibm-amqSendExitUserData』	송신 엑시트에 전달되는 사용자 데이터.	SENDDATA
1066 페이지의 『ibm-amqSendExitName』	채널 송신 엑시트에서 실행할 엑시트 프로그램의 이름.	SENDEXIT
1067 페이지의 『ibm-amqReceiveExitUserData』	수신 엑시트에 전달되는 사용자 데이터.	RCVDATA
1067 페이지의 『ibm-amqReceiveExitName』	채널 수신 엑시트에서 실행할 사용자 엑시트 프로그램의 이름.	RCVEXIT

ALW IBM MQ 클라이언트 채널 디렉토리 문자열 속성

문자열 속성과 해당 속성 IBM MQ 매핑. 속성은 directoryString(UTF-8 인코딩된 유니코드 즉, 서브세트로 IA5/ASCII를 포함하는 변수 바이트 인코딩 시스템) 구문의 값을 보유할 수 있습니다. 해당 오브젝트 ID(OID)에 의해 구문이 지정됩니다.

공용 이름

공용 이름(CN)은 채널 이름과 정의된 큐 관리자 이름으로 구성됩니다.

기존 속성입니다.

CN의 형식은 다음과 같습니다.

```
CN=CHANNEL_NAME(DEFINING_Q_MGR_NAME)
```

예를 들면, 다음과 같습니다.

```
CN=TC1(QM_T1)
```

이 속성에 대해서는 하나의 값만 지정할 수 있습니다.

이 속성은 문자열 속성이며 값은 대소문자를 구분하지 않습니다. 하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 하위 문자열(예: CN=jim* 여기서 CN은 속성)을 사용하여 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙이며, 하나 이상의 와일드카드가 포함되어 있습니다.

ibm-amqChannelName

이 속성은 채널 정의의 이름을 지정합니다.

이 속성은 대소문자를 구분하지 않는 최대 20개 문자의 단일 문자열 값을 가지고 있습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 하위 문자열을 사용하여 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙이며, 하나 이상의 와일드카드가 포함되어 있습니다.

ibm-amqConnectionName

이 LDAP 속성은 통신 연결 ID입니다. 이 채널에서 사용할 특정 통신 링크를 지정합니다.

이 속성은 최대 264개 문자의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

ibm-amqDescription

이 LDAP 속성은 채널 설명을 제공합니다.

이 속성은 최대 64바이트의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

ibm-amqLocalAddress

이 속성은 채널에 대해 로컬 통신 주소를 지정합니다.

이 속성은 최대 48개 문자의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

ibm-amqModeName

이 속성은 LU 6.2 연결에 사용하기 위한 것입니다. 통신 세션 할당이 수행될 때 연결의 세션 특성에 대한 추가 정보를 제공합니다.

이 속성은 정확히 8개 문자의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

ibm-amqPassword

이 LDAP 속성은 리모트 MCA로 보안 LU 6.2 세션을 시작하려고 시도하는 경우 MCA에서 사용할 수 있는 비밀번호를 지정합니다.

이 속성은 최대 12자리의 단일 정수 값을 가지고 있습니다. 기존 속성이 아닙니다.

ibm-amqQueueManagerName

이 속성은 IBM MQ 클라이언트 애플리케이션에서 연결을 요청할 수 있는 큐 관리자 또는 큐 관리자 그룹의 이름을 지정합니다.

이 속성은 최대 48개 문자의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

1065 페이지의 [『IBM MQ 클라이언트 채널 부울 속성 ibm-amqIsClientDefault』](#)도 참조하십시오.

ibm-amqSecurityExitName

이 LDAP 속성은 채널 보안 엑시트에서 실행할 엑시트 프로그램의 이름을 지정합니다.

채널 보안 엑시트가 시행되지 않는 경우 공백으로 두십시오.

이 속성은 최대 999개 문자의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 이 속성은 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

ibm-amqSecurityExitUserData

이 LDAP 속성은 보안 엑시트에 전달되는 사용자 데이터를 지정합니다.

이 속성은 최대 999개 문자의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

ibm-amqSslCipherSpec

이 LDAP 속성은 TLS 연결을 위한 단일 CipherSpec을 지정합니다.

이 속성은 최대 32개 문자의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

ibm-amqSslPeerName

이 LDAP 속성은 IBM MQ 채널의 다른 끝에 있는 클라이언트 또는 피어 큐 관리자에서 인증서의 식별 이름(DN)을 확인하는 데 사용됩니다.

이 속성은 최대 1,024바이트의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

ibm-amqTransactionProgramName

이 LDAP 속성은 트랜잭션 프로그램 이름을 지정합니다. LU 6.2 연결에 사용하기 위한 것입니다.

이 속성은 최대 64개 문자의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

ibm-amqUserID

이 LDAP 속성은 리모트 MCA로 보안 SNA 세션을 시작하려고 시도하는 경우 MCA에서 사용할 사용자 ID를 지정합니다.

이 속성은 정확히 12개 문자의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

ALW IBM MQ 클라이언트 연결 정수 속성

사전 정의된 값(예: 열거된 유형)을 가진 속성은 표준 정수로 저장됩니다. 이러한 값은 정수 값으로 LDAP 디렉토리에 저장되며 연관된 상수 이름을 사용하지 않습니다.

ibm-amqConnectionAffinity

이 LDAP 속성은 동일 큐 관리자 이름을 사용하여 여러 번 연결하는 클라이언트 애플리케이션이 동일한 클라이언트 채널을 사용할지 지정합니다.

이 속성은 단일 정수 값을 가지고 있습니다. 기존 속성이 아닙니다.

ibm-amqClientChannelWeight

이 LDAP 속성은 어느 클라이언트 연결 채널 정의가 사용되는지에 영향을 주는 가중치를 지정합니다.

클라이언트 채널 가중치 속성은 둘 이상의 적절한 정의가 사용 가능한 경우 클라이언트 채널 정의 선택을 결정하는 데 사용됩니다.

이 속성은 단일 정수 값을 가지고 있습니다. 기존 속성이 아닙니다.

ibm-amqHeartBeatInterval

이 LDAP 속성은 전송 큐에 메시지가 없는 경우 송신 MCA로부터 전달되는 하트비트 플로우 간의 예상 시간을 지정합니다.

이 속성은 단일 정수 값을 가지고 있습니다. 기존 속성이 아닙니다. 기본값은 1입니다. 기본값은 현재 MQSERVER 환경 변수 조작에서 설정됩니다.

ibm-amqKeepAliveInterval

이 LDAP 속성은 채널에 대한 제한시간 값을 지정하는 데 사용됩니다.

이 속성의 값은 채널에 대한 활성 유지 시간을 지정하는 통신 스택에 전달됩니다. 이를 사용하여 각 채널에 대해서도 다른 활성 유지 값을 지정할 수 있습니다.

이 속성은 단일 정수 값을 가지고 있습니다. 기존 속성이 아닙니다.

ibm-amqMaximumMessageLength

이 LDAP 속성은 채널에서 전송할 수 있는 최대 메시지 길이를 지정합니다.

이 속성의 기본값은 현재 MQSERVER 환경 변수 조작에 따라 104857600입니다. 이 속성은 단일 정수 값을 가지고 있으며 기존 속성이 아닙니다.

ibm-amqSharingConversations

이 LDAP 속성은 각 TCP/IP 채널 인스턴스를 공유하는 최대 대화 수를 지정합니다.

이 속성은 단일 정수 값을 가지고 있습니다. 이 속성은 기존 속성이 아닙니다.

ibm-amqTransportType

이 LDAP 속성은 사용될 전송 유형을 지정합니다.

이 속성은 단일 정수 값을 가지고 있습니다. 기존 속성이 아닙니다.

IBM MQ 클라이언트 채널 부울 속성 *ibm-amqIsClientDefault*

이 부울 속성은 IBM MQ 특성에 매핑되지 않습니다. 해당 **ibm-amqQueueManagerName** 속성이 정의되지 않은 항목을 검색하는 문제를 해결합니다.

사전 연결 엑시트 모듈은 일반적으로 **ibm-amqQueueManagerName** 속성의 값을 검색 기준으로 사용하여 LDAP 서버를 검색합니다. 이러한 조회는 **ibm-amqQueueManagerName** 속성 값이 MQCONN/X 호출에 지정된 큐 관리자의 이름과 일치하는 모든 입력항목을 리턴합니다. 하지만 클라이언트 채널 정의 테이블(CCDT)을 사용할 때, MQCONN/X 호출의 큐 매니저 이름을 공백으로 설정하거나 이름에 별표(*)를 접두부로 붙일 수 있습니다. 큐 관리자의 이름이 공백이면 클라이언트는 기본 큐 관리자에 연결합니다. 이름이 큐 관리자에 별표(*)를 접두부로 지정하는 경우에는 클라이언트는 큐 관리자에 연결합니다.

마찬가지로 입력 항목의 **ibm-amqQueueManagerName** 속성을 정의되지 않은 상태로 남겨 둘 수 있습니다. 이 런 경우 이 엔드포인트 정보를 사용하는 클라이언트는 큐 관리자에 연결할 수 있습니다. 예를 들어, 입력 항목에는 다음 행이 포함되어 있습니다.

```
ibm-amqChannelName = "CHANNEL1"
ibm-amqConnectionName = myhost(1414)
```

이 예에서, 클라이언트는 myhost에서 실행 중인 지정된 큐 관리자에 연결하려고 시도합니다.

그러나 LDAP 서버에서는 정의되지 않은 속성 값에 대해 검색이 수행되지 않습니다. 예를 들어, 입력 항목에 `ibm-amqQueueManagerName`을 제외한 연결 정보가 포함되어 있는 경우 검색 결과는 이 입력 항목을 포함하지 않습니다. 이 문제점을 극복하기 위해 `ibm-amqIsClientDefault`를 설정할 수 있습니다. 이는 부울 속성이며, 정의되지 않은 경우에는 FALSE 값을 가지고 있다고 가정합니다.

`ibm-amqQueueManagerName`이 정의되지 않고 검색의 일부가 될 것으로 예상되는 입력 항목의 경우 `ibm-amqIsClientDefault`를 TRUE로 설정하십시오. MQCONN/X에 대한 호출에 큐 관리자 이름으로 공백 또는 별표 (*)가 지정된 경우 사전 연결 엑시트는 `ibm-amqIsClientDefault` 속성 값이 TRUE로 설정된 모든 입력 항목에 대해 LDAP 서버를 검색합니다.

참고: `ibm-amqIsClientDefault`이 TRUE로 설정된 경우 `ibm-amqQueueManagerName` 속성을 설정하거나 정의하지 마십시오.

1063 페이지의 『`ibm-amqQueueManagerName`』도 참조하십시오.

IBM MQ 클라이언트 채널 목록 속성

IBM MQ 특성은 단일 값의 심표로 구분된 목록 속성으로 LDAP 디렉토리 내에 저장됩니다. 속성은 다른 디렉토리 문자열 속성과 동일한 방식으로 정의됩니다.

ibm-amqHeaderCompression

이 LDAP 속성은 채널에서 지원하는 헤더 데이터 압축 기법의 목록입니다.

이 속성의 최대 크기는 48개의 문자입니다. 기존 속성이 아닙니다.

이 속성에 대해서는 하나의 값만 지정할 수 있습니다.

이 목록 속성은 심표로 구분된 형식을 사용하는 디렉토리 문자열로 지정됩니다. 예를 들어, **ibm-amqHeaderCompression**에 지정된 값은 0이며 NONE으로 맵핑됩니다. 최대 허용 한계를 초과하는 값은 클라이언트에서 무시합니다. 예를 들어, `ibm-amqHeaderCompression`은 목록에 최대 16개 정수를 포함합니다.

ibm-amqMessageCompression

이 LDAP 속성은 채널에서 지원하는 메시지 데이터 압축 기법의 목록입니다.

이 속성의 최대 크기는 48개의 문자입니다. 기존 속성이 아닙니다.

이 속성은 다중 값을 지원하지 않습니다.

이 목록 속성은 심표로 구분된 형식을 사용하는 디렉토리 문자열로 지정됩니다. 예를 들어, 이 속성에 지정된 값은 1,2,4,16,32이며, 이는 기본 압축 순서인 RLE, ZLIBFAST, ZLIBHIGH, LZ4FAST 및 LZ4HIGH에 매핑됩니다.

최대 허용 한계를 초과하는 값은 클라이언트에서 무시합니다. 예를 들어, `ibm-amqMessageCompression`은 목록에 최대 16개 정수를 포함합니다.

ibm-amqSendExitName

이 LDAP 속성은 채널 송신 엑시트에서 실행할 엑시트 프로그램의 이름을 지정합니다.

이 속성은 최대 999개 문자의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

참고: `ibm-amqSendExitName` 및 `ibm-amqSendExitUserData`를 쌍으로 동기화해야 합니다. 사용자 데이터는 엑시트 이름으로 동기화되어야 합니다. 하나가 지정되면, 데이터를 포함하지 않더라도 나머지도 대칭적으로 지정해야 합니다.

ibm-amqSendExitUserData

이 LDAP 속성은 송신 엑시트에 전달되는 사용자 데이터를 지정합니다.

이 속성은 최대 999개 문자의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

참고: **ibm-amqSendExitName** 및 **ibm-amqSendExitUserData**를 쌍으로 동기화해야 합니다. 사용자 데이터는 엑시트 이름으로 동기화되어야 합니다. 하나가 지정되면, 데이터를 포함하지 않더라도 나머지도 대칭적으로 지정해야 합니다.

ibm-amqReceiveExitName

이 LDAP 속성은 채널 수신 사용자 엑시트에서 실행할 사용자 엑시트 프로그램의 이름을 지정합니다.

이 속성은 연속으로 실행되는 프로그램 이름의 목록입니다. 채널 수신 엑시트가 시행되지 않는 경우 공백으로 두십시오.

이 속성은 최대 999개 문자의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

참고: **ibm-amqReceiveExitName** 및 **ibm-amqReceiveExitUserData**를 쌍으로 동기화해야 합니다. 사용자 데이터는 엑시트 이름으로 동기화되어야 합니다. 하나가 지정되면, 데이터를 포함하지 않더라도 나머지도 대칭적으로 지정해야 합니다.

ibm-amqReceiveExitUserData

이 LDAP 속성은 수신 엑시트에 전달되는 사용자 데이터를 지정합니다.

일련의 수신 엑시트를 실행할 수 있습니다. 일련의 엑시트에 대한 사용자 데이터의 문자열은 쉼표, 공백 또는 두 가지 모두로 구분됩니다.

이 속성은 최대 999개 문자의 단일 문자열 값을 가지고 있으며, 대소문자를 구분하지 않습니다. 기존 속성이 아닙니다.

하위 문자열 일치는 무시됩니다. 하위 문자열 일치는 검색 필터에서 속성의 동작을 지정하는 하위 스키마에 사용된 일치 규칙입니다.

참고: **ibm-amqReceiveExitName** 및 **ibm-amqReceiveExitUserData**를 쌍으로 동기화해야 합니다. 사용자 데이터는 엑시트 이름으로 동기화되어야 합니다. 하나가 지정되면, 데이터를 포함하지 않더라도 나머지도 대칭적으로 지정해야 합니다.

IBM i RPG 샘플 프로그램을 사용하여 IBM i

이 항목은 .NET IBM MQ for IBM iFramework와 함께 제공되는 RPG 샘플 프로그램에 대해 설명합니다. 샘플은 MQI(Message Queue Interface)의 일반적인 사용을 보여줍니다.

샘플은 일반 프로그래밍 기술을 보여주려는 목적이 아니므로 제작 프로그램에서 원하는 몇 가지 오류 검사가 생략되어 있습니다. 그러나, 이러한 샘플은 사용자의 고유 메시지 큐잉 프로그램을 만드는 기반으로 사용하기에 적합합니다.

모든 샘플에 대한 소스 코드는 제품과 함께 제공됩니다. 이 소스에는 프로그램에서 보여준 메시지 큐잉 기술을 설명하는 주석이 포함되어 있습니다.

ILE 샘플 프로그램 세트가 하나 있습니다.

1. MQI에 대해 프로토타입 호출을 사용하는 프로그램(정적 바인드 호출)

소스가 QMQMSAMP/QRPGLESRC에 있습니다. 멤버에는 AMQ3xxx4라는 이름이 지정되고, 여기서 xxx는 샘플 함수를 나타냅니다. 복사 멤버는 QMQM/QRPGLESRC에 존재합니다. 각 멤버 이름에는 G 또는 H라는 접미부가 있습니다.

1068 페이지의 표 174에 IBM MQ for IBM i에서 제공되는 샘플 프로그램의 전체 목록과 지원되는 프로그래밍 언어별 프로그램의 이름이 나와 있습니다. 해당 이름은 모두 접두부 AMQ로 시작하고 이름의 4번째 문자가 프로그래밍 언어를 나타내는 점에 유의하십시오.

표 174. 샘플 프로그램의 이름	
	RPG(ILE)
Put 샘플	AMQ3PUT4
찾아보기 샘플	AMQ3GBR4
Get 샘플	AMQ3GET4
요청 샘플	AMQ3REQ4
에코 샘플	AMQ3ECH4
조회 샘플	AMQ3INQ4
설정 샘플	AMQ3SET4
트리거 모니터 샘플	AMQ3TRG4
트리거 서버 샘플	AMQ3SRV4

뿐만 아니라, IBM MQ for IBM i 샘플 옵션에는 특정 샘플 프로그램과 관리 태스크를 표시하는 샘플 CL 프로그램에 대한 입력으로 사용할 수 있는 샘플 데이터 파일 AMQSDATA가 포함되어 있습니다. CL 샘플은 IBM i관리에서 설명합니다. 샘플 CL 프로그램을 사용하여 이 주제에 설명된 샘플 프로그램에 사용할 큐를 작성할 수 있습니다.

샘플 프로그램 실행 방법에 대한 정보는 1069 페이지의 『RPG 샘플 프로그램 준비 및 실행 IBM i』의 내용을 참조하십시오.

IBM i RPG 샘플 프로그램에 표시된 기능들 IBM i

IBM MQ for IBM i 샘플 프로그램에 나타나는 기술 설명 표

일부 기술은 둘 이상의 샘플 프로그램에서 발생하지만 하나의 프로그램만 테이블에 나열됩니다. 모든 샘플은 MQOPEN 및 MQCLOSE 호출을 사용하여 큐를 열고 닫으므로 이 기술은 표에 별도로 나열되지 않습니다.

표 175. MQI 사용을 보여주는 샘플 프로그램	
기술	RPG(ILE)
MQCONN 및 MQDISC 호출 사용	AMQ3ECH4 또는 AMQ3INQ4
암시적으로 연결 및 연결 끊기	AMQ3PUT4
MQPUT 호출을 사용한 메시지 넣기	AMQ3PUT4
MQPUT1 호출을 사용한 단일 메시지 넣기	AMQ3ECH4 또는 AMQ3INQ4
요청 메시지에 응답	AMQ3INQ4
메시지 가져오기(대기 없음)	AMQ3GBR4
메시지 가져오기(시간 제한이 있는 대기)	AMQ3GET4
메시지 가져오기(데이터 변환 사용)	AMQ3ECH4
큐 찾아보기	AMQ3GBR4
공유 입력 큐 사용	AMQ3INQ4
독점 입력 큐 사용	AMQ3REQ4

표 175. MQI 사용을 보여주는 샘플 프로그램 (계속)	
기술	RPG(ILE)
MQINQ 호출 사용	AMQ3INQ4
MQSET 호출 사용	AMQ3SET4
응답 대상 큐 사용	AMQ3REQ4
예외 메시지 요청	AMQ3REQ4
잘린 메시지 허용	AMQ3GBR4
해석된 큐 이름 사용	AMQ3GBR4
처리 트리거	AMQ3SRV4 또는 AMQ3TRG4

참고: 모든 샘플 프로그램은 처리 결과를 포함한 스푼 파일을 생성합니다.

관련 참조

977 페이지의 『C, COBOL 및 RPG MQI 샘플에 표시된 기능은 다음과 같습니다. IBM i』
샘플 프로그램에서 시연한 기법을 IBM MQIBM i 보여줍니다.

IBM i RPG 샘플 프로그램 준비 및 실행 IBM i

시작하기 전에

C 및 COBOL 샘플은 에서 참조하십시오 983 페이지의 『C 및 COBOL 샘플 준비 및 실행 IBM i』.

RPG 샘플 프로그램을 실행하기 전에, 다른 IBM MQ for IBM i 애플리케이션과 마찬가지로 컴파일해야 합니다. 이렇게 하려면 IBM i 명령 CRTRPGMOD 및 CRTPGM을 사용합니다.

AMQ3xxx4 프로그램을 작성할 때 CRTPGM 명령에 BNDSRVPGM(QMQM/LIBMQM)을 지정해야 합니다. 이 작업에는 프로그램에 있는 다양한 IBM MQ 프로시저가 포함됩니다.

이 태스크 정보

RPG 샘플 프로그램의 소스 코드는 라이브러리 QMQMSAMP에 QRPGLSRC의 멤버로 제공됩니다. 라이브러리 QMQM에서 제공되는 복사 파일을 사용하므로 컴파일할 때 이 라이브러리가 라이브러리 목록에 있는지 확인하십시오. 샘플은 복사 파일에 선언되는 많은 변수를 사용하지 않기 때문에 RPG 컴파일러에서 정보 메시지를 제공합니다.

샘플을 실행할 때 사용자 정의 큐를 사용할 수 있으며, 샘플 프로그램을 실행하여 AMQSAMP4 샘플 큐를 생성할 수도 있습니다. 이 프로그램의 소스 코드는 라이브러리 QMQMSAMP의 파일 QCLSRC에 포함되어 있습니다. CRTCLPGM 명령을 사용하여 이를 컴파일할 수 있습니다.

예

샘플 프로그램 중 하나를 호출하려면 다음 명령을 사용하십시오.

```
CALL PGM(QMQMSAMP/AMQ3PUT4) PARM('Queue_Name', 'Queue_Manager_Name')
```

Queue_Name 및 Queue_Manager_Name이 48자여야 하는 경우, Queue_Name 및 Queue_Manager_Name에 필요한 수만큼 공백을 채우면 됩니다.

Inquire 및 Set 샘플 프로그램의 경우, 생성된 샘플 정의로 AMQSAMP4 인해 해당 샘플의 C 버전이 트리거됩니다. RPG 버전을 트리거하려면 프로세스 정의 SYSTEM.SAMPLE.ECHOPROCESS 및 SYSTEM.SAMPLE.INQPROCESS, SYSTEM.SAMPLE.SETPROCESS를 변경해야 합니다. CHGMQMPCR 명령어 (CHGMQMPCR(MQ 프로세스 변경) 항목 참조)를 사용하거나, 대체 정의로 프로그램을 AMQSAMP4 편집 및 실행할 수 있습니다.

IBM i Browse RPG 샘플 프로그램 IBM i

찾아보기 샘플 프로그램, AMQ3GBR4는 MQGET 호출을 사용하여 큐에서 메시지를 찾습니다.

프로그램을 호출하면 프로그램은 사용자가 지정한 큐에서 모든 메시지의 사본을 검색합니다. 해당 메시지는 큐에 그대로 있습니다. 제공된 큐 SYSTEM.SAMPLE.LOCAL을 사용할 수 있습니다. 먼저 Put 샘플 프로그램을 실행하여 일부 메시지를 큐에 넣습니다. 큐 SYSTEM.SAMPLE.ALIAS를 사용할 수 있는데, 이는 동일한 로컬 큐에 대한 알리아스 이름입니다. 프로그램은 큐의 끝에 도달하거나 MQI 호출에 실패할 때까지 계속됩니다.

RPG 프로그램 호출 명령의 예는 다음과 같습니다.

```
CALL PGM(QMQMSAMP/AMQ3GBR4) PARM('Queue_Name', 'Queue_Manager_Name')
```

Queue_Name 및 Queue_Manager_Name이 48자여야 하는 경우, Queue_Name 및 Queue_Manager_Name에 필요한 수만큼 공백을 채우면 됩니다. 따라서 SYSTEM.SAMPLE.LOCAL을 대상 큐로 사용하는 경우, 29개 공백 문자가 필요합니다.

찾아보기 샘플 프로그램의 디자인

이 프로그램은 OOBROW 옵션과 MQOPEN 호출을 사용하여 대상 큐를 엽니다. 큐를 열 수 없으면 프로그램에서 MQOPEN 호출이 리턴한 이유 코드를 포함하는 오류 메시지를 스펴 파일에 기록합니다.

큐의 각 메시지의 경우 프로그램은 큐에서 메시지를 복사하기 위해 MQGET 호출을 사용한 후 메시지에 포함된 데이터를 표시합니다. MQGET 호출은 이러한 옵션을 사용합니다.

GMBRW

MQOPEN 호출 후, 찾아보기 커서가 큐의 첫 번째 메시지 앞에 논리적으로 배치되므로 이 옵션을 사용하면 첫 호출 시 첫 번째 메시지가 리턴됩니다.

GMNWT

큐 위의 메시지가 없는 경우 프로그램은 대기하지 않습니다.

GMATM

MQGET 호출은 고정된 크기의 버퍼를 지정합니다. 메시지가 이 버퍼보다 긴 경우 프로그램은 메시지가 잘렸다는 경과와 함께 잘린 메시지를 표시합니다.

각 MQGET 호출 이후 MQMD 구조의 MDMID 및 MDCID 필드를 어떻게 지워야 하는지가 프로그램에 표시됩니다. 이 호출이 해당 필드를 검색한 메시지에 있는 값으로 설정하기 때문입니다. 이러한 필드를 지운다는 것은 연속적인 MQGET 호출이 메시지가 큐에 보유되는 순서대로 메시지를 검색한다는 것을 의미합니다.

큐의 끝까지 프로그램이 계속됩니다. 따라서 MQGET 호출은 RC2033(메시지 없음) 이유 코드를 리턴하고 프로그램에 경고 메시지가 표시됩니다. MQGET 호출이 실패하면 스펴 파일에 이유 코드를 포함한 오류 메시지가 기록됩니다.

그런 다음 프로그램은 MQCLOSE 호출을 사용하여 큐를 닫습니다.

관련 개념

990 페이지의 『IBM i의 찾아보기 샘플 프로그램』

프로그램을 호출하면 각 프로그램은 사용자가 지정한 큐의 모든 메시지 사본을 검색합니다. 메시지는 큐에 남아 있습니다.

IBM i 에코 RPG 샘플 프로그램 IBM i

에코 샘플 프로그램은 응답 큐로 송신된 메시지를 리턴합니다. 프로그램의 이름은 AMQ3ECH4입니다.

트리거 프로세스가 작동하려면 사용할 에코 샘플 프로그램이 SYSTEM.SAMPLE.ECHO 큐에 도착하는 메시지에 의해 트리거되는지 확인해야 합니다. 이렇게 하려면 사용할 에코 샘플 프로그램의 이름을 프로세스 정의 SYSTEM.SAMPLE.ECHOPROCESS의 ApplId 필드에 지정하십시오. (이 경우, IBM i 관리에 설명된 CHGMQMPRC 명령을 사용할 수 있습니다.) 샘플 큐의 트리거 유형이 FIRST이므로, 요청 샘플을 실행하기 전에 이미 메시지가 큐에 있는 경우 사용자가 송신한 메시지로 인해 에코 샘플이 트리거되지 않습니다.

정의를 올바르게 설정한 경우, 먼저 하나의 작업에서 AMQ3SRV4를 시작한 다음 다른 작업에서 AMQ3REQ4를 시작하십시오. AMQ3SRV4 대신 AMQ3TRG4를 사용할 수 있지만, 작업 제출 지연 가능성 때문에 과정을 따르기가 쉽지 않을 수 있습니다.

요청 샘플 프로그램을 사용하여 메시지를 SYSTEM.SAMPLE.ECHO 큐에 보냅니다. Echo 샘플 프로그램은 요청 메시지에 데이터를 포함하는 응답 메시지를 요청 메시지에 지정된 응답 큐로 보냅니다.

에코 샘플 프로그램의 설계

프로그램이 트리거되는 경우, 이는 MQCONN 호출을 사용하여 기본 큐 관리자에 명시적으로 연결됩니다. IBM i에서는 이 기능이 필요하지 않지만, 다른 플랫폼에서 소스 코드를 변경하지 않고도 동일한 프로그램을 사용할 수 있습니다.

그런 다음 프로그램은 시작 시 전달된 트리거 메시지 구조에 지정된 큐를 엽니다. (명확성을 위해 이를 요청 큐라고 합니다.) 프로그램은 MQOPEN 호출을 사용하여 공유 입력에 대한 이 큐를 엽니다.

프로그램은 MQGET 호출을 사용하여 이 큐에서 메시지를 제거합니다. 이 호출은 GMATM 및 GMWT 옵션을 사용하여 대기 간격은 5초입니다. 프로그램은 요청 메시지인지 확인하기 위해 각 메시지의 디스크립터를 테스트합니다. 요청 메시지가 아닌 경우 프로그램은 메시지를 제거하고 경고 메시지를 표시합니다.

요청 큐에서 제거된 각 요청 메시지를 대상으로, 프로그램은 MQPUT 호출을 사용하여 응답 메시지를 응답 대상 큐에 넣습니다. 이 메시지는 요청 메시지의 콘텐츠를 포함합니다.

요청 큐에 남아 있는 메시지가 없는 경우 프로그램은 해당 큐를 닫고 큐 관리자의 연결을 끊습니다.

또한 이 프로그램은 이 상황에 대한 샘플이 제공되지 않더라도 IBM i 이외의 플랫폼에서 큐로 송신된 메시지에 응답할 수 있습니다. 에코(ECHO) 프로그램이 작동하려면 다음을 수행합니다.

- 프로그램을 작성하고, *Format, Encoding* 및 *CCSID* 필드를 올바르게 지정하여 텍스트 요청 메시지를 송신합니다.

ECHO 프로그램은 필요한 경우 큐 관리자에게 메시지 데이터 변환을 수행하도록 요청합니다.

- 작성한 프로그램이 응답을 위해 비슷한 변환을 제공하지 않는 경우, IBM MQ for IBM i 송신 채널에 CONVERT(*YES)를 지정하십시오.

관련 개념

1000 페이지의 『Echo 샘플 프로그램』

Echo 샘플 프로그램은 메시지 큐의 메시지를 응답 큐로 에코합니다.

IBM i Get RPG 샘플 프로그램 IBM i

Get 샘플 프로그램, AMQ3GET4는 MQGET 호출을 사용하여 큐에서 메시지를 가져옵니다.

프로그램이 호출되면 지정된 큐에서 메시지를 제거합니다. 제공된 큐 SYSTEM.SAMPLE.LOCAL을 사용할 수 있습니다. 먼저 Put 샘플 프로그램을 실행하여 일부 메시지를 큐에 넣습니다. SYSTEM.SAMPLE.ALIAS 큐를 사용할 수 있는데, 이는 동일한 로컬 큐에 대한 알리어스 이름입니다. 프로그램은 큐가 비거나 MQI 호출이 실패할 때까지 계속됩니다.

RPG 프로그램 호출 명령의 예는 다음과 같습니다.

```
CALL PGM(QMQMSAMP/AMQ3GET4) PARM('Queue_Name','Queue_Manager_Name')
```

여기서 Queue_Name 및 Queue_Manager_Name의 길이는 48자여야 하며 Queue_Name 및 Queue_Manager_Name에 필요한 수만큼 공백을 채우면 됩니다. 따라서 SYSTEM.SAMPLE.LOCAL을 대상 큐로 사용하는 경우, 29개 공백 문자가 필요합니다.

Get 샘플 프로그램의 디자인

프로그램이 메시지를 가져올 대상 큐를 엽니다. OOINPQ 옵션과 MQOPEN 호출을 사용합니다. 큐를 열 수 없으면 프로그램에서 MQOPEN 호출이 리턴한 이유 코드를 포함하는 오류 메시지를 스푼 파일에 기록합니다.

큐의 각 메시지에 대해, 프로그램은 MQGET 호출을 사용하여 큐에서 메시지를 제거한 다음 메시지에 있는 데이터를 표시합니다. MQGET 호출은 GMWT 옵션을 사용하여 15초의 대기 간격(GMWT)을 지정하므로 큐에 메시지가 없으면 이 시간 동안 프로그램이 대기합니다. 이 간격의 만기 전에 메시지가 도착하지 않으면 호출이 실패하고 RC2033(메시지 없음) 이유 코드를 리턴합니다.

각 MQGET 호출 이후 MQMD 구조의 *MDMID* 및 *MDCID* 필드를 어떻게 지워야 하는지가 프로그램에 표시됩니다. 이 호출이 해당 필드를 검색한 메시지에 있는 값으로 설정하기 때문입니다. 이러한 필드를 지운다는 것은 연속적인 MQGET 호출이 메시지가 큐에 보유되는 순서대로 메시지를 검색한다는 것을 의미합니다.

MQGET 호출은 고정된 크기의 버퍼를 지정합니다. 메시지가 이 버퍼보다 더 긴 경우 호출에 실패하고 프로그램은 중지됩니다.

MQGET 호출이 RC2033(메시지 없음) 이유 코드를 리턴하거나 MQGET 호출이 실패할 때까지 프로그램이 계속됩니다. 호출에 실패하는 경우 프로그램은 이유 코드를 포함하는 오류 메시지를 표시합니다.

그런 다음 프로그램은 MQCLOSE 호출을 사용하여 큐를 닫습니다.

관련 개념

[1001 페이지의 『Get 샘플 프로그램』](#)

Get 샘플 프로그램은 MQGET 호출을 사용하여 큐에서 메시지를 가져옵니다.

IBM i Inquire RPG 샘플 프로그램 IBM i

조회 샘플 프로그램, AMQ3INQ4는 MQINQ 호출을 사용하여 큐의 몇 가지 속성을 조회합니다.

이 프로그램은 트리거된 프로그램으로 실행되므로 유일한 입력은 MQTMC(트리거 메시지) 구조입니다. 이 구조에는 조회될 속성이 있는 대상 큐의 이름이 포함되어 있습니다.

트리거 프로세스가 작동하려면 조회 샘플 프로그램이 SYSTEM.SAMPLE.INQ 큐에 도착하는 메시지에 의해 트리거되는지 확인해야 합니다. 이렇게 하려면 SYSTEM.SAMPLE.INQPROCESS 프로세스 정의의 *AppId* 필드에 조회 샘플 프로그램의 이름을 지정하십시오. (이를 위해 [MQ 프로세스 변경\(CHGMQMPRC\)](#)에 설명된 CHGMQMPRC 명령을 사용할 수 있습니다.) 샘플 큐의 트리거 유형은 FIRST이므로, Request 샘플을 실행하기 전에 큐에 이미 메시지가 있으면 보내는 메시지에 의해 Inquire 샘플이 트리거되지 않습니다.

정의를 올바르게 설정한 경우, 먼저 하나의 작업에서 AMQ3SRV4를 시작한 다음 다른 작업에서 AMQ3REQ4를 시작하십시오. AMQ3SRV4 대신 AMQ3TRG4를 사용할 수 있지만, 작업 제출 지연 가능성 때문에 과정을 따르기가 쉽지 않을 수 있습니다.

각각 큐 이름만 포함한 요청 메시지를 SYSTEM.SAMPLE.INQ 큐로 송신하려면 요청 샘플 프로그램을 사용하십시오. 각 요청 메시지를 대상으로, 조회 샘플 프로그램은 요청 메시지에 지정된 큐 정보를 포함한 응답 메시지를 송신합니다. 응답은 요청 메시지에 지정된 응답 대상 큐로 송신됩니다.

조회 샘플 프로그램의 디자인

프로그램이 트리거되는 경우, 이는 MQCONN 호출을 사용하여 기본 큐 관리자에 명시적으로 연결됩니다. IBM i에서는 필요하지 않지만, 이러한 설계 덕분에 다른 플랫폼에서 소스 코드를 변경하지 않고도 동일한 프로그램을 사용할 수 있습니다.

그런 다음 프로그램은 시작 시 전달된 트리거 메시지 구조에 지정된 큐를 엽니다. (명확성을 위해 이를 요청 큐라고 합니다.) 프로그램은 MQOPEN 호출을 사용하여 공유 입력에 대한 이 큐를 엽니다.

프로그램은 MQGET 호출을 사용하여 이 큐에서 메시지를 제거합니다. 이 호출은 GMATM 및 GMWT 옵션을 사용하며 대기 간격은 5초입니다. 프로그램은 요청 메시지인지를 확인하기 위해 각 메시지의 디스크립터를 테스트합니다. 아닌 경우, 프로그램은 메시지를 제거하고, 경고 메시지를 표시합니다.

요청 큐에서 제거된 각 요청 메시지를 대상으로, 프로그램은 데이터에 있는 큐(대상 큐라고 하겠음)의 이름을 읽어오고 OOINQ 옵션과 MQOPEN 호출을 사용하여 해당 큐를 엽니다. 그럼 다음 MQINQ 호출을 사용하여 대상 큐의 **InhibitGet**, **CurrentQDepth** 및 **OpenInputCount** 속성 값을 조회합니다.

MQINQ 호출이 성공하면 프로그램은 MQPUT 호출을 사용하여 응답 메시지를 응답 대상 큐에 넣습니다. 이 메시지는 세 가지 속성 값을 포함합니다.

MQOPEN 또는 MQINQ 호출이 실패하면 프로그램은 MQPUT 호출을 사용하여 보고 메시지를 응답 대상 큐에 넣습니다. 이 보고 메시지의 메시지 디스크립터의 *MDFB* 필드에 실패한 항목에 따라 MQOPEN 또는 MQINQ 호출에서 리턴된 이유 코드가 있습니다.

MQINQ 호출 후 프로그램은 MQCLOSE 호출을 사용하여 대상 큐를 닫습니다.

요청 큐에 남아 있는 메시지가 없는 경우 프로그램은 해당 큐를 닫고 큐 관리자의 연결을 끊습니다.

관련 개념

1007 페이지의 『조희 샘플 프로그램』

조희 샘플 프로그램은 MQINQ 호출을 사용하여 큐의 일부 속성에 대해 조회합니다.

IBM i Put RPG 샘플 프로그램 IBM i

Put 샘플 프로그램, AMQ3PUT4는 MQPUT 호출을 사용하여 메시지를 큐에 넣습니다.

프로그램을 시작하려면, 프로그램을 호출하고 프로그램 매개변수로 대상 큐의 이름을 지정하십시오. 프로그램에서 고정 메시지 세트를 큐에 넣습니다. 이 메시지는 프로그램 소스 코드 끝의 데이터 블록에서 가져온 것입니다. 샘플 Put 프로그램은 라이브러리 QMQMSAMP에서 AMQ3PUT4입니다.

이 예제 프로그램을 사용하는 경우, 명령은 다음과 같습니다.

```
CALL PGM(QMQMSAMP/AMQ3PUT4) PARM('Queue_Name','Queue_Manager_Name')
```

Queue_Name 및 Queue_Manager_Name이 48자여야 하는 경우, Queue_Name 및 Queue_Manager_Name에 필요한 수만큼 공백을 채우면 됩니다.

Put 샘플 프로그램의 디자인

프로그램은 OOOOUT 옵션과 MQOPEN 호출을 사용하여 메시지를 넣을 대상 큐를 엽니다. 그 결과가 스푼 파일의 출력입니다. 큐를 열 수 없으면 프로그램에서 MQOPEN 호출이 리턴한 이유 코드를 포함하는 오류 메시지를 기록합니다. 프로그램을 단순히 유지하기 위해 이 호출 및 후속 MQI 호출에서 프로그램은 여러 옵션의 기본값을 사용합니다.

소스 코드에 있는 각 데이터 행에 대해, 텍스트를 버퍼로 읽어오고 MQPUT 호출을 사용하여 해당 행의 텍스트를 포함한 데이터그램 메시지를 작성합니다. 프로그램은 입력의 끝에 도달하거나 MQPUT 호출에 실패할 때까지 계속됩니다. 프로그램이 입력의 끝에 도달하는 경우 MQCLOSE 호출을 사용하여 큐를 닫습니다.

관련 참조

1013 페이지의 『Put 샘플 프로그램』

Put 샘플 프로그램은 MQPUT 호출을 사용하여 큐에 메시지를 넣습니다.

IBM i 요청 RPG 샘플 프로그램 IBM i

요청 샘플 프로그램, AMQ3REQ4는 클라이언트/서버 처리를 보여줍니다. 샘플은 서버 프로그램이 처리한 요청 메시지를 큐에 넣는 클라이언트입니다. 서버 프로그램이 응답 메시지를 응답 대상 큐에 넣을 때까지 대기합니다.

요청 샘플은 MQPUT 호출을 사용하여 일련의 요청 메시지를 큐에 넣습니다. 이 메시지가 응답 대상 큐로 SYSTEM.SAMPLE.REPLY를 지정합니다. 프로그램에서 응답 메시지를 대기했다가 표시합니다. 서버 애플리케이션이 대상 큐(서버 큐라고 함)를 처리하고 있거나 해당 목적으로 애플리케이션이 트리거된 경우(조회 및 설정 샘플 프로그램이 트리거되도록 설계됨)에만 응답이 송신됩니다. 샘플은 첫 번째 응답이 도착할 때까지 5분(서버 애플리케이션의 트리거 시간 허용), 후속 응답에 대해 15초 대기하지만, 응답을 가져오지 않고도 종료될 수 있습니다.

프로그램을 시작하려면, 프로그램을 호출하고 프로그램 매개변수로 대상 큐의 이름을 지정하십시오. 프로그램에서 고정 메시지 세트를 큐에 넣습니다. 이 메시지는 프로그램 소스 코드 끝의 데이터 블록에서 가져온 것입니다.

요청 샘플 프로그램의 디자인

프로그램이 메시지를 넣을 수 있도록 서버 큐를 엽니다. OOOOUT 옵션과 MQOPEN 호출을 사용합니다. 큐를 열 수 없는 경우 프로그램은 MQOPEN 호출에서 리턴한 이유 코드가 포함된 오류 메시지를 표시합니다.

그런 다음 응답 메시지를 가져올 수 있도록 프로그램은 SYSTEM.SAMPLE.REPLY라는 응답 대상 큐를 엽니다. 이를 위해 OOINPX 옵션과 MQOPEN 호출을 사용합니다. 큐를 열 수 없는 경우 프로그램은 MQOPEN 호출에서 리턴한 이유 코드가 포함된 오류 메시지를 표시합니다.

입력의 각 행에 대해 프로그램은 텍스트를 버퍼로 해석하고 MQPUT 호출을 사용하여 해당 행의 텍스트를 포함하는 요청 메시지를 작성합니다. 이 호출에서 프로그램은 ROEXCD 보고 옵션을 사용하여 요청 메시지에 관해 발송된 보고 메시지에 메시지 데이터의 처음 100바이트가 포함되도록 요청합니다. 프로그램은 입력의 끝에 도달하거나 MQPUT 호출에 실패할 때까지 계속됩니다.

그런 다음 프로그램은 MQGET 호출을 사용하여 큐에서 응답 메시지를 제거하고 응답에 포함된 데이터를 표시합니다. MQGET 호출은 GMWT 옵션을 사용하여 첫 번째 응답에 대해 5분(서버 애플리케이션의 트리거 시간 허용), 후속 응답에 대해 15초의 대기 간격(GMWI)을 지정합니다. 큐에 메시지가 없는 경우 프로그램은 이 기간 동안 대기합니다. 이 간격의 만기 전에 메시지가 도착하지 않으면 호출이 실패하고 RC2033(메시지 없음) 이유 코드를 리턴합니다. 또한 호출이 GMATM 옵션을 사용하므로 선언된 버퍼 크기보다 긴 메시지는 잘립니다.

각 MQGET 호출 이후 MQMD 구조의 MDMID 및 MDCOD 필드를 어떻게 지워야 하는지가 프로그램에 표시됩니다. 이 호출이 해당 필드를 검색한 메시지에 있는 값으로 설정하기 때문입니다. 이러한 필드를 지운다는 것은 연속적인 MQGET 호출이 메시지가 큐에 보유되는 순서대로 메시지를 검색한다는 것을 의미합니다.

MQGET 호출이 RC2033(메시지 없음) 이유 코드를 리턴하거나 MQGET 호출이 실패할 때까지 프로그램이 계속됩니다. 호출에 실패하는 경우 프로그램은 이유 코드를 포함하는 오류 메시지를 표시합니다.

그런 다음 프로그램이 MQCLOSE 호출을 사용하여 서버 큐와 응답 대상 큐를 닫습니다. [1074 페이지의 표 176](#)은 조회 및 설정 샘플 프로그램 실행에 필요한 에코 샘플 프로그램의 변경사항을 보여줍니다.

참고: 에코 샘플 프로그램에 대한 자세한 정보가 참고로 제공됩니다.

표 176. 클라이언트/서버 샘플 프로그램 세부사항		
프로그램 이름	SYSTEM/SAMPLE 큐	프로그램 시작
에코	ECHO	AMQ3ECH4
조회	INQ	AMQ3INQ4
설정	SET	AMQ3SET4

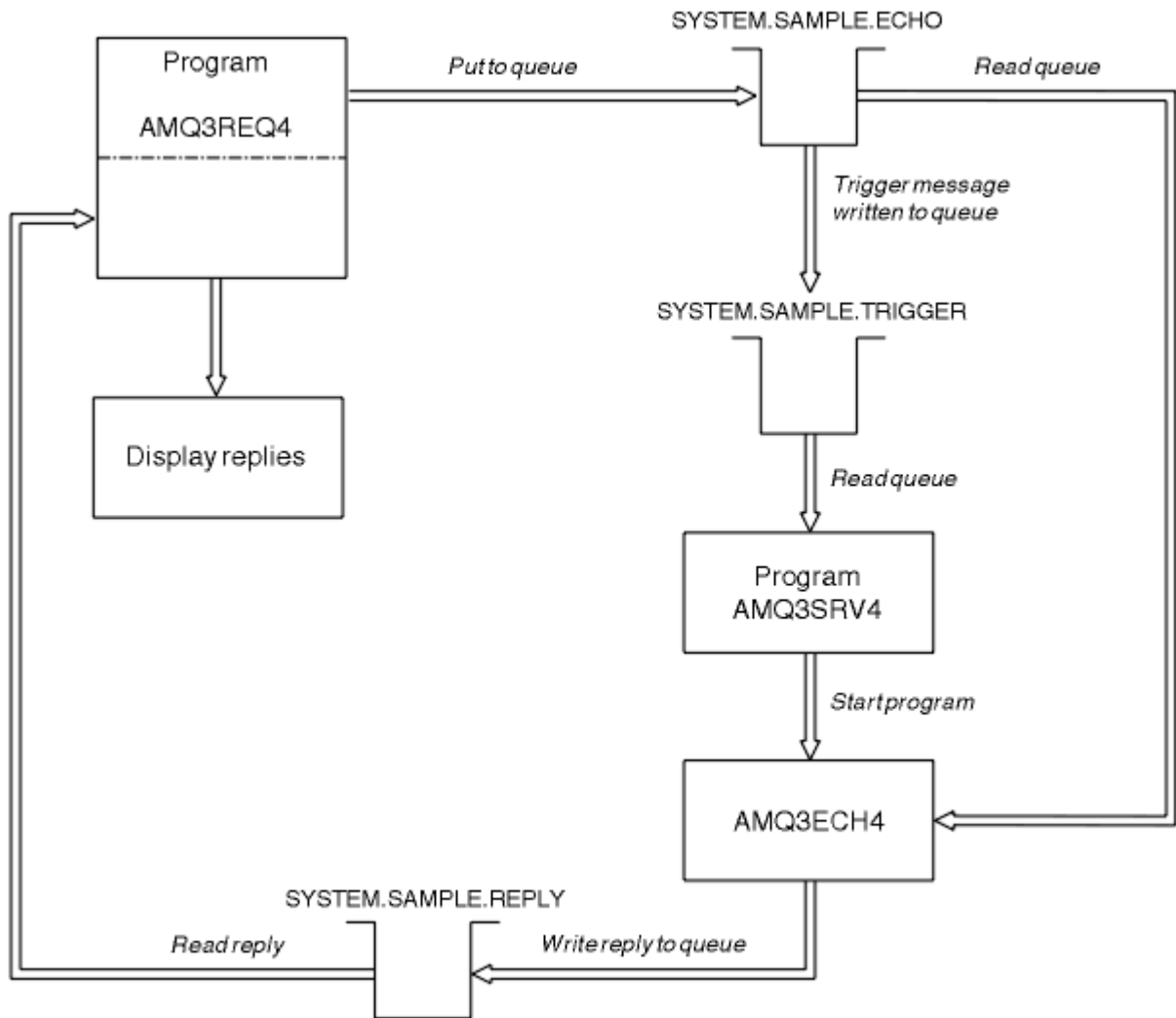


그림 155. 샘플 클라이언트/서버(에코) 프로그램 흐름도

관련 개념

1021 페이지의 『요청 샘플 프로그램』

요청 샘플 프로그램에서는 클라이언트/서버 처리를 보여줍니다. 샘플은 서버 프로그램에서 처리하는 대상 서버 큐 위에 요청 메시지를 넣는 클라이언트입니다. 서버 프로그램이 응답 대상 큐에 응답 메시지를 넣을 때까지 대기합니다.

IBM i Request RPG 샘플에 트리거링 사용하기 IBM i

트리거를 사용하여 샘플을 실행하려면 하나의 작업에서 필요한 이니시에이션 큐를 대상으로 트리거 서버 프로그램, AMQ3SRV4를 시작한 다음 다른 작업에서 AMQ3REQ4를 시작하십시오.

요청 샘플 프로그램에서 메시지를 보내는 경우 트리거 서버가 준비되었음을 의미합니다.

참고:

1. 샘플은 SYSTEM.SAMPLE.ECHO, SYSTEM.SAMPLE.INQ 또는 SYSTEM.SAMPLE.SET 로컬 큐에 대한 이니시에이션 큐로 SYSTEM SAMPLE TRIGGER 큐를 사용합니다. 또는, 고유의 이니시에이션 큐를 정의할 수 있습니다.
2. 프로그램이 AMQSAMP4 생성한 샘플 정의는 C 버전의 샘플이 트리거되도록 합니다. RPG 버전을 트리거하려면 프로세스 정의 SYSTEM.SAMPLE.ECHOPROCESS 및 SYSTEM.SAMPLE.INQPROCESS, SYSTEM.SAMPLE.SETPROCESS를 변경해야 합니다. CHGMQMPRC 명령(자세한 정보는 [MQ 프로세스 변경 \(CHGMQMPRC\)](#) 참조)을 사용하여 이 작업을 수행하거나, 고유 AMQSAMP4 버전을 편집 및 실행할 수 있습니다.

3. QMQMSAMP/QRPGLESRC에 제공되는 소스에서 트리거 서버 프로그램을 컴파일해야 합니다.

실행할 트리거 프로세스에 따라 AMQ3REQ4는 이들 샘플 서버 큐 중 하나에 요청 메시지를 넣도록 지정하는 매개변수를 사용하여 호출해야 합니다.

- SYSTEM.SAMPLE.ECHO (Echo 샘플 프로그램의 경우)
- SYSTEM.SAMPLE.INQ (Inquire 샘플 프로그램의 경우)
- SYSTEM.SAMPLE.SET (Set 샘플 프로그램의 경우)

SYSTEM.SAMPLE.ECHO 프로그램에 대한 플로우차트가 1075 페이지의 [그림 155](#)에 표시됩니다. 예를 사용하는 경우, 이 서버에 RPG 프로그램 요청을 발행하는 명령은 다음과 같습니다.

```
CALL PGM(QMQMSAMP/AMQ3REQ4) PARM('SYSTEM.SAMPLE.ECHO
+ 30 blank characters','Queue_Manager_Name')
```

큐 이름과 큐 관리자 이름의 길이는 48자여야 하기 때문입니다.

참고: 이 샘플 큐에는 FIRST 트리거 유형이 있으므로 Request 샘플을 실행하기 전에 큐에 이미 메시지가 있는 경우 서버 애플리케이션은 송신된 메시지에 의해 트리거되지 않습니다.

추가 예를 시도하려는 경우 다음 변형을 시도할 수 있습니다.

- 작업을 대신 제출하려면 AMQ3SRV4 대신 AMQ3TRG4를 사용하십시오. 작업 제출 지연 가능성 때문에 과정을 따르기가 쉽지 않을 수 있습니다.
- SYSTEM.SAMPLE.INQ 및 SYSTEM.SAMPLE.SET 샘플 큐를 사용하십시오. 예제 데이터 파일을 사용하는 경우, 이 서버에 RPG 프로그램 요청을 발행하는 명령은 다음과 같습니다.

```
CALL PGM(QMQMSAMP/AMQ3INQ4) PARM('SYSTEM.SAMPLE.INQ
+ 31 blank characters')
CALL PGM(QMQMSAMP/AMQ3SET4) PARM('SYSTEM.SAMPLE.SET
+ 31 blank characters')
```

큐 이름의 길이는 48자여야 하기 때문입니다.

이러한 샘플 큐에도 FIRST 트리거 유형이 있습니다.

IBM i Set RPG 샘플 프로그램 IBM i

설정 샘플 프로그램, AMQ3SET4는 MQSET 호출을 사용하여 큐에서 Put 조작이 큐의 **InhibitPut** 속성을 변경하지 못하도록 금지합니다.

이 프로그램은 트리거된 프로그램으로 실행되므로 유일한 입력은 조회할 속성을 포함한 대상 큐의 이름이 있는 MQTMC(트리거 메시지) 구조입니다.

트리거 프로세스가 작동하려면 설정 샘플 프로그램이 SYSTEM.SAMPLE.SET 큐에 도착하는 메시지에 의해 트리거되는지 확인해야 합니다. 이렇게 하려면 SYSTEM.SAMPLE.SETPROCESS 프로세스 정의의 *AppId* 필드에 설정 샘플 프로그램의 이름을 지정하십시오. (이 경우, IBM i 관리에 설명된 CHGMQMPRC 명령을 사용할 수 있습니다.) 샘플 큐의 트리거 유형이 FIRST이므로, 요청 샘플을 실행하기 전에 이미 메시지가 큐에 있는 경우 사용자가 송신한 메시지로 인해 설정 샘플이 트리거되지 않습니다.

정의를 올바르게 설정한 경우, 먼저 하나의 작업에서 AMQ3SRV4를 시작한 다음 다른 작업에서 AMQ3REQ4를 시작하십시오. AMQ3SRV4 대신 AMQ3TRG4를 사용할 수 있지만, 작업 제출 지연 가능성 때문에 과정을 따르기가 쉽지 않을 수 있습니다.

각각 큐 이름만 포함한 요청 메시지를 SYSTEM.SAMPLE.SET 큐로 송신하려면 요청 샘플 프로그램을 사용하십시오. 각 요청 메시지를 대상으로, 설정 샘플 프로그램은 Put 조작이 지정된 큐에서 금지되었는지 확인 내용을 포함한 응답 메시지를 송신합니다. 응답은 요청 메시지에 지정된 응답 대상 큐로 송신됩니다.

설정 샘플 프로그램의 디자인

프로그램이 트리거되는 경우, 이는 MQCONN 호출을 사용하여 기본 큐 관리자에 명시적으로 연결됩니다. IBM i에서는 필요하지 않지만, 다른 플랫폼에서 소스 코드를 변경하지 않고도 동일한 프로그램을 사용할 수 있습니다.

그런 다음 프로그램은 시작 시 전달된 트리거 메시지 구조에 지정된 큐를 엽니다. (명확성을 위해 이를 요청 큐라고 합니다.) 프로그램은 MQOPEN 호출을 사용하여 공유 입력에 대한 이 큐를 엽니다.

프로그램은 MQGET 호출을 사용하여 이 큐에서 메시지를 제거합니다. 이 호출은 GMATM 및 GMWT 옵션을 사용하여 대기 간격은 5초입니다. 프로그램은 요청 메시지의 디스크립터를 테스트합니다. 요청 메시지가 아닌 경우 프로그램은 메시지를 제거하고 경고 메시지를 표시합니다.

요청 큐에서 제거된 각 요청 메시지를 대상으로, 프로그램은 데이터에 있는 큐(대상 큐라고 하겠음)의 이름을 읽어오고 OOSSET 옵션과 MQOPEN 호출을 사용하여 해당 큐를 엽니다. 그런 다음 MQSET 호출을 사용하여 대상 큐의 **InhibitPut** 속성 값을 QAPUTI로 설정합니다.

MQSET 호출이 성공하면 프로그램은 MQPUT 호출을 사용하여 응답 메시지를 응답 대상 큐에 넣습니다. 이 메시지에는 PUT inhibited 문자열이 포함됩니다.

MQOPEN 또는 MQSET 호출이 실패하면 프로그램은 MQPUT 호출을 사용하여 보고 메시지를 응답 대상 큐에 넣습니다. 이 보고 메시지의 메시지 디스크립터의 *MDFB* 필드에 실패한 항목에 따라 MQOPEN 또는 MQSET 호출에서 리턴된 이유 코드가 있습니다.

MQSET 호출 후 프로그램은 MQCLOSE 호출을 사용하여 대상 큐를 닫습니다.

요청 큐에 남아 있는 메시지가 없는 경우 프로그램은 해당 큐를 닫고 큐 관리자의 연결을 끊습니다.

관련 개념

[1027 페이지의 『설정 샘플 프로그램』](#)

설정 샘플 프로그램은 큐의 **InhibitPut** 속성을 변경하기 위해 MQSET 호출을 사용하여 큐에서 put 조작을 금지합니다. 또한 설정 샘플 프로그램의 디자인에 대해 알아보십시오.

▶ IBM i 트리거링 RPG 샘플 프로그램 IBM i

IBM MQ for IBM i는 ILE/RPG로 작성된 두 가지 트리거 샘플 프로그램을 제공합니다.

프로그램은 다음과 같습니다.

AMQ3TRG4

IBM i 환경을 위한 트리거 모니터입니다. 애플리케이션이 시작되도록 IBM i 작업을 제출하지만, 이는 각 트리거 메시지와 연관된 추가 처리 비용이 있음을 의미합니다.

AMQ3SRV4

이는 IBM i 환경을 위한 트리거 서버입니다. 각 트리거 메시지의 경우 이 서버는 지정된 애플리케이션을 시작하기 위해 고유 작업에서 시작 명령을 실행합니다. 트리거 서버는 CICS 트랜잭션을 호출할 수 있습니다.

이 샘플의 C 언어 버전은 라이브러리 QMQM에서 AMQSTRG4 및 AMQSERV4라는 실행 가능 프로그램으로 제공됩니다.

관련 개념

[1031 페이지의 『트리거링 샘플 프로그램』](#)

트리거링 샘플에서 제공된 기능은 **runmqtrm** 프로그램의 트리거 모니터에 제공된 기능의 서브세트입니다.

▶ IBM i AMQ3TRG4 용 RPG 샘플 트리거 모니터 IBM i

AMQ3TRG4는 트리거 모니터입니다. 한 개 매개변수를 사용하는데, 바로 제공할 이니시에이션 큐의 이름입니다. **AMQSAMP4** 샘플 프로그램 실행 시 사용할 수 있는 샘플 초기화 큐(SYSTEM.SAMPLE.TRIGGER)를 정의합니다.

AMQ3TRG4는 이니시에이션 큐에서 가져온 유효한 트리거 메시지마다 IBM i 작업을 제출합니다.

트리거 모니터의 설계

트리거 모니터는 이니시에이션 큐를 열고, 큐에서 메시지를 가지고, 무제한 대기 간격을 지정합니다.

트리거 모니터는 IBM i 작업을 제출하여 트리거 메시지에 지정된 애플리케이션을 시작하고, MQTMC(트리거 메시지의 문자 버전) 구조를 전달합니다. 트리거 메시지의 환경 데이터가 작업 제출 매개변수로 사용됩니다.

마지막으로 프로그램은 이니시에이션 큐를 닫습니다.

▶ IBM i AMQ3SRV4 용 RPG 샘플 트리거 서버 IBM i

AMQ3SRV4는 트리거 서버입니다. 한 개 매개변수를 사용하는데, 바로 제공할 이니시에이션 큐의 이름입니다. AMQSAMP4 샘플 프로그램 실행 시 사용할 수 있는 샘플 초기화 큐(SYSTEM.SAMPLE.TRIGGER)를 정의합니다.

각 트리거 메시지에 대해, AMQ3SRV4는 고유 작업에서 시작 명령을 실행하여 지정된 애플리케이션을 시작합니다.

예 트리거 큐를 사용하여 발행할 명령은 다음과 같습니다.

```
CALL PGM(QMQM/AMQ3SRV4) PARM('Queue Name')
```

여기서 Queue Name의 길이는 48자여야 하며 큐 이름에 필요한 수만큼 공백을 채우면 됩니다. 따라서 SYSTEM.SAMPLE.TRIGGER를 대상 큐로 사용하는 경우, 28개 공백 문자가 필요합니다.

트리거 서버의 디자인

트리거 서버의 설계는 다음 사항을 제외하고 트리거 모니터의 설계와 비슷합니다.

- CICS 및 IBM i 애플리케이션 허용
- 트리거 메시지의 환경 데이터를 사용하지 않음
- 자체 작업 내에서 애플리케이션을 IBM i 호출하거나(또는 CICSSTRUSR을 사용하여 애플리케이션을 CICS 시 작함) 작업을 IBM i 제출하지 않습니다
- 공유 입력을 위해 이니시에이션 큐를 열기 때문에 많은 트리거 서버를 동시에 실행할 수 있음

참고: AMQ3SRV4에 의해 시작된 프로그램은 MQDISC 호출이 트리거 서버를 중지하므로 해당 호출을 사용하지 않아야 합니다. AMQ3SRV4에 의해 시작된 프로그램이 MQCONN 호출을 사용하면 RC2002 이유 코드가 표시됩니다.

▶ IBM i 트리거링 RPG 샘플 프로그램 종료 IBM i

sysrequest 옵션 2(ENDRQS)에 의해 또는 트리거 큐에서 가져오기를 금지하여 트리거 모니터 프로그램을 종료할 수 있습니다.

샘플 트리거 큐를 사용하는 경우, 명령은 다음과 같습니다.

```
CHGMQM QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*NO)
```

참고: 이 큐에서 트리거를 다시 시작하려면 다음 명령을 입력해야 합니다.

```
CHGMQM QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*YES)
```

▶ IBM i 원격 큐를 사용하여 RPG 샘플 실행 IBM i

연결된 메시지 큐 관리자에서 샘플을 실행하여 리모트 큐잉을 보여줄 수 있습니다.

프로그램은 원격 큐 관리자라는 이름의 원격 큐(OTHERSYSTEM.SAMPLE.REMOTEAMQSAMP4)에 대한 로컬 정의를 제공합니다. 이 샘플 정의를 사용하려면 OTHER를 사용할 두 번째 메시지 큐 관리자의 이름으로 변경하십시오. 또한 두 메시지 큐 관리자 사이에 메시지 채널을 설정해야 합니다. 설정 방법에 대한 자세한 내용은 을 참조하십시오 710 페이지의 『메시지 채널에 대한 채널 엑시트 프로그램』.

요청 샘플 프로그램은 송신하는 메시지의 MDRM 필드에 고유의 로컬 큐 관리자 이름을 넣습니다. 조회 및 설정 샘플은 처리하는 요청 메시지의 MDRQ 및 MDRM 필드에 지정된 큐 및 메시지 큐 관리자로 응답 메시지를 송신합니다.

관련 개념

[1042 페이지의 『리모트 큐를 사용하여 샘플 실행』](#)

연결된 큐 관리자의 샘플을 실행하여 리모트 큐잉을 보여줄 수 있습니다.

IBM MQ for z/OS와 함께 전달된 샘플 프로시저 애플리케이션은 MQI(Message Queue Interface)의 일반적인 사용에 대해 설명합니다.

시작하기 전에

일부 샘플 애플리케이션에 기본 패널 구동 인터페이스가 있지만 이러한 인터페이스는 애플리케이션의 룩앤필을 디자인하는 방법을 설명하기 위한 것이 아닙니다. 프로그래밍이 불가능한 터미널에 대해 패널 구동 인터페이스를 디자인하는 방법에 대한 자세한 정보는 SAA 공통 사용자 액세스: 기본 인터페이스 디자인 안내서(SC26-4583) 및 부록(GG22-9508)을 참조하십시오. 이는 애플리케이션 내에서 그리고 다른 애플리케이션에서 모두 일관된 애플리케이션을 설계하는 데 도움이 되는 지침을 제공합니다.

또한 IBM MQ for z/OS는 729 페이지의 [『데이터 변환 엑시트 작성』](#)에서 설명되는 샘플 데이터 변환 엑시트를 제공합니다.

이 태스크 정보

모든 샘플 애플리케이션은 소스 양식으로 제공되며 그 중 일부는 또한 실행 가능한 양식으로 제공됩니다. 소스 모듈에는 프로그램 로직을 설명하는 의사코드(pseudocode)가 포함됩니다.

프로시저

- 다음 링크를 사용하여 샘플 프로그램에 대해 자세히 알아보십시오.
 - [1080 페이지의 『MQI 샘플에 표시된 기능은 다음과 같습니다. z/OS』](#)
 - [1084 페이지의 『z/OS에서 클라이언트 연결을 승인하도록 큐 관리자 구성』](#)
 - [1086 페이지의 『배치 환경용 샘플 애플리케이션 준비 z/OS』](#)
 - [1089 페이지의 『z/OS에서 TSO 환경에 대한 샘플 애플리케이션 준비』](#)
 - [1091 페이지의 『환경에 CICS 대한 샘플 애플리케이션 준비 z/OS』](#)
 - [1094 페이지의 『환경에 IMS 대한 샘플 애플리케이션 준비 z/OS』](#)
 - [1095 페이지의 『z/OS에서의 Put 샘플』](#)
 - [1097 페이지의 『z/OS에서의 Get 샘플』](#)
 - [1100 페이지의 『z/OS에서의 찾아보기 샘플』](#)
 - [1101 페이지의 『z/OS에서의 메시지 인쇄 샘플』](#)
 - [1104 페이지의 『z/OS에서의 큐 속성 샘플』](#)
 - [1106 페이지의 『z/OS에서의 메일 관리자 샘플』](#)
 - [1112 페이지의 『z/OS에서의 신용 조회 샘플』](#)
 - [1122 페이지의 『z/OS에서의 메시지 핸들러 샘플』](#)
 - [1125 페이지의 『z/OS에서의 비동기 Put 샘플』](#)
 - [1126 페이지의 『z/OS의 배치 비동기 이용 샘플』](#)
 - [1128 페이지의 『z/OS의 CICS Asynchronous Consumption and Publish/Subscribe 샘플』](#)
 - [1130 페이지의 『z/OS에서의 발행/구독 샘플』](#)
 - [1132 페이지의 『z/OS에서 메시지 특성 샘플의 설정 및 조회』](#)

관련 태스크

[971 페이지의 『멀티플랫폼용 샘플 절차적 프로그램 사용』](#)

이 샘플 프로시저 프로그램은 제품과 함께 전달됩니다. 샘플은 C 및 COBOL로 작성되며 MQI(Message Queue Interface)의 일반 사용을 보여줍니다.

MQI 샘플에 표시된 기능은 다음과 같습니다. z/OS

각 샘플 애플리케이션에서 시연된 ‘MQI’ 기능에 대한 요약으로, 각 샘플이 작성된 프로그래밍 언어와 실행 환경이 포함됩니다.

z/OS에서의 Put 샘플

Put 샘플은 MQPUT 호출을 사용하여 큐에 메시지를 넣는 방법을 보여줍니다.

애플리케이션은 이러한 MQI 호출을 사용합니다.

- MQCONN
- MQOPEN
- MQPUT
- MQCLOSE
- MQDISC

이 프로그램은 COBOL 과 C로 구현되었으며, 배치 및 CICS 환경에서 실행됩니다. 자세한 내용은 일괄 신청에 대해서는 [1087 페이지의 『배치 Put 및 Get 샘플』](#) 신청에 CICS 대해서는 [1092 페이지의 『CICS Put 및 Get 샘플』](#) 를 참조하십시오.

z/OS에서의 Get 샘플

Get 샘플은 MQGET 호출을 사용하여 큐에서 메시지를 가져오는 방법을 보여줍니다.

애플리케이션은 이러한 MQI 호출을 사용합니다.

- MQCONN
- MQOPEN
- MQGET
- MQCLOSE
- MQDISC

이 프로그램은 COBOL 과 C로 구현되었으며, 배치 및 CICS 환경에서 실행됩니다. 자세한 내용은 일괄 신청에 대해서는 [1087 페이지의 『배치 Put 및 Get 샘플』](#) 신청에 CICS 대해서는 [1092 페이지의 『CICS Put 및 Get 샘플』](#) 를 참조하십시오.

z/OS에서의 찾아보기 샘플

찾아보기 샘플은 찾아보기 옵션을 사용하여 메시지를 찾고, 출력한 후, 큐에서 메시지 사이를 이동하는 방법을 보여줍니다.

애플리케이션은 이러한 MQI 호출을 사용합니다.

- MQCONN
- MQOPEN
- MQGET - 메시지 찾아보기용
- MQCLOSE
- MQDISC

COBOL, 어셈블러, PL/I 및 C 언어로 프로그램이 전달됩니다. 애플리케이션이 배치 환경에서 실행됩니다. 일괄 신청에 대한 자세한 내용은 [1087 페이지의 『배치 찾아보기 샘플』](#) 을 참조하십시오.

z/OS에서의 메시지 인쇄 샘플

메시지 출력 샘플은 해당 메시지 디스크립터의 모든 필드와 함께, 큐에서 메시지를 제거하고 메시지에 데이터를 출력하는 방법을 보여줍니다. 선택적으로 각 메시지와 연관된 메시지 특성을 모두 표시할 수 있습니다. 소스 모듈에서 두 개 행의 주석 문자를 제거하면 큐의 메시지를 제거하는 대신 찾아볼 수 있도록 프로그램을 변경할 수 있

습니다. 이 프로그램은 큐에 메시지를 넣는 애플리케이션에서 문제점을 진단하는 데 유용하게 사용할 수 있습니다.

애플리케이션은 이러한 MQI 호출을 사용합니다.

- MQCONN
- MQOPEN
- MQGET - 큐에서 메시지 제거용(찾아보려는 옵션과 함께)
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

프로그램은 C 언어로 전달됩니다. 애플리케이션이 배치 환경에서 실행됩니다. 일괄 신청에 대한 자세한 내용은 [참조하십시오 1088 페이지의 『배치 출력 메시지 샘플\(C 언어 전용\)』](#).

z/OS에서의 큐 속성 샘플

큐 속성 샘플은 IBM MQ for z/OS 오브젝트 속성의 값을 조회하고 설정하는 방법을 보여줍니다.

애플리케이션은 이러한 MQI 호출을 사용합니다.

- MQOPEN
- MQINQ
- MQSET
- MQCLOSE

프로그램은 COBOL, 어셈블러 및 C 언어로 전달됩니다. 애플리케이션은 CICS 환경에서 실행됩니다. 응용 CICS 프로그램에 대한 자세한 내용은 [참조하십시오 1092 페이지의 『CICS 큐 속성 샘플』](#).

z/OS에서의 메일 관리자 샘플

메일 관리자 샘플은 다음 기술을 선보입니다.

- 알리어스 큐 사용
- 임시 동적 큐를 작성하기 위해 모델 큐 사용
- 응답 대상 큐 사용
- CICS 및 배치 환경에서 동기점 사용
- 시스템 명령 입력 큐에 명령 송신
- 리턴 코드 테스트
- 리모트 큐의 로컬 정의를 사용하고 리모트 큐 관리자에서 이름 지정된 큐에 직접 메시지를 넣어 리모트 큐 관리자에 메시지 송신

애플리케이션은 이러한 MQI 호출을 사용합니다.

- MQCONN
- MQOPEN
- MQPUT1
- MQGET
- MQINQ
- MQCMIT
- MQCLOSE
- MQDISC

애플리케이션의 세 가지 버전이 제공됩니다.

- COBOL로 작성된 CICS 애플리케이션
- COBOL로 작성된 TSO 애플리케이션
- C로 작성된 TSO 애플리케이션

TSO 애플리케이션은 IBM MQ for z/OS 배치 어댑터를 사용하고 일부 ISPF 패널을 포함합니다.

자세한 내용은 TSO 애플리케이션에 대해서는 [를](#), 1092 페이지의 『CICS 메일 관리자 샘플(COBOL 전용)』 1089 페이지의 『TSO 메일 관리자 샘플』 CICS 애플리케이션에 대해서는 [를](#) 참조하십시오.

z/OS에서의 신용 조회 샘플

신용 조회 샘플은 다음 기술을 보여주는 프로그램 스위트입니다.

- 둘 이상의 환경에서 실행되는 애플리케이션 개발
- 임시 동적 큐를 작성하기 위해 모델 큐 사용
- 상관 ID 사용
- 컨텍스트 정보 설정 및 전달
- 메시지 우선순위 및 지속성 사용
- 트리거링을 사용하여 프로그램 시작
- 응답 대상 큐 사용
- 알리어스 큐 사용
- 데드-레터 큐 사용
- 이름 목록 사용
- 리턴 코드 테스트

애플리케이션은 이러한 MQI 호출을 사용합니다.

- MQOPEN
- MQPUT
- MQPUT1
- MQGET - 대기 및 신호 옵션을 사용하여 메시지 찾아보기 및 가져오기 및 특정 메시지 가져오기용
- MQINQ
- MQSET
- MQCLOSE

샘플은 독립형 CICS 애플리케이션으로 실행할 수 있습니다. 그러나 CICS 및 IMS 환경 모두에서 제공한 기능을 사용하는 메시지 큐잉 애플리케이션의 설계 방법을 보여주기 위해, IMS 배치 메시지 처리 프로그램으로 하나의 모듈이 제공됩니다. CICS 프로그램은 C 및 COBOL로 전달됩니다. 단일 IMS 프로그램이 C로 전달됩니다.

자세한 내용은 애플리케이션의 CICS 경우 [를](#) 1093 페이지의 『CICS 신용 조회 샘플』 참조하십시오. 애플리케이션의 IMS 경우 [1095 페이지의 『신용 조회 IMS 샘플에 대한 소스 및 JCL\(C 전용\)』](#) 를 참조하십시오.

메시지 핸들러 샘플 z/OS

메시지 핸들러 샘플은 큐에서 메시지를 찾아보고, 전달하고, 삭제할 수 있습니다.

애플리케이션은 이러한 MQI 호출을 사용합니다.

- MQCONN
- MQOPEN
- MQINQ
- MQPUT1
- MQCMIT

- MQBACK
- MQGET
- MQCLOSE
- MQDISC

프로그램은 C 및 COBOL 프로그래밍 언어로 전달됩니다. 애플리케이션은 TSO에서 실행됩니다. TSO 신청에 대한 자세한 내용은 을 참조하십시오 [1090 페이지의 『TSO 메시지 핸들러 샘플』](#).

z/OS의 분산 큐잉 엑시트 샘플

다음 표는 분산 큐잉 출구 샘플의 소스 프로그램 이름을 나열합니다:

표 177. 분산 큐잉 엑시트 샘플에 대한 소스			
멤버 이름	언어	설명	라이브러리에 제공
CSQ4BAX0	어셈블러	소스 프로그램	SCSQASMS
CSQ4BCX1	C	소스 프로그램	SCSQC37S
CSQ4BCX2	C	소스 프로그램	SCSQC37S
CSQ4BCX4	C	소스 프로그램	SCSQC37S

z/OS에서의 데이터 변환 엑시트 샘플

데이터 변환 엑시트 루틴에 대해 스케레톤이 제공되며, MQXCNCV 호출을 설명하는 IBM MQ와 함께 샘플이 제공됩니다. 다음 표는 데이터 변환 종료 샘플의 소스 프로그램 이름을 나열합니다:

표 178. 데이터 변환 엑시트 샘플에 대한 소스(어셈블러 언어 전용)		
멤버 이름	설명	라이브러리에 제공
CSQ4BAX8	소스 프로그램	SCSQASMS
CSQ4BAX9	소스 프로그램	SCSQASMS
CSQ4CAX9	소스 프로그램	SCSQASMS

참고: 소스 프로그램은 CSQASTUB으로 링크 편집됩니다.

자세한 정보는 [729 페이지의 『데이터 변환 엑시트 작성』](#)의 내용을 참조하십시오.

z/OS에서의 발행/구독 샘플

발행/구독 샘플 프로그램은 IBM MQ의 발행 및 구독 기능의 사용을 보여줍니다. IBM MQ 발행/구독 인터페이스에 프로그래밍하는 방법을 보여주는 네 개의 C 및 두 개의 COBOL 프로그래밍 언어 샘플 프로그램이 있습니다.

애플리케이션은 이러한 MQI 호출을 사용합니다.

- MQCONN
- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

발행/구독 샘플 프로그램은 C 및 COBOL 프로그래밍 언어로 전달됩니다. 샘플 애플리케이션은 배치 환경에서 실행됩니다. 배치 애플리케이션에 대한 자세한 내용은 [게시/구독 샘플](#)을 참조하십시오.

관련 개념

[1143 페이지의 『C 언어로 ‘MQI’ 앱 개발하기』](#)

C로 IBM MQ 프로그램을 코딩할 때 다음 섹션의 정보를 참고하십시오.

[1162 페이지의 『COBOL 에서 MQI 앱 개발하기』](#)

COBOL로 IBM MQ 프로그램을 코딩하는 경우 다음 섹션의 정보를 참고하십시오.

[1177 페이지의 『어셈블리어로 ‘MQI’ 앱 개발하기』](#)

System/390 어셈블리 언어로 작성된 IBM MQ for z/OS MQI 프로그램에 대한 정보.

[1192 페이지의 『PL/I 에서 MQI 앱 개발하기』](#)

PL/I로 IBM MQ를 코딩하는 경우 유용한 정보입니다.

z/OS에서 클라이언트 연결을 승인하도록 큐 관리자 구성

샘플 애플리케이션을 실행하기 전에 먼저 큐 관리자를 작성해야 합니다. 그런 다음 클라이언트 모드로 실행 중인 애플리케이션으로부터 수신되는 연결 요청을 안전하게 승인하도록 큐 관리자를 구성할 수 있습니다.

시작하기 전에

큐 관리자가 이미 존재하며 시작되었는지 확인하십시오. MQSC 명령을 발행하여 채널 인증 레코드가 이미 사용으로 설정되어 있는지 판별하십시오.

```
DISPLAY QMGR CHLAUTH
```

중요사항: 이 태스크에서는 채널 인증 레코드를 사용할 수 있는 것으로 예상합니다. 이것이 기타 사용자 및 애플리케이션에서 사용하는 큐 관리자인 경우 이 설정을 변경하면 기타 모든 사용자 및 애플리케이션에 영향을 미칩니다. 큐 관리자가 채널 인증 레코드를 이용하지 않는 경우에는 4단계를 MCAUSER를 1084 페이지의 『1』 단계에서 연결될 *non-privileged-user-id*로 설정하는 대체 인증 방법(예: 보안 엑시트)으로 대체할 수 있습니다.

애플리케이션이 사용할 것으로 예상되는 채널 이름을 알아야 애플리케이션이 해당 채널을 사용하도록 허용할 수 있습니다. 애플리케이션이 사용할 것으로 예상되는 오브젝트(예: 큐 또는 토픽)를 알아야 애플리케이션이 해당 오브젝트를 사용하도록 허용할 수 있습니다.

이 태스크 정보

이 태스크는 큐 관리자에 연결하는 클라이언트 애플리케이션에 사용될 권한이 없는 사용자 ID를 작성합니다. 이 사용자 ID를 사용하면 클라이언트 애플리케이션이 필요한 큐 및 필요한 채널만을 사용할 수 있도록 액세스가 부여됩니다.

프로시저

1. 큐 관리자가 실행 중인 시스템에서 사용자 ID를 확보하십시오.

이 태스크의 경우 이 사용자 ID는 권한이 있는 관리자가 아니어야 합니다. 이 사용자 ID를 사용하면 큐 관리자에서 클라이언트 연결을 실행할 수 있는 권한이 부여됩니다.

2. 리스너 프로그램을 시작하십시오.

a) 채널 시작기가 시작되었는지 확인하십시오. 그렇지 않다면, 명령어를 **START CHINIT** 실행하여 시작하십시오.

b) 다음 명령을 발행하여 리스너 프로그램을 시작하십시오.

```
START LISTENER TRPTYPE(TCP) PORT(nnnn)
```

여기서 *nnnn*은 사용자가 선택한 포트 번호입니다.

3. 애플리케이션에서 SYSTEM.DEF.SVRCONN을 사용하는 경우 이 채널은 이미 정의되어 있습니다. 애플리케이션에서 다른 채널을 사용하는 경우라면 MQSC 명령을 발행하여 작성하십시오.

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


*channel-name*은 채널의 이름입니다.

4. MQSC 명령을 발행함으로써 클라이언트 시스템의 IP 주소에서만 채널을 사용하도록 허용하는 채널 인증 규칙을 작성하십시오.

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

여기서,

*channel-name*은 채널의 이름입니다.

*client-machine-IP-address*는 클라이언트 시스템의 IP 주소입니다. 샘플 클라이언트 애플리케이션이 큐 관리자와 동일한 시스템에서 실행 중인 경우 애플리케이션이 'localhost'를 사용하여 연결할 경우 IP 주소 '127.0.0.1'을 사용하십시오. 서로 다른 여러 클라이언트 시스템에 연결하려는 경우, 단일 IP 주소 대신 패턴 또는 범위를 사용할 수 있습니다. 자세한 내용은 일반 IP 주소의 내용을 참조하십시오.

*non-privileged-user-id*는 1084 페이지의 『1』 단계에서 획득한 사용자 ID입니다.

5. 애플리케이션에서 SYSTEM.DEFAULT.LOCAL.QUEUE를 사용하는 경우 이 큐는 이미 정의되어 있습니다. 애플리케이션에서 다른 큐를 사용하는 경우라면 MQSC 명령을 발행하여 작성하십시오.

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

여기서 *queue-name*은 큐의 이름입니다.

6. 큐 관리자에 연결하고 조회할 수 있는 액세스를 부여하십시오.

- a) 채널 시작기가 시작되었는지 확인하십시오. 시작되지 않았으면 START CHINIT 명령을 실행하여 채널 시작기를 시작하십시오.
- b) TCP 리스너를 시작하십시오. 예를 들어, 다음 명령을 발행하십시오.

```
START LISTENER TRPTYPE(TCP) PORT(nnnn)
```

여기서 *nnnn*은 사용자가 선택한 포트 번호입니다.

7. 애플리케이션이 포인트-투-포인트 애플리케이션인 경우 즉, 큐를 사용하는 경우 다음 MQSC 명령을 발행함으로써 사용될 사용자 ID별로 큐를 사용하여 메시지를 조회, 삽입 및 가져올 수 있도록 액세스를 부여하십시오.

RACF 명령을 발행하십시오.

```
RDEFINE MQQUEUE qmgr-name.QUEUE. queue-name UACC(NONE)  
PERMIT qmgr-name.QUEUE. queue-name CLASS(MQQUEUE) ID(non-privileged-user-id) ACCESS(UPDATE)
```

여기서,

*qmgr-name*은 큐 관리자의 이름입니다.

*queue-name*은 큐의 이름입니다.

*non-privileged-user-id*는 1084 페이지의 『1』 단계에서 획득한 사용자 ID입니다.

8. 애플리케이션이 발행/구독 애플리케이션인 경우, 즉, 토픽을 사용하는 경우, 다음 RACF 명령을 발행하여 사용할 ID별로 토픽을 사용하여 발행 및 구독할 수 있도록 허용하는 액세스를 부여하십시오.

```
RDEFINE MQTOPIC qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC UACC(NONE)  
PERMIT qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)  
ACCESS(UPDATE)  
  
RDEFINE MQTOPIC qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC UACC(NONE)  
PERMIT qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)  
ACCESS(UPDATE)
```

여기서,

*qmgr-name*은 큐 관리자의 이름입니다.

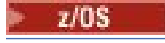
*non-privileged-user-id*는 1084 페이지의 『1』 단계에서 획득한 사용자 ID입니다.

이 사용자 ID를 사용하면 토픽 트리에 있는 모든 토픽에 *non-privileged-user-id* 액세스가 부여되며, 또는 **DEFINE TOPIC**을 사용하여 토픽 오브젝트를 정의한 후 해당 토픽 오브젝트가 참조하는 토픽 트리의 파티에만 액세스를 부여할 수 있습니다. 자세한 정보는 [토픽에 대한 사용자 액세스 제어](#)를 참조하십시오.

다음에 수행할 작업

이제 클라이언트 애플리케이션은 큐 관리자에 연결하고 큐를 사용하여 메시지를 삽입하거나 가져올 수 있습니다.

관련 개념

 [객체에 IBM MQ 대한 작업 권한 z/OS](#)

관련 참조

[SET CHLAUTH](#)

[DEFINE CHANNEL](#)

[DEFINE QLOCAL](#)

[SET AUTHREC](#)

배치 환경용 샘플 애플리케이션 준비 z/OS

.z/OSNET 환경에서 배치 환경에서 실행되는 샘플 애플리케이션을 준비하는 방법

이 태스크 정보

다음 샘플 응용 프로그램은 배치 환경에서 사용할 수 z/OS 있습니다:

- [1087 페이지의 『배치 Put 및 Get 샘플』](#)
- [1087 페이지의 『배치 찾아보기 샘플』](#)
- [1088 페이지의 『배치 출력 메시지 샘플\(C 언어 전용\)』](#)
- [1088 페이지의 『발행/구독 샘플』](#)
- [1088 페이지의 『기타 샘플』](#)

프로시저

배치 환경에서 실행하는 샘플 애플리케이션을 준비하려면 배치 IBM MQ for z/OS 애플리케이션을 빌드할 때와 동일한 단계를 수행하십시오.

자세한 정보는 [958 페이지의 『배치 애플리케이션 z/OS 구축 z/OS』](#)의 내용을 참조하십시오.

또는 샘플의 실행 가능한 형태가 있는 경우, thlqual.SCSQLOAD 로드 라이브러리에서 실행할 수 있습니다.

참고: 찾아보기 샘플의 어셈블러 언어 버전은 데이터 제어 블록(DCB)을 사용하므로 RMODE(24)를 사용하여 링크 편집해야 합니다.

사용할 라이브러리 멤버가 [1087 페이지의 『배치 Put 및 Get 샘플』](#), [1087 페이지의 『배치 찾아보기 샘플』](#), [1088 페이지의 『배치 출력 메시지 샘플\(C 언어 전용\)』](#) 및 [1088 페이지의 『발행/구독 샘플』](#)에 나열됩니다.

사용하려는 샘플을 위해 제공된 실행 JCL을 편집해야 합니다([1087 페이지의 『배치 Put 및 Get 샘플』](#), [1087 페이지의 『배치 찾아보기 샘플』](#), [1088 페이지의 『배치 출력 메시지 샘플\(C 언어 전용\)』](#) 및 [1088 페이지의 『발행/구독 샘플』](#) 참조).

제공된 JCL의 PARM 명령문에는 수정해야 하는 여러 매개변수가 포함되어 있습니다. C 샘플 프로그램을 실행하려면, 공백으로 매개변수를 구분하십시오. 어셈블러, COBOL 및 PL/I 샘플 프로그램을 실행하려면, 쉼표로 매개변수를 구분하십시오. 예를 들어, 큐 관리자의 이름이 CSQ1이고 LOCALQ1이라는 큐로 애플리케이션을 실행하려는 경우, COBOL, PL/I 및 어셈블러 언어 JCL에서 PARM 명령문은 다음과 같아야 합니다.

```
PARM=(CSQ1,LOCALQ1)
```

C 언어 JCL에서 PARM 명령문은 다음과 같아야 합니다.

```
PARM=( 'CSQ1 LOCALQ1 ' )
```

이제 작업을 제출할 준비가 되었습니다.

▶ z/OS z/OS에서 샘플 배치 애플리케이션의 이름

샘플 배치 애플리케이션에 제공된 프로그램에 대한 요약입니다.

배치 애플리케이션 프로그램은 다음 섹션의 표에 요약되어 있습니다:

- 1087 페이지의 『배치 Put 및 Get 샘플』
- 1087 페이지의 『배치 찾아보기 샘플』
- 1088 페이지의 『배치 출력 메시지 샘플(C 언어 전용)』
- 1088 페이지의 『발행/구독 샘플』
- 1088 페이지의 『기타 샘플』

배치 Put 및 Get 샘플

표 179. 배치 Put 및 Get 샘플				
멤버 이름	언어	설명	라이브러리에 제공된 소스 파일	라이브러리에 제공된 실행 파일
CSQ4BCJ1	C	Get 소스 프로그램	SCSQC37S	SCSQLOAD
CSQ4BCK1	C	Put 소스 프로그램	SCSQC37S	SCSQLOAD
CSQ4BCJR	C	CSQ4BCJ1 및 CSQ4BCK1에 대한 샘플 실행 JCL	SCSQPROC	없음
CSQ4BVJ1	COBOL	Get 소스 프로그램	SCSQCOBS	SCSQLOAD
CSQ4BVK1	COBOL	Put 소스 프로그램	SCSQCOBS	SCSQLOAD
CSQ4BVJR	COBOL	CSQ4BVJ1 및 CSQ4BVK1에 대한 샘플 실행 JCL	SCSQPROC	없음

배치 찾아보기 샘플

표 180. 배치 찾아보기 샘플				
멤버 이름	언어	설명	라이브러리에 제공된 소스 파일	라이브러리에 제공된 실행 파일
CSQ4BVA1	COBOL	소스 프로그램	SCSQCOBS	SCSQLOAD
CSQ4BVAR	COBOL	CSQ4BVA1에 대한 샘플 실행 JCL	SCSQPROC	없음
CSQ4BAA1	어셈블러	소스 프로그램	SCSQASMS	SCSQLOAD
CSQ4BAAR	어셈블러	CSQ4BAA1에 대한 샘플 실행 JCL	SCSQPROC	없음
CSQ4BCA1	C	소스 프로그램	SCSQC37S	SCSQLOAD
CSQ4BCAR	C	CSQ4BCA1에 대한 샘플 실행 JCL	SCSQPROC	없음
CSQ4BPA1	PL/I	소스 프로그램	SCSQPLIS	SCSQLOAD
CSQ4BPAR	PL/I	CSQ4BPA1에 대한 샘플 실행 JCL	SCSQPROC	없음

배치 출력 메시지 샘플(C 언어 전용)

표 181. 배치 출력 메시지 샘플(C 언어 전용)			
멤버 이름	설명	라이브러리에 제공된 소스 파일	라이브러리에 제공된 실행 파일
CSQ4BCG1	소스 프로그램	SCSQC37S	SCSQLOAD
CSQ4BCGR	CSQ4BCG1에 대한 샘플 실행 JCL	SCSQPROC	없음
CSQ4BCL1	찾아보기 소스 프로그램	SCSQC37S	SCSQLOAD
CSQ4BCLR	CSQ4BCL1에 대한 샘플 실행 JCL	SCSQPROC	없음

발행/구독 샘플

표 182. 발행/구독 샘플					
멤버 이름	언어	설명	라이브러리에 제공된 소스 파일	SCSQPROC의 JCL	라이브러리에 제공된 실행 파일
CSQ4BCP1	C	주제 소스 프로그램에 발행	SCSQC37S	CSQ4BCPP	SCSQLOAD
CSQ4BCP2	C	주제 및 Get 메시지 소스 프로그램 구독	SCSQC37S	CSQ4BCPS	SCSQLOAD
CSQ4BCP3	C	사용자 제공 대상을 사용하는 주제 및 Get 메시지 소스 프로그램 구독	SCSQC37S	CSQ4BCPD	SCSQLOAD
CSQ4BCP4	C	확장 옵션을 사용하는 주제 및 Get 메시지 소스 프로그램 구독	SCSQC37S	CSQ4BCPE	SCSQLOAD
CSQ4BVP1	COBOL	주제 소스 프로그램에 발행	SCSQCOBS	CSQ4BVPP	SCSQLOAD
CSQ4BVP2	COBOL	주제 및 Get 메시지 소스 프로그램 구독	SCSQCOBS	CSQ4BVPS	SCSQLOAD

기타 샘플

표 183. 기타 샘플					
멤버 이름	언어	설명	라이브러리에 제공된 소스 파일	SCSQPROC의 JCL	라이브러리에 제공된 실행 파일
CSQ4BCS1	C	비동기 이용 소스 프로그램	SCSQC37S	CSQ4BCSC	SCSQLOAD
CSQ4BCS2	C	비동기 Put 및 검사 상태 소스 프로그램	SCSQC37S	CSQ4BCSP	SCSQLOAD
CSQ4BCM1	C	조회 메시지 특성 소스 프로그램	SCSQC37S	CSQ4BCMP	SCSQLOAD
CSQ4BCM2	C	설정 메시지 특성 소스 프로그램	SCSQC37S	CSQ4BCMP	SCSQLOAD

▶ z/OS z/OS에서 TSO 환경에 대한 샘플 애플리케이션 준비

TSO 환경에서 실행되는 샘플 z/OS 애플리케이션을 준비하는 방법.

이 태스크 정보

다음 샘플 응용 프로그램은 TSO 환경에서 사용할 수 z/OS 있습니다:

- 1089 페이지의 『TSO 메일 관리자 샘플』
- 1090 페이지의 『TSO 메시지 핸들러 샘플』

프로시저

TSO 환경에서 실행하는 샘플 애플리케이션을 준비하려면 배치 IBM MQ for z/OS 애플리케이션을 빌드할 때와 동일한 단계를 수행하십시오.

자세한 정보는 958 페이지의 『배치 애플리케이션 z/OS 구축 z/OS』의 내용을 참조하십시오. 사용할 라이브러리 멤버가 1089 페이지의 『TSO 메일 관리자 샘플』에 나열됩니다.

또는 샘플의 실행 가능한 형태가 있는 경우, thlqual.SCSQLOAD 로드 라이브러리에서 실행할 수 있습니다.

메일 관리자 샘플 애플리케이션의 경우 사용하는 큐가 시스템에서 사용 가능한지 확인하십시오.

thlqual.SCSQPROC(CSQ4CVD) 멤버에 정의됩니다. 이러한 큐를 항상 사용 가능하게 하려면, CSQINP2 초기화 입력 데이터 세트에 이러한 멤버를 추가하거나 CSQUTIL 프로그램을 사용하여 이러한 큐 정의를 로드할 수 있습니다.

▶ z/OS z/OS에서 샘플 TSO 애플리케이션의 이름

각 샘플 TSO 애플리케이션에 제공된 프로그램의 이름과 소스, JCL 및 실행 파일(메시지 핸들러 샘플의 경우에만)이 상주하는 라이브러리의 이름에 대한 정보입니다.

TSO 애플리케이션 프로그램에 대해서는 다음 표에서 설명합니다.

- 1089 페이지의 『TSO 메일 관리자 샘플』
- 1090 페이지의 『TSO 메시지 핸들러 샘플』

이러한 샘플은 ISPF 패널을 사용합니다. 따라서 프로그램을 링크 편집할 때 ISPF 스텝, ISPLINK를 반드시 포함해야 합니다.

TSO 메일 관리자 샘플

표 184. TSO 메일 관리자 샘플			
멤버 이름	언어	설명	라이브러리에 제공된 소스 파일
CSQ4CVD	독립적	IBM MQ for z/OS 오브젝트 정의	SCSQPROC
CSQ40	독립적	ISPF 메시지	SCSQMSGGE
CSQ4RVD1	COBOL	CSQ4TVD1을 시작하기 위한 CLIST	SCSQCLST
CSQ4TVD1	COBOL	메뉴 프로그램을 위한 소스 프로그램	SCSQCOBS
CSQ4TVD2	COBOL	메일 가져오기 프로그램을 위한 소스 프로그램	SCSQCOBS
CSQ4TVD4	COBOL	메일 송신 프로그램을 위한 소스 프로그램	SCSQCOBS
CSQ4TVD5	COBOL	별명 프로그램을 위한 소스 프로그램	SCSQCOBS

표 184. TSO 메일 관리자 샘플 (계속)			
멤버 이름	언어	설명	라이브러리에 제공된 소스 파일
CSQ4VDP1-6	COBOL	패널 정의	SCSQPNLA
CSQ4VD0	COBOL	데이터 정의	SCSQCOBC
CSQ4VD1	COBOL	데이터 정의	SCSQCOBC
CSQ4VD2	COBOL	데이터 정의	SCSQCOBC
CSQ4VD4	COBOL	데이터 정의	SCSQCOBC
CSQ4RCD1	C	CSQ4TCD1을 시작하기 위한 CLIST	SCSQCLST
CSQ4TCD1	C	메뉴 프로그램을 위한 소스 프로그램	SCSQC37S
CSQ4TCD2	C	메일 가져오기 프로그램을 위한 소스 프로그램	SCSQC37S
CSQ4TCD4	C	메일 송신 프로그램을 위한 소스 프로그램	SCSQC37S
CSQ4TCD5	C	별명 프로그램을 위한 소스 프로그램	SCSQC37S
CSQ4CDP1-6	C	패널 정의	SCSQPNLA
CSQ4TC0	C	포함 파일	SCSQC370

TSO 메시지 핸들러 샘플

표 185. TSO 메시지 핸들러 샘플				
멤버 이름	언어	설명	라이브러리에 제공된 소스 파일	라이브러리에 제공된 실행 파일
CSQ4TCH0	C	데이터 정의	SCSQC370	없음
CSQ4TCH1	C	소스 프로그램	SCSQC37S	SCSQLOAD
CSQ4TCH2	C	소스 프로그램	SCSQC37S	SCSQLOAD
CSQ4TCH3	C	소스 프로그램	SCSQC37S	SCSQLOAD
CSQ4RCH1	C 및 COBOL	CSQ4TCH1 또는 CSQ4TVH1을 시작하기 위한 CLIST	SCSQCLST	없음
CSQ4CHP1	C 및 COBOL	패널 정의	SCSQPNLA	없음
CSQ4CHP2	C 및 COBOL	패널 정의	SCSQPNLA	없음
CSQ4CHP3	C 및 COBOL	패널 정의	SCSQPNLA	없음
CSQ4CHP9	C 및 COBOL	패널 정의	SCSQPNLA	없음
CSQ4TVH0	COBOL	데이터 정의	SCSQCOBC	없음
CSQ4TVH1	COBOL	소스 프로그램	SCSQCOBS	SCSQLOAD
CSQ4TVH2	COBOL	소스 프로그램	SCSQCOBS	SCSQLOAD
CSQ4TVH3	COBOL	소스 프로그램	SCSQCOBS	SCSQLOAD

▶ z/OS 환경에 CICS 대한 샘플 애플리케이션 준비 z/OS

환경에서 CICS 실행되는 샘플 z/OS 애플리케이션을 준비하는 방법.

시작하기 전에

CICS 샘플 프로그램을 실행하기 전에, 32702의 LOGMODE를 사용하여 CICS에 로그인하십시오. 이는 샘플 프로그램이 3270 모드 2 화면을 사용하도록 작성되었기 때문입니다.

이 태스크 정보

다음 샘플 응용 프로그램은 환경에서 CICS 사용할 수 z/OS 있습니다:

- [1092 페이지의 『CICS Put 및 Get 샘플』](#)
- [1092 페이지의 『CICS 큐 속성 샘플』](#)
- [1092 페이지의 『CICS 메일 관리자 샘플\(COBOL 전용\)』](#)
- [1093 페이지의 『CICS 신용 조회 샘플』](#)
- [1094 페이지의 『CICS 비동기 이용 및 발행/구독 샘플』](#)

프로시저

1. BMS 화면 정의 소스를 조립하여 샘플에 대한 기호형 설명 맵과 실제 화면 맵을 작성하십시오 ([thlqual.SCSQMAPS](#)에 제공됨, 여기서 **thlqual**은 설치에서 사용된 상위 레벨 규정자임).
맵의 이름을 지정하는 경우 BMS 화면 정의 소스의 이름(Put 및 Get 샘플 프로그램에는 사용 불가능)을 사용하지만 해당 이름의 마지막 문자를 생략하십시오.
2. CICS IBM MQ for z/OS 애플리케이션을 빌드하는 경우 수행하는 동일한 단계를 수행하십시오.
다음 단계가 961 페이지의 『z/OS에서 CICS 애플리케이션 빌드』에 나열됩니다. 사용할 라이브러리 멤버가 [1092 페이지의 『CICS Put 및 Get 샘플』](#), [1092 페이지의 『CICS 큐 속성 샘플』](#), [1092 페이지의 『CICS 메일 관리자 샘플\(COBOL 전용\)』](#) 및 [1093 페이지의 『CICS 신용 조회 샘플』](#)에 나열됩니다.
또는 샘플의 실행 가능한 형태가 있는 경우, **thlqual.SCSQCICS** 로드 라이브러리에서 실행할 수 있습니다.
3. CICS 시스템 정의(CSD) 데이터 세트를 업데이트하여 맵 세트, 프로그램 및 CICS에 대한 트랜잭션을 식별하십시오. 필요한 정의는 멤버 **thlqual.SCSQPROC(CSQ4S100)**에 있습니다.
이 작업을 수행하는 방법에 대한 안내는 제품 CICS Transaction Server 설명서의 z/OS4.1 해당 섹션을 *CICS-IBM MQ* 어댑터 참조하십시오. 위치: [CICS Transaction Server for 4.1z/OS](#), [The CICS-IBM MQ 어댑터](#).
참고: 신용 조회 샘플 애플리케이션의 경우 샘플이 사용하는 VSAM 데이터 세트를 이미 작성하지 않은 경우 이 단계에서 오류 메시지를 수신합니다.
4. 신용 조회 및 메일 관리자 샘플 애플리케이션의 경우 애플리케이션에서 사용하는 큐가 시스템에서 사용 가능한지 확인하십시오.
신용 조회 샘플에서는, COBOL의 경우 멤버 **thlqual.SCSQPROC(CSQ4CVB)** 및 C의 경우 **thlqual.SCSQPROC(CSQ4CCB)**에 이러한 큐가 정의됩니다. 메일 관리자 샘플의 경우에는, 멤버 **thlqual.SCSQPROC(CSQ4CVD)**에 큐가 정의됩니다. 이러한 큐를 항상 사용 가능하게 하려면, **CSQINP2** 초기화 입력 데이터 세트에 이러한 멤버를 추가하거나 **CSQUTIL** 프로그램을 사용하여 이러한 큐 정의를 로드할 수 있습니다.
큐 속성 샘플 애플리케이션의 경우 다른 샘플 애플리케이션에 제공되는 하나 이상의 큐를 사용할 수 있습니다. 또는 고유 큐를 사용할 수 있습니다. 그러나 제공되는 양식에서, 이 샘플은 이름의 처음 8바이트에 **CSQ4SAMP** 문자가 있는 큐에서만 작동합니다.

▶ z/OS 샘플 CICS 애플리케이션의 이름 z/OS

이 주제는 샘플 CICS 애플리케이션을 위해 제공된 프로그램의 요약을 제공합니다.

CICS 애플리케이션 프로그램이 다음 표에 요약됩니다.

- [1092 페이지의 『CICS Put 및 Get 샘플』](#)
- [1092 페이지의 『CICS 큐 속성 샘플』](#)

- 1092 페이지의 『CICS 메일 관리자 샘플(COBOL 전용)』
- 1093 페이지의 『CICS 신용 조회 샘플』
- 1094 페이지의 『CICS 비동기 이용 및 발행/구독 샘플』

CICS Put 및 Get 샘플

표 186. CICS Put 및 Get 샘플				
멤버 이름	언어	설명	라이브러리에 제공된 소스 파일	라이브러리에 제공된 실행 파일
CSQ4CCK1	C	Put 소스 프로그램	SCSQC37S	SCSQCICS
CSQ4CCJ1	C	Get 소스 프로그램	SCSQC37S	SCSQCICS
CSQ4CVJ1	COBOL	Get 소스 프로그램	SCSQCOBS	SCSQCICS
CSQ4CVK1	COBOL	Put 소스 프로그램	SCSQCOBS	SCSQCICS
CSQ4S100	독립적	CICS 시스템 정의 데이터 세트	SCSQPROC	없음

CICS 큐 속성 샘플

표 187. CICS 큐 속성 샘플				
멤버 이름	언어	설명	라이브러리에 제공된 소스 파일	라이브러리에 제공된 실행 파일
CSQ4CVC1	COBOL	소스 프로그램	SCSQCOBS	SCSQCICS
CSQ4VMSG	COBOL	메시지 정의	SCSQCOBC	없음
CSQ4VCMS	COBOL	BMS 화면 정의	SCSQMAPS	SCSQCICS (CSQ4ACM으로 이름 지정된)
CSQ4CAC1	어셈블러	소스 프로그램	SCSQASMS	SCSQCICS
CSQ4AMSG	어셈블러	메시지 정의	SCSQMACS	없음
CSQ4ACMS	어셈블러	BMS 화면 정의	SCSQMAPS	SCSQCICS (CSQ4ACM으로 이름 지정된)
CSQ4CCC1	C	소스 프로그램	SCSQC37S	SCSQCICS
CSQ4CMMSG	C	메시지 정의	SCSQC370	없음
CSQ4CCMS	C	BMS 화면 정의	SCSQMAPS	SCSQCICS (CSQ4ACM으로 이름 지정된)
CSQ4S100	독립적	CICS 시스템 정의 데이터 세트	SCSQPROC	없음

CICS 메일 관리자 샘플(COBOL 전용)

표 188. CICS 메일 관리자 샘플(COBOL 전용)		
멤버 이름	설명	라이브러리에 제공된 소스 파일
CSQ4CVD	IBM MQ for z/OS 오브젝트 정의	SCSQPROC

표 188. CICS 메일 관리자 샘플(COBOL 전용) (계속)

멤버 이름	설명	라이브러리에 제공된 소스 파일
CSQ4CVD1	메뉴 프로그램을 위한 소스	SCSQCOBS
CSQ4CVD2	메일 가져오기 프로그램을 위한 소스	SCSQCOBS
CSQ4CVD3	메시지 표시 프로그램을 위한 소스	SCSQCOBS
CSQ4CVD4	메일 송신 프로그램을 위한 소스	SCSQCOBS
CSQ4CVD5	별명 프로그램을 위한 소스	SCSQCOBS
CSQ4VDMS	BMS 화면 정의 소스	SCSQMAPS
CSQ4S100	CICS 시스템 정의 데이터 세트	SCSQPROC
CSQ4VD0	데이터 정의	SCSQCOBC
CSQ4VD3	데이터 정의	SCSQCOBC
CSQ4VD4	데이터 정의	SCSQCOBC

CICS 신용 조회 샘플

표 189. CICS 신용 조회 샘플

멤버 이름	언어	설명	라이브러리에 제공된 소스 파일
CSQ4CVB	독립적	IBM MQ 오브젝트 정의	SCSQPROC
CSQ4CCB	독립적	IBM MQ 오브젝트 정의	SCSQPROC
CSQ4CVB1	COBOL	사용자 인터페이스 프로그램을 위한 소스	SCSQCOBS
CSQ4CVB2	COBOL	신용 애플리케이션 관리자를 위한 소스	SCSQCOBS
CSQ4CVB3	COBOL	당좌 예금 계좌 프로그램을 위한 소스	SCSQCOBS
CSQ4CVB4	COBOL	분산 프로그램을 위한 소스	SCSQCOBS
CSQ4CVB5	COBOL	에이전시 조회 프로그램을 위한 소스	SCSQCOBS
CSQ4CCB1	C	사용자 인터페이스 프로그램을 위한 소스	SCSQ37S
CSQ4CCB2	C	신용 애플리케이션 관리자를 위한 소스	SCSQ37S
CSQ4CCB3	C	당좌 예금 계좌 프로그램을 위한 소스	SCSQ37S
CSQ4CCB4	C	분산 프로그램을 위한 소스	SCSQ37S
CSQ4CCB5	C	에이전시 조회 프로그램을 위한 소스	SCSQ37S
CSQ4CB0	C	포함 파일	SCSQ370
CSQ4CBMS	C	BMS 화면 정의 소스	SCSQMAPS
CSQ4VBMS	COBOL	BMS 화면 정의 소스	SCSQMAPS
CSQ4VB0	COBOL	데이터 정의	SCSQCOBC
CSQ4VB1	COBOL	데이터 정의	SCSQCOBC
CSQ4VB2	COBOL	데이터 정의	SCSQCOBC
CSQ4VB3	COBOL	데이터 정의	SCSQCOBC

표 189. CICS 신용 조회 샘플 (계속)			
멤버 이름	언어	설명	라이브러리에 제공된 소스 파일
CSQ4VB4	COBOL	데이터 정의	SCSQCOBC
CSQ4VB5	COBOL	데이터 정의	SCSQCOBC
CSQ4VB6	COBOL	데이터 정의	SCSQCOBC
CSQ4VB7	COBOL	데이터 정의	SCSQCOBC
CSQ4VB8	COBOL	데이터 정의	SCSQCOBC
CSQ4BAQ	독립적	VSAM 데이터 세트를 위한 소스	SCSQPROC
CSQ4FILE	독립적	CSQ4CVB3에서 사용한 VSAM 데이터 세트를 빌드하기 위한 JCL	SCSQPROC
CSQ4S100	독립적	CICS 시스템 정의 데이터 세트	SCSQPROC

CICS 비동기 이용 및 발행/구독 샘플

표 190. CICS 비동기 이용 및 발행/구독 샘플		
멤버 이름	설명	라이브러리에 제공된 소스 파일
CSQ4CVCN	단순 메시지 이용 프로그램을 위한 소스	SCSQCOBS
CSQ4CVCT	제어 메시지 이용 프로그램을 위한 소스	SCSQCOBS
CSQ4CVEV	이벤트 핸들러 프로그램을 위한 소스	SCSQCOBS
CSQ4CVPT	메시지 넣기 클라이언트 프로그램을 위한 소스	SCSQCOBS
CSQ4CVRG	등록 클라이언트 프로그램을 위한 소스	SCSQCOBS
CSQ4S100	CICS 시스템 정의 데이터 세트	SCSQPROC

▶ z/OS 환경에 IMS 대한 샘플 애플리케이션 준비 z/OS

신용 조회 샘플 애플리케이션의 일부를 에서 실행 가능한 환경으로 IMSz/OS 준비하는 방법.

이 태스크 정보

샘플 애플리케이션에는 신용 조회 IMS 샘플(C 언어 전용 z/OS)의 소스 코드와 JCL이 포함됩니다. 자세한 정보는 1095 페이지의 『신용 조회 IMS 샘플에 대한 소스 및 JCL(C 전용)』의 내용을 참조하십시오.

프로시저

1. 1091 페이지의 『환경에 CICS 대한 샘플 애플리케이션 준비 z/OS』에 설명되어 있는 단계를 수행하십시오.
2. IMS IBM MQ for z/OS 애플리케이션을 빌드하는 경우 수행하는 동일한 단계를 수행하십시오.
다음 단계가 962 페이지의 『IMS(BMP 또는 MPP) 애플리케이션 빌드』에 나열됩니다. 사용할 라이브러리 멤버가 1095 페이지의 『신용 조회 IMS 샘플에 대한 소스 및 JCL(C 전용)』에 나열됩니다.
3. IMS에 대한 애플리케이션 프로그램 및 데이터베이스를 식별하십시오.

이를 사용으로 설정하기 위해 PSBGEN, DBDGEN, ACB 정의, IMSGEN 및 IMSDALOC 명령문과 함께 샘플이 제공됩니다.

- 이러한 목적으로 제공된 샘플 JCL을 조정 및 실행하여 CSQ4CA 데이터베이스를 로드하십시오(CSQ4ILDB). 이 JCL은 CSQ4BAQ 파일의 데이터를 포함하는 데이터베이스를 로드합니다. 데이터베이스 CSQ4CA에 대한 DD 명령문으로 IMS 제어 영역을 업데이트하십시오.
- 이러한 목적으로 제공된 샘플 JCL을 조정 및 실행하여 당좌 예금 계좌 프로그램을 배치 메시지 처리(BMP) 프로그램으로 시작하십시오.
이 JCL은 배치 지향 BMP 프로그램을 시작합니다. 메시지 지향성 BMP 프로그램으로 프로그램을 실행하려면, IN= 명령문을 포함하는 JCL에서 행의 주석 문자를 제거하십시오.

z/OS 샘플 IMS 애플리케이션의 이름 z/OS

이 정보는 신용 조회 샘플 IMS 애플리케이션에 제공되는 소스 및 JCL의 목록을 표로 제공합니다.

신용 조회 IMS 샘플에 대한 소스 및 JCL(C 전용)

표 191. 신용 조회 IMS 샘플에 대한 소스 및 JCL(C 전용)		
멤버 이름	설명	라이브러리에 제공
CSQ4CVB	IBM MQ 오브젝트 정의	SCSQPROC
CSQ4ICB3	당좌 예금 계좌 프로그램을 위한 소스	SCSQC37S
CSQ4ICBL	당좌 예금 데이터베이스 로드를 위한 소스	SCSQC37S
CSQ4CBI	데이터 정의	SCSQC370
CSQ4PSBL	데이터베이스 로드 프로그램을 위한 PSBGEN JCL	SCSQPROC
CSQ4PSB3	당좌 예금 계좌 프로그램을 위한 PSBGEN JCL	SCSQPROC
CSQ4DBDS	데이터베이스 CSQ4CA를 위한 DBDGEN JCL	SCSQPROC
CSQ4GIMS	CSQ4IVB3 및 CSQ4CA를 위한 IMSGEN 매크로 정의	SCSQPROC
CSQ4ACBG	CSQ4IVB3을 위한 애플리케이션 제어 블록(ACB) 정의	SCSQPROC
CSQ4BAQ	데이터베이스에 대한 소스	SCSQPROC
CSQ4ILDB	데이터베이스 로드 작업을 위한 샘플 실행 JCL	SCSQPROC
CSQ4ICBR	당좌 예금 계좌 프로그램을 위한 샘플 실행 JCL	SCSQPROC
CSQ4DYNA	데이터베이스에 대한 IMSDALOC 매크로 정의	SCSQPROC

z/OS z/OS에서의 Put 샘플

Put 샘플 프로그램은 MQPUT 호출을 사용하여 큐에 메시지를 넣습니다.

소스 프로그램은 배치 및 CICS 환경에서 C 및 COBOL로 제공됩니다(1087 페이지의 『배치 Put 및 Get 샘플』 및 1092 페이지의 『CICS Put 및 Get 샘플』 참조).

Put 샘플의 디자인

프로그램 로직을 실행하는 플로우는 다음과 같습니다.

1. MQCONN 호출을 사용하여 큐 관리자에 연결합니다. 이 호출에 실패하는 경우 완료 및 이유 코드를 출력하고 처리를 중지하십시오.

참고: CICS 환경에서 샘플을 실행 중인 경우 MQCONN 호출을 발행할 필요가 없습니다. 해당되는 경우 DEF_HCONN을 리턴합니다. 뒤따르는 MQI 호출에 대해 MQHC_DEF_HCONN 연결 핸들을 사용할 수 있습니다.

2. MQOPEN 호출을 MQOO_OUTPUT 옵션과 함께 사용하여 큐를 여십시오. 이 호출에 대한 입력에서 프로그램은 1097 페이지의 『1』 단계에서 리턴된 연결 핸들을 사용합니다. 오브젝트 디스크립터 구조(MQOD)의 경우 프로그램에 매개변수로 전달되는 큐 이름 필드를 제외한 모든 필드에 대해 기본값을 사용합니다. MQOPEN 호출에 실패하는 경우 완료 및 이유 코드를 출력하고 처리를 중지하십시오.
3. 필요한 수 만큼의 메시지를 큐에 넣을 때까지 MQPUT 호출을 발행하는 루프를 프로그램 내에 작성하십시오. MQPUT 호출이 실패하면 루프가 초기에 중단되고 추가적인 MQPUT 호출이 시도되지 않으며 완료 및 이유 코드가 리턴됩니다.
4. 1097 페이지의 『2』 단계에서 리턴된 오브젝트 핸들과 함께 MQCLOSE 호출을 사용하여 큐를 닫으십시오. 이 호출에 실패하는 경우 완료 및 이유 코드를 출력하십시오.
5. 1097 페이지의 『1』 단계에서 리턴된 연결 핸들과 함께 MQDISC 호출을 사용하여 큐 관리자에서 연결을 끊으십시오. 이 호출에 실패하는 경우 완료 및 이유 코드를 출력하십시오.

참고: CICS 환경에서 샘플을 실행 중인 경우 MQDISCN 호출을 발행할 필요가 없습니다.

z/OS의 배치 환경에 대한 Put 샘플

배치 환경에 대한 Put 샘플을 고려하는 경우 이 주제를 사용하십시오.

샘플을 실행하려면 1086 페이지의 『배치 환경용 샘플 애플리케이션 준비 z/OS』에서 설명된 대로 샘플 JCL을 편집하고 실행하십시오.

프로그램은 C의 경우 공백, COBOL의 경우 쉼표로 구분된 EXEC PARM에서 다음 매개변수를 사용합니다.

1. 큐 관리자의 이름(4자)
2. 대상 큐의 이름(48개 문자)
3. 메시지 수(최대 4자리)
4. 메시지에 작성하기 위한 패딩 문자(1개 문자)
5. 메시지에 작성하기 위한 문자 수(최대 4자리)
6. 메시지의 지속성(1개 문자: P는 지속적인 경우, N은 비지속적인 경우)

이러한 매개변수를 잘못 입력하는 경우 해당되는 오류 메시지를 수신합니다.

샘플의 메시지가 SYSPRINT 데이터 세트에 작성됩니다.

사용법 참고

- 샘플을 단순히 유지하기 위해 언어 버전 간에 몇 가지 약간의 기능적 차이가 있습니다. 그러나, 이러한 차이는 샘플 실행 JCL, CSQ4BCJR 및 CSQ4BVJR에 표시된 매개변수의 레이아웃을 사용하는 경우 최소화됩니다. MQI에 관련된 차이점은 없습니다.
- CSQ4BCK1을 통해 송신된 메시지 수 및 메시지 길이에 대해 4자리 이상을 입력할 수 있습니다.
- 두 개 숫자 필드의 경우 1에서 9,999 범위의 숫자를 입력하십시오. 입력하는 값은 양수여야 합니다. 예를 들어, 단일 메시지를 넣기 위해 1, 01, 001 또는 0001 값을 입력할 수 있습니다. 숫자가 아닌 값 또는 음수를 입력하면 오류를 수신할 수 있습니다. 예를 들어, -1을 입력하는 경우 COBOL 프로그램은 1바이트 메시지를 송신하지만 C 프로그램은 오류를 수신합니다.
- CSQ4BCK1 및 CSQ4BVK1 두 프로그램의 경우, 메시지를 지속 메시지가 되게 하려면 지속성 매개변수에 P를 입력하십시오. 이를 수행하는 데 실패하면 비지속 메시지가 됩니다.

'CICS 환경에 대한 샘플을 'z/OS'에 넣습니다

CICS 환경에 대한 Put 샘플을 고려하는 경우 이 주제를 사용하십시오.

트랜잭션은 쉽표로 구분된 다음 매개변수를 사용합니다.

1. 메시지 수(최대 4자리)
2. 메시지에 작성하기 위한 패딩 문자(1개 문자)
3. 메시지에 작성하기 위한 문자 수(최대 4자리)
4. 메시지의 지속성(1개 문자: P는 지속적인 경우, N은 비지속적인 경우)
5. 대상 큐의 이름(48개 문자)

이러한 매개변수를 잘못 입력하는 경우 해당하는 오류 메시지를 수신합니다.

COBOL 샘플의 경우 다음을 입력하여 CICS 환경에서 Put 샘플을 호출하십시오.

```
MVPT,9999,*,9999,P,QUEUE.NAME
```

C 샘플의 경우 다음을 입력하여 CICS 환경에서 Put 샘플을 호출하십시오.

```
MCPT,9999,*,9999,P,QUEUE.NAME
```

샘플의 메시지가 화면에 표시됩니다.

사용법 참고

- 샘플을 단순히 유지하기 위해 언어 버전 간에 몇 가지 약간의 기능적 차이가 있습니다. MQI에 관련된 차이점은 없습니다.
- 48자보다 긴 큐 이름을 입력하는 경우 해당 길이는 최대 48자로 잘리지만 오류 메시지는 리턴되지 않습니다.
- 트랜잭션을 입력하기 전에 CLEAR 키를 누르십시오.
- 두 개 숫자 필드의 경우 1에서 9,999 범위의 수를 입력하십시오. 입력하는 값은 양수여야 합니다. 예를 들어, 단일 메시지를 넣기 위해 1, 01, 001 또는 0001 값을 입력할 수 있습니다. 숫자가 아닌 값 또는 음수를 입력하면 오류를 수신할 수 있습니다. 예를 들어, -1을 입력하는 경우 COBOL 프로그램은 1바이트 메시지를 송신하지만 C 프로그램은 malloc()에서 오류와 함께 종료됩니다.
- CSQ4CCK1 및 CSQ4CVK1 두 프로그램의 경우, 메시지를 지속 메시지가 되게 하려면 지속성 매개변수에 P를 입력하십시오. 비지속 메시지의 경우 지속성 매개변수에 N을 입력하십시오. 이외 다른 값을 입력하면 오류 메시지를 수신합니다.
- 프로그램 호출 중에 설정된 매개변수를 제외하고 모든 매개변수에 기본값이 사용되기 때문에 동기점에 메시지를 넣습니다.

z/OS에서의 Get 샘플

Get 샘플 프로그램은 MQGET 호출을 사용하여 큐에서 메시지를 가져옵니다.

소스 프로그램은 배치 및 CICS 환경에서 C 및 COBOL로 제공됩니다([1087 페이지의 『배치 Put 및 Get 샘플』](#) 및 [1092 페이지의 『CICS Put 및 Get 샘플』](#) 참조).

z/OS에서의 Get 샘플 디자인

Get 샘플의 디자인 및 고려할 일부 사용법 참고사항에 대해 알아보십시오.

프로그램 로직을 실행하는 플로우는 다음과 같습니다.

1. MQCONN 호출을 사용하여 큐 관리자에 연결합니다. 이 호출에 실패하는 경우 완료 및 이유 코드를 출력하고 처리를 중지하십시오.

참고: CICS 환경에서 샘플을 실행 중인 경우 MQCONN 호출을 발행할 필요가 없습니다. 해당되는 경우 DEF_HCONN을 리턴합니다. 뒤따르는 MQI 호출에 대해 MQHC_DEF_HCONN 연결 핸들을 사용할 수 있습니다.

2. MQOPEN 호출을 MQOO_INPUT_SHARED 및 MQOO_BROWSE 옵션과 함께 사용하여 큐를 여십시오. 이 호출에 대한 입력에서 프로그램은 [1097 페이지의 『1』](#) 단계에서 리턴된 연결 핸들을 사용합니다. 오브젝트 디

스크립터 구조(MQOD)의 경우 프로그램에 매개변수로 전달되는 큐 이름 필드를 제외한 모든 필드에 대해 기본값을 사용합니다. MQOPEN 호출에 실패하는 경우 완료 및 이유 코드를 출력하고 처리를 중지하십시오.

3. 큐에서 검색되는 필요한 메시지 수까지 MQGET 호출을 발행하는 루프를 프로그램 내에 작성하십시오. MQGET 호출이 실패하면 루프가 초기에 중단되고 추가적인 MQGET 호출이 시도되지 않으며 완료 및 이유 코드가 리턴됩니다. 다음 옵션이 MQGET 호출에 지정됩니다.

- MQGMO_NO_WAIT
- MQGMO_ACCEPT_TRUNCATED_MESSAGE
- MQGMO_SYNCPOINT 또는 MQGMO_NO_SYNCPOINT
- MQGMO_BROWSE_FIRST 및 MQGMO_BROWSE_NEXT

이러한 옵션에 대한 설명은 MQGET을 참조하십시오. 각 메시지의 경우 메시지 번호가 출력되며 뒤에 메시지 길이와 메시지 데이터가 출력됩니다.

4. 1097 페이지의 『2』 단계에서 리턴된 오브젝트 핸들과 함께 MQCLOSE 호출을 사용하여 큐를 닫으십시오. 이 호출에 실패하는 경우 완료 및 이유 코드를 출력하십시오.
5. 1097 페이지의 『1』 단계에서 리턴된 연결 핸들과 함께 MQDISC 호출을 사용하여 큐 관리자에서 연결을 끊으십시오. 이 호출에 실패하는 경우 완료 및 이유 코드를 출력하십시오.

참고: CICS 환경에서 샘플을 실행 중인 경우 MQDISCN 호출을 발행할 필요가 없습니다.

사용법 참고

- 샘플을 단순하게 유지하기 위해 언어 버전 간에 몇 가지 약간의 기능적 차이가 있습니다. 그러나, 이러한 차이는 샘플 실행 JCL, CSQ4BCJR 및 CSQ4BVJR에 표시된 매개변수의 레이아웃을 사용하는 경우 최소화됩니다. MQI에 관련된 차이점은 없습니다.
- CSQ4BCJ1을 통해 검색된 메시지 수에 대해 4자리 이상을 입력할 수 있습니다.
- 64KB보다 긴 메시지는 잘립니다.
- CSQ4BCJ1은 처음 널(\0) 문자가 표시될 때까지만 표시하기 때문에 항상 문자 메시지를 올바르게 표시할 수 있습니다.
- 숫자 메시지 수 필드의 경우 1에서 9,999 범위의 숫자를 입력하십시오. 입력하는 값은 양수여야 합니다. 예를 들어, 단일 메시지를 가져오기 위해 1, 01, 001 또는 0001 값을 입력할 수 있습니다. 숫자가 아닌 값 또는 음수를 입력하면 오류를 수신할 수 있습니다. 예를 들어, -1을 입력하는 경우 COBOL 프로그램은 하나의 메시지를 검색하지만 C 프로그램은 메시지를 검색하지 않습니다.
- CSQ4BCJ1 및 CSQ4BVJ1 두 프로그램의 경우, 메시지를 찾아보려면 가져오기 매개변수 ++GET++에 B를 입력하십시오.
- CSQ4BCJ1 및 CSQ4BVJ1 두 프로그램의 경우, 동기점에서 검색될 메시지에 대해 동기점 매개변수 ++SYNC++에 S를 입력하십시오.

▶ z/OS z/OS의 배치 환경에 대한 Get 샘플

샘플을 실행하려면 1086 페이지의 『배치 환경용 샘플 애플리케이션 준비 z/OS』에서 설명된 대로 샘플 JCL을 편집하고 실행하십시오.

프로그램은 C의 경우 공백, COBOL의 경우 쉼표로 구분된 EXEC PARM에서 다음 매개변수를 사용합니다.

1. 큐 관리자의 이름(4자)
2. 대상 큐의 이름(48개 문자)
3. 가져올 메시지 수(최대 4자리)
4. 찾아보기/가져오기 메시지 옵션(1개 문자: B는 찾아보기, D는 파괴적으로 메시지 가져오기)
5. 동기점 제어(1개 문자: S는 동기점, N은 동기점 없음)

이러한 매개변수를 잘못 입력하는 경우 해당하는 오류 메시지를 수신합니다.

샘플의 출력은 SYSPRINT 데이터 세트에 작성됩니다.

```
=====
PARAMETERS PASSED :
```

```

QMGR      - VC9
QNAME     - A.Q
NUMMSGs   - 0000000002
GET       - D
SYNCPPOINT - N
=====
MQCONN SUCCESSFUL
MQOPEN SUCCESSFUL
000000000 : 000000010 : *****
000000001 : 000000010 : *****
000000002 MESSAGES GOT FROM QUEUE
MQCLOSE SUCCESSFUL
MQDISC SUCCESSFUL

```

▶ z/OS 'z/OS'에서 'CICS' 환경에 대한 샘플 가져오기

CICS 환경에 대한 Get 샘플 관련 특수 고려사항입니다.

트랜잭션은 쉽표로 구분된 EXEC PARM에서 다음 매개변수를 사용합니다.

1. 가져올 메시지 수(최대 네 자리)
2. 찾아보기/가져오기 메시지 옵션(한 문자: B는 찾아보기, D는 파괴적으로 메시지 가져오기)
3. 동기점 제어(한 문자: S는 동기점, N은 동기점 없음)
4. 대상 큐의 이름(48개 문자)

이러한 매개변수를 잘못 입력하는 경우 해당하는 오류 메시지를 수신합니다.

COBOL 샘플의 경우 다음을 입력하여 CICS 환경에서 Get 샘플을 호출하십시오.

```
MVGT,9999,B,S,QUEUE.NAME
```

C 샘플의 경우 다음을 입력하여 CICS 환경에서 Get 샘플을 호출하십시오.

```
MCGT,9999,B,S,QUEUE.NAME
```

큐에서 메시지가 검색되는 경우 CICS 트랜잭션과 동일한 이름을 사용하여 CICS 임시 동적 큐에 메시지를 넣습니다(예: C 샘플의 경우 MCGT).

다음은 Get 샘플의 출력 예입니다.

```

***** TOP OF QUEUE *****
000000000 : 000000010 : *****
000000001 : 000000010 : *****
***** BOTTOM OF QUEUE *****

```

사용법 참고

- 샘플을 단순히 유지하기 위해 언어 버전 간에 몇 가지 약간의 기능적 차이가 있습니다. MQI에 관련된 차이점은 없습니다.
- 48자보다 긴 큐 이름을 입력하는 경우 해당 길이는 최대 48자로 잘리지만 오류 메시지는 리턴되지 않습니다.
- 트랜잭션을 입력하기 전에 CLEAR 키를 누르십시오.
- CSQ4CCJ1은 처음 널(\0) 문자가 표시될 때까지만 표시하기 때문에 항상 문자 메시지를 올바르게 표시할 수 있습니다.
- 숫자 필드의 경우 1에서 9,999 범위의 수를 입력하십시오. 입력하는 값은 양수여야 합니다. 예를 들어, 단일 메시지를 가져오기 위해 1, 01, 001 또는 0001 값을 입력할 수 있습니다. 숫자가 아닌 값 또는 음수를 입력하면 오류를 수신할 수 있습니다.
- C의 경우 24,526바이트, COBOL의 경우 9,950바이트보다 긴 메시지는 잘립니다. 이는 CICS 임시 동적 큐가 사용되는 방법 때문입니다.
- 두 프로그램 CSQ4CCK1 및 CSQ4CVK1 모두에 대해 메시지를 찾아보려면 가져오기 매개변수에 B를 입력하고 그렇지 않으면 D를 입력하십시오. 이는 파괴적 MQGET 호출을 수행합니다. 이외 다른 값을 입력하면 오류 메시지를 수신합니다.

- CSQ4CCJ1 및 CSQ4CVJ1 두 프로그램의 경우, 동기점에서 메시지를 검색하려면 동기점 매개변수에 S를 입력하십시오. N을 동기점 매개변수에 입력하면 MQGET 호출이 동기점 외부에서 발행됩니다. 이외 다른 값을 입력하면 오류 메시지를 수신합니다.

z/OS z/OS에서의 찾아보기 샘플

Browse 샘플은 MQGET 호출을 사용하여 큐에서 메시지를 찾아보는 방법을 보여주는 일괄처리 애플리케이션입니다.

애플리케이션은 큐의 모든 메시지를 스텝 스루하고 각 메시지의 처음 80바이트를 출력합니다. 이 애플리케이션을 사용하여 메시지를 변경하지 않고 큐에서 메시지를 볼 수 있습니다.

소스 프로그램 및 샘플 실행 JCL은 COBOL, 어셈블러, PL/I 및 C 언어로 제공됩니다(1087 페이지의 『배치 찾아보기 샘플』 참조).

애플리케이션을 시작하려면 1086 페이지의 『배치 환경용 샘플 애플리케이션 준비 z/OS』에서 설명된 대로 샘플 실행 JCL을 편집하고 실행하십시오. 실행 JCL에 큐의 이름을 지정하여 고유 큐 중 하나에서 메시지를 볼 수 있습니다.

애플리케이션을 실행하는 경우(그리고 큐에 일부 메시지가 있는 경우) 출력 데이터 세트는 다음과 같습니다.

```
07/12/1998          SAMPLE QUEUE REPORT          PAGE 1
QUEUE MANAGER NAME : VC4
QUEUE NAME : CSQ4SAMP.DEAD.QUEUE
RELATIVE
MESSAGE MESSAGE
NUMBER LENGTH ----- MESSAGE DATA -----
1      740 HELLO. PLEASE CALL ME WHEN YOU GET BACK.
2      429 CSQ4BQRM
3      429 CSQ4BQRM
4      429 CSQ4BQRM
5      22 THIS IS A TEST MESSAGE
6       8 CSQ4TEST
7      36 CSQ4MSG - ANOTHER TEST MESSAGE.....
!8      9 CSQ4STOP
***** END OF REPORT *****
```

큐에 메시지가 없는 경우 데이터 세트에는 표제 및 보고서 끝 메시지만 포함됩니다. MQI 호출에서 오류가 발생하는 경우 완료 및 이유 코드가 출력 데이터 세트에 추가됩니다.

z/OS z/OS의 찾아보기 샘플 디자인

찾아보기 샘플 애플리케이션은 단일 프로그램 모듈을 사용합니다. 지원되는 각 프로그래밍 언어마다 하나씩 제공됩니다.

프로그램 로직을 실행하는 플로우는 다음과 같습니다.

1. 출력 데이터 세트를 열고 보고서의 제목 행을 출력하십시오. 큐 관리자 및 큐의 이름이 실행 JCL에서 전달되었는지 확인합니다. 두 이름이 모두 전달된 경우 이름이 포함되어 있는 보고서의 행을 출력하십시오. 해당되지 않는 경우 오류 메시지를 출력하고, 출력 데이터 세트를 닫고 처리를 중지하십시오.
프로그램이 JCL에서 전달되는 매개변수를 테스트하는 방법은 프로그램이 작성된 언어에 따라 달라집니다. 자세한 정보는 1101 페이지의 『z/OS에서의 언어 종속적 디자인 고려사항』의 내용을 참조하십시오.
2. MQCONN 호출을 사용하여 큐 관리자에 연결합니다. 이 호출에 실패하는 경우 완료 및 이유 코드를 출력하고, 출력 데이터 세트를 닫고, 처리를 중지하십시오.
3. MQOPEN 호출을 MQOO_BROWSE 옵션과 함께 사용하여 큐를 여십시오. 이 호출에 대한 입력에서 프로그램은 1100 페이지의 『2』 단계에서 리턴된 연결 핸들을 사용합니다. 오브젝트 디스크립터 구조(MQOD)의 경우 큐 이름을 제외한 모든 필드에 기본값을 사용합니다(1100 페이지의 『1』 단계에서 전달됨). 이 호출에 실패하는 경우 완료 및 이유 코드를 출력하고, 출력 데이터 세트를 닫고, 처리를 중지하십시오.
4. MQGET 호출을 사용하여 큐에서 첫 번째 메시지를 찾아보십시오. 이 호출에 대한 입력에서 프로그램은 다음을 지정합니다.
 - 1100 페이지의 『2』 단계 및 1100 페이지의 『3』 단계의 연결 및 큐 핸들
 - 모든 필드가 초기값으로 설정된 MQMD 구조

- 두 가지 옵션:

- MQGMO_BROWSE_FIRST
- MQGMO_ACCEPT_TRUNCATED_MSG

- 메시지에서 복사된 데이터를 보관하기 위한 80바이트 크기의 버퍼

MQGMO_ACCEPT_TRUNCATED_MSG 옵션을 통해 메시지가 호출에서 지정된 80바이트 버퍼보다 길더라도 호출을 완료할 수 있습니다. 메시지가 버퍼보다 긴 경우 메시지는 버퍼에 맞게 잘려지고 완료 및 이유 코드가 이를 표시하도록 설정됩니다. 보고서를 읽기 쉽게 만들기 위해 메시지가 80개의 문자로 잘릴 수 있도록 샘플이 설계됩니다. 버퍼 크기는 DEFINE 명령문에 의해 설정되므로 원하는 경우 쉽게 변경할 수 있습니다.

5. MQGET 호출이 실패할 때까지 다음 루프를 수행하십시오.

- a. 다음을 표시하는 보고서 행을 출력하십시오.

- 메시지의 순서 번호(찾아보기 조작의 수).
- 메시지의 실제 길이(잘린 길이가 아닌). 이 값은 MQGET 호출의 DataLength 필드에 리턴됩니다.
- 메시지 데이터의 처음 80바이트.

- b. MQMD 구조의 MsgId 및 CorrelId 필드를 널로 재설정하십시오.

- c. 다음 두 가지 옵션과 함께 MQGET 호출을 사용하여 다음 메시지를 찾아보십시오.

- MQGMO_BROWSE_NEXT
- MQGMO_ACCEPT_TRUNCATED_MSG

6. MQGET 호출에 실패하는 경우 찾아보기 커서가 큐의 끝에 있어야 하기 때문에 호출이 실패했는지 이유 코드를 테스트하십시오. 이 경우 보고서 끝 메시지를 출력하고 [1101 페이지의 『7』](#) 단계로 이동하십시오. 그렇지 않으면 완료 및 이유 코드를 인쇄하고 출력 데이터 세트를 닫고 처리를 중지하십시오.

7. [1100 페이지의 『3』](#) 단계에서 리턴된 오브젝트 핸들과 함께 MQCLOSE 호출을 사용하여 큐를 닫으십시오.

8. [1100 페이지의 『2』](#) 단계에서 리턴된 연결 핸들과 함께 MQDISC 호출을 사용하여 큐 관리자에서 연결을 끊으십시오.

9. 출력 데이터 세트를 닫고 처리를 중지하십시오.

▶ **z/OS** z/OS에서의 언어 종속적 디자인 고려사항

4개의 프로그래밍 언어로 된 찾아보기 샘플에 대한 소스 모듈이 제공됩니다.

소스 모듈 간에 두 가지의 주요한 차이가 있습니다.

- 실행 JCL에서 전달된 매개변수를 테스트할 때 COBOL, PL/I 및 어셈블러 언어 모듈은 쉼표 문자(,)를 검색합니다. JCL이 PARM=(, LOCALQ1)을(를) 전달하면 애플리케이션은 기본 큐 관리자에서 LOCALQ1 큐를 열려고 합니다. 쉼표(또는 쉼표 없음) 뒤에 이름이 없는 경우 애플리케이션은 오류를 리턴합니다. C 모듈은 쉼표 문자를 검색하지 않습니다. JCL이 단일 매개변수(예: PARM=('LOCALQ1'))를 전달하면 C 모듈은 이 매개변수를 기본 큐 관리자에서 큐 이름으로 사용합니다.
- 어셈블러 언어 모듈을 단순히 유지하기 위해 인쇄 보고서를 작성하는 경우 yy/ddd 날짜 형식(예: 05/116)을 사용합니다. 다른 모듈에서는 mm/dd/yy 형식의 달력 날짜를 사용합니다.

▶ **z/OS** z/OS에서의 메시지 인쇄 샘플

메시지 인쇄 샘플은 MQGET 호출을 사용하여 큐에서 모든 메시지를 제거하는 방법을 보여주는 배치 애플리케이션입니다.

메시지 출력 샘플은 다음 세 가지 매개변수를 사용합니다.

1. 큐 관리자의 이름
2. 소스 큐의 이름
3. 특성에 대한 선택적 매개변수

또한, 각 메시지에 대해 메시지 디스크립터의 필드와 그 뒤에 메시지 데이터를 출력합니다. 프로그램은 16진 문자로 데이터를 출력합니다(출력 가능한 경우). 문자를 인쇄할 수 없는 경우 프로그램은 이 문자를 마침표(.)로 대체합니다. 큐에 메시지를 넣는 애플리케이션의 문제점을 진단할 때 프로그램을 사용할 수 있습니다.

특성 매개변수에 대한 허용 가능한 값은 다음과 같습니다.

표 192. 특성 매개변수에 대해 허용 가능한 값	
값	동작
0	기본 동작입니다. 애플리케이션에 전달시키는 특성은 메시지가 검색되는 PropertyControl 큐 속성에 따라 달라집니다.
1	메시지 핸들이 작성되고 MQGET과 함께 사용됩니다. 메시지 디스크립터(또는 확장)에 포함된 특성을 제외하고, 메시지의 특성은 메시지 디스크립터와 유사한 방식으로 표시됩니다. 예를 들면, 다음과 같습니다. <pre>****Message properties**** property name: property value</pre> 또는 특성을 사용할 수 없는 경우, <pre>****Message properties**** None</pre> 숫자 값은 printf를 사용하여 표시되고, 문자열 값은 작은따옴표로 묶으며, 바이트 문자열은 메시지 디스크립터의 경우 X와 작은따옴표로 묶습니다.
2	메시지 디스크립터 특성만 리턴되도록 MQGMO_NO_PROPERTIES가 지정됩니다.
3	모든 특성이 메시지 데이터에 리턴되도록 MQGMO_PROPERTIES_FORCE_MQRFH2가 지정됩니다.
4	IBM MQ 특성이 포함되는지에 따라 모든 특성을 리턴할 수 있으며 그렇지 않은 경우 특성이 제거되도록 MQGMO_PROPERTIES_COMPATIBILITY가 지정됩니다.

큐에서 메시지를 제거하기 보다 메시지를 찾아보도록 애플리케이션을 변경할 수 있습니다. 이를 수행하려면, 1103 페이지의 『z/OS에서의 메시지 인쇄 샘플 디자인』에 표시된 대로 BROWSE 매크로를 정의하기 위해 -DBROWSE의 옵션으로 컴파일하십시오. 실행 코드는 SCSQLOAD 라이브러리에 제공됩니다. CSQ4BCG0 모듈은 -DBROWSE를 사용하여 빌드됩니다. CSQ4BCG1 모듈은 파괴적으로 큐를 읽습니다.

애플리케이션에는 C 언어로 작성된 단일 소스 프로그램이 있습니다. 또한 샘플 실행 JCL 코드도 제공됩니다 (1088 페이지의 『배치 출력 메시지 샘플(C 언어 전용)』 참조).

애플리케이션을 시작하려면 1086 페이지의 『배치 환경용 샘플 애플리케이션 준비 z/OS』에서 설명된 대로 샘플 실행 JCL을 편집하고 실행하십시오. 애플리케이션을 실행하는 경우(그리고 큐에 일부 메시지가 있는 경우) 출력 데이터 세트는 1103 페이지의 그림 156처럼 보입니다.


```

CSQ4BCG1 - starts here
*****

MQCONN to MQ1E
MQOPEN - 'TEST.QUEUE'
MQCRTMH

MQGET of message number 1
****Message descriptor****
StrucId : 'MD ' Version : 2
Report : 0 MsgType : 8
Expiry : -1 Feedback : 0
Encoding : 785 CodedCharSetId : 500
Format : 'MQSTR '
Priority : 0 Persistence : 0
MsgId : X'C3E2D840D4D8F1C5404040404040C1EA537F03167D88'
CorrelId : X'C3E2D840D4D8F1C5404040404040C1EA537F0317A928'
BackoutCount : 0
ReplyToQ : ' '
ReplyToQMgr : ' '
** Identity Context
UserIdentifier : 'FRED '
AccountingToken :
X'0000000000000000000000000000000000000000000000000000000000000000'
ApplIdentityData : ' '
** Origin Context
PutApplType : '2'
PutApplName : 'FRED6 '
PutDate : '20080207' PutTime : '17373745'
ApplOriginData : ' '
GroupId : X'0000000000000000000000000000000000000000000000000000000000000000'
MsgSeqNumber : '1'
Offset : '0'
MsgFlags : '0'
OriginalLength : '-1'

****Message properties****
None

**** Message ****

length - 30 bytes

00000000: E388 89A2 4089 A240 8140 A289 9497 9385 'This is a simple'
00000010: 40A3 85A2 A340 9485 A2A2 8187 855A ' test message! '

No more messages
MQDLTMH
MQCLOSE
MQDISC

```

그림 156. 인쇄 메시지 샘플 애플리케이션의 보고서 예

z/OS에서의 메시지 인쇄 샘플 디자인

메시지 인쇄 샘플 애플리케이션은 C 언어로 작성된 단일 프로그램을 사용합니다.

프로그램 로직을 실행하는 플로우는 다음과 같습니다.

1. 큐 관리자 및 큐의 이름이 실행 JCL에서 전달되었는지 확인합니다. 해당되지 않는 경우 오류 메시지를 출력하고 처리를 중지합니다.
2. MQCONN 호출을 사용하여 큐 관리자에 연결합니다. 이 호출에 실패하는 경우 완료 및 이유 코드를 출력하고 처리를 중지합니다. 그렇지 않으면 큐 관리자의 이름을 출력합니다.
3. MQOPEN 호출을 MQOO_INPUT_SHARED 옵션과 함께 사용하여 큐를 엽니다.

참고: 애플리케이션이 큐에서 메시지를 제거하는 대신 메시지를 찾아보게 하려는 경우 -DBROWSE를 사용하여 샘플을 컴파일하거나 소스의 시작에 `#define BROWSE`를 추가하십시오. 이를 수행하면 매크로 프리 프로세서는 컴파일 작업에서 MQOO_BROWSE 옵션을 선택하는 프로그램에 행을 추가합니다.

이 호출에 대한 입력에서 프로그램은 [1103 페이지의 『2』](#) 단계에서 리턴된 연결 핸들을 사용합니다. 오브젝트 디스크립터 구조(MQOD)의 경우 큐 이름을 제외한 모든 필드에 기본값을 사용합니다([1103 페이지의](#)

『1』 단계에서 전달됨). 이 호출에 실패하는 경우 완료 및 이유 코드를 출력하고 처리를 중지하십시오. 그렇지 않으면 큐의 이름을 출력하십시오.

4. 메시지 특성을 확보하기 위해 메시지 핸들을 사용하는 경우 MQCRTMH를 사용하여 후속 MQGET 호출에 사용할 핸들을 작성하십시오. 이 호출에 실패하는 경우 완료 및 이유 코드를 출력하고 처리를 중지하십시오.
5. Get 메시지 옵션을 설정하여 메시지 특성에 대한 요청 조치를 반영하십시오.
6. MQGET 호출이 실패할 때까지 다음 루프를 수행하십시오.

- a. 메시지 데이터가 이미 버퍼에 있는 데이터에 의해 손상되지 않도록 버퍼를 공백으로 초기화하십시오.
- b. MQGET 호출에서 큐의 첫 번째 메시지를 선택하도록 MQMD 구조의 MsgId 및 CorrelId 필드를 널로 설정하십시오.
- c. MQGET 호출을 사용하여 큐에서 메시지를 가져오십시오. 이 호출에 대한 입력에서 프로그램은 다음을 지정합니다.
 - 1103 페이지의 『2』 단계 및 1103 페이지의 『3』 단계의 연결 및 오브젝트 핸들.
 - 모든 필드가 초기값으로 설정된 MQMD 구조입니다. (MsgId 및 CorrelId 는 각 MQGET 호출에 대해 널로 재설정됩니다.)
 - MQGMO_NO_WAIT 옵션.

참고: 애플리케이션이 큐에서 메시지를 제거하는 대신 메시지를 찾아보게 하려는 경우 -DBROWSE를 사용하여 샘플을 컴파일하거나 소스의 시작에 #define BROWSE를 추가하십시오. 이를 수행하면 매크로 프리프로세서는 컴파일 작업에서 MQOO_BROWSE_NEXT 옵션을 선택하는 프로그램에 행을 추가합니다. 현재 오브젝트 핸들과 함께 이전에 사용된 브라우저 커서가 없는 큐에 대한 호출에 이 옵션이 사용되는 경우 논리적으로 첫 번째 메시지 전에 브라우저 커서가 배치됩니다.

- 메시지에서 복사된 데이터를 보관하기 위한 64KB 크기의 버퍼.
- d. printMD 서브루틴을 호출하십시오. 메시지 디스크립터에 있는 각 필드의 이름과 그 뒤에 콘텐츠를 출력합니다.
- e. 1104 페이지의 『4』 단계에서 메시지 핸들을 작성한 경우 printProperties 서브루틴을 호출하여 메시지 특성을 표시하십시오.
- f. 메시지 길이와 그 뒤에 메시지 데이터를 출력하십시오. 메시지 데이터의 각 행은 다음 형식입니다.
 - 데이터의 이 부분에 대한 상대적 위치(16진)
 - 16진 데이터의 16바이트
 - 출력 가능한 경우(출력할 수 없는 문자는 마침표로 대체됨) 문자 형식의 동일한 16바이트 데이터
7. MQGET 호출에 실패하는 경우 큐에 메시지가 없기 때문에 호출이 실패했는지 이유 코드를 테스트하십시오. 이 경우 더 이상 메시지가 없음이라는 메시지를 출력하십시오. 그렇지 않으면 완료 및 이유 코드를 출력하십시오. 두 경우 모두 1104 페이지의 『9』 단계로 이동하십시오.

참고: 64KB 이상의 데이터가 있는 메시지를 찾는 경우 MQGET 호출에 실패합니다. 프로그램을 더 큰 메시지를 처리하도록 변경하기 위해, 다음 중 하나를 수행할 수 있습니다.

- 호출에서 데이터의 처음 64KB를 가져오고 나머지는 제거하도록 MQGMO_ACCEPT_TRUNCATED_MSG 옵션을 MQGET 호출에 추가하십시오.
 - 프로그램에서 이 크기의 데이터를 찾으면 큐에 메시지를 남겨 두도록 하십시오.
 - 버퍼의 크기를 늘리십시오.
8. 1104 페이지의 『4』 단계에서 메시지 핸들을 작성한 경우 MQDLTMH를 호출하여 삭제하십시오.
 9. 1103 페이지의 『3』 단계에서 리턴된 오브젝트 핸들과 함께 MQCLOSE 호출을 사용하여 큐를 닫으십시오.
 10. 1103 페이지의 『2』 단계에서 리턴된 연결 핸들과 함께 MQDISC 호출을 사용하여 큐 관리자에서 연결을 끊으십시오.

z/OS에서의 큐 속성 샘플

큐 속성 샘플은 MQINQ 및 MQSET 호출의 사용을 보여주는 대화식 모드 CICS 애플리케이션입니다.

큐의 **InhibitPut** 및 **InhibitGet** 속성 값을 조회하는 방법, 프로그램에서 큐에 메시지를 넣을 수 없거나 큐에서 메시지를 가져올 수 없도록 값을 변경하는 방법을 표시합니다. 프로그램을 테스트 중인 경우 이러한 방식으로 큐를 잠금하려고 할 수 있습니다.

자체 큐에 대한 우발적인 개입을 방지하기 위해 이 샘플은 이름의 처음 8바이트에 CSQ4SAMP 문자가 있는 큐 오브젝트에서만 작업합니다. 그러나 소스 코드에 이 제한사항을 제거하는 방법을 표시하는 주석이 포함되어 있습니다.

소스 프로그램은 COBOL, 어셈블러 및 C 언어로 제공됩니다([1092 페이지의 『CICS 큐 속성 샘플』](#)의 내용 참조).

샘플의 어셈블러 언어 버전은 재입 가능한 코드를 사용합니다. 이를 수행하기 위해 샘플의 해당 버전에서 각 MQI 호출에 대한 코드에는 MF 키워드가 포함되어 있습니다. 예를 들며, 다음과 같습니다.

```
CALL MQCONN,(NAME,HCONN,COMPCode,REASON),MF=(E,PARMAREA),VL
```

(VL 키워드는 프로그램을 디버깅하는 데 CICS Execution Diagnostic Facility(CEDF)에서 제공하는 트랜잭션을 사용할 수 있다는 것을 의미합니다.) 재진입 가능한 프로그램 작성에 대한 자세한 내용은 [1177 페이지의 『어셈블러로 ‘MQI’ 앱 개발하기』](#).

애플리케이션을 시작하려면, CICS 시스템을 시작하고 다음 CICS 트랜잭션을 사용하십시오.

- COBOL의 경우, MVC1
- 어셈블러 언어의 경우, MAC1
- C의 경우, MCC1

[3단계](#)에서 언급된 CSD 데이터 세트를 변경하여 이러한 트랜잭션의 이름을 변경할 수 있습니다.

샘플의 디자인

샘플을 시작할 때, 다음에 대한 필드가 포함되어 있는 화면 맵을 표시합니다.

- 큐의 이름
- 사용자 요청(유효한 조치: 조회, 허용 또는 금지)
- 큐에 대한 Put 조작의 현재 상태
- 큐에 대한 Get 조작의 현재 상태

처음 두 개 필드는 사용자 입력용입니다. 마지막 두 필드는 애플리케이션에 의해 채워집니다. 여기에서는 INHIBITED 또는 ALLOWED 단어를 표시합니다.

애플리케이션은 처음 두 필드에 입력하는 값의 유효성을 검증합니다. 큐 이름이 CSQ4SAMP 문자로 시작하고 조치 필드에 세 가지 유효한 요청 중 하나를 입력했는지 확인합니다. 애플리케이션은 모든 입력을 대문자로 변환하므로 소문자를 포함하는 이름의 큐를 사용할 수 없습니다.

조치 필드에 inquire를 입력하는 경우, 프로그램 논리 플로우는 다음과 같습니다.

1. MQOPEN 호출에 MQOO_INQUIRE 옵션을 사용하여 큐를 엽니다.
2. MQIA_INHIBIT_GET 및 MQIA_INHIBIT_PUT 선택자를 사용하여 MQINQ를 호출합니다.
3. MQCLOSE 호출을 사용하여 큐를 닫습니다.
4. MQINQ 호출의 **IntAttr**s 매개변수에 리턴되는 속성을 분석하고 INHIBITED 또는 ALLOWED 중 적절한 단어를 관련 화면 필드로 이동합니다.

조치 필드에 inhibit를 입력하는 경우, 프로그램 논리 플로우는 다음과 같습니다.

1. MQOPEN 호출에 MQOO_SET 옵션을 사용하여 큐를 엽니다.
2. **IntAttr**s 매개변수에 MQQA_GET_INHIBITED 및 MQQA_PUT_INHIBITED 값과 함께 MQIA_INHIBIT_GET 및 MQIA_INHIBIT_PUT 선택자를 사용하는 MQSET을 호출하십시오.
3. MQCLOSE 호출을 사용하여 큐를 닫습니다.
4. INHIBITED 단어를 관련 화면필드로 이동하십시오.

조치 필드에 allow를 입력하는 경우, 애플리케이션은 금지 요청에 대한 조치와 유사한 처리를 수행합니다. 유일한 차이점은 속성의 설정과 화면에 표시되는 단어입니다.

애플리케이션은 큐를 열 때 큐 관리자에 대한 기본 연결 핸들을 사용합니다(CICS은(는) CICS 시스템을 시작할 때 큐 관리자에 대한 연결을 설정함). 이 단계에서 애플리케이션은 다음 오류를 트래핑할 수 있습니다.

- 애플리케이션이 큐 관리자에 연결되어 있지 않습니다.
- 큐가 존재하지 않습니다.
- 사용자가 큐에 액세스할 수 있는 권한이 없습니다.
- 애플리케이션에 큐를 열 수 있는 권한이 없습니다.

다른 MQI 오류의 경우 애플리케이션은 완료 코드 및 이유 코드를 표시합니다.

z/OS에서의 메일 관리자 샘플

메일 관리자 샘플 애플리케이션은 단일 환경 및 다른 환경에서 모두 메시지 송신 및 수신을 보여주는 프로그램의 스위트입니다. 이 애플리케이션은 사용자가 다른 큐 관리자를 사용하는 경우에도 메시지를 교환할 수 있는 간단한 전자 메일링 시스템입니다.

애플리케이션은 시스템 명령 입력 큐에 IBM MQ for z/OS 명령을 입력하여 MQOPEN 호출로 큐를 작성하는 방법을 보여줍니다.

애플리케이션의 세 가지 버전이 제공됩니다.

- COBOL로 작성된 CICS 애플리케이션
- COBOL로 작성된 TSO 애플리케이션
- C로 작성된 TSO 애플리케이션

z/OS에서의 메일 관리자 샘플 준비

메일 관리자는 두 가지 환경에서 실행되는 버전으로 제공됩니다. 애플리케이션을 실행하기 전에 수행해야 하는 준비는 사용하려는 환경에 따라 달라집니다.

사인온 사용자 ID가 각 시스템에서 동일한 동안은 TSO 및 CICS 둘 다에 대한 메일 큐와 별명 큐에 액세스할 수 있습니다.

다른 큐 관리자에 메시지를 송신할 수 있으려면 해당 큐 관리자에 대한 메시지 채널을 설정해야 합니다. 이를 수행하려면, IBM MQ의 채널 제어 기능을 사용하고 이 기능은 [채널 제어 기능](#)에서 설명됩니다.

TSO 환경을 위한 샘플 준비

다음 단계를 수행하십시오.

1. 1089 페이지의 『[z/OS에서 TSO 환경에 대한 샘플 애플리케이션 준비](#)』에서 설명된 대로 샘플을 준비하십시오.
2. 샘플에 제공된 CLIST를 조정하여 다음을 정의하십시오.
 - 패널의 위치
 - 메시지 파일의 위치
 - 로드 모듈의 위치
 - 애플리케이션과 함께 사용할 큐 관리자의 이름샘플의 각 언어 버전에 대해 별도의 CLIST가 제공됩니다.
 - COBOL 버전의 경우: CSQ4RVD1
 - C 버전의 경우: CSQ4RCD1
3. 애플리케이션에서 사용한 큐가 큐 관리자에서 사용 가능한지 확인하십시오. (큐는 CSQ4CVD에 정의됩니다.)

참고: VS COBOL II는 ISPF에서 멀티태스킹을 지원하지 않습니다. 이는 분할 화면의 양쪽 모두에서 메일 관리자 샘플 애플리케이션을 사용할 수 없다는 것을 의미합니다. 사용하는 경우 결과는 예측할 수 없습니다.

z/OS에서 메일 관리자 샘플 실행

CICS Transaction Server for z/OS 환경에서 샘플을 시작하려면, 트랜잭션 MAIL을 실행하십시오. 아직 CICS에 사인온하지 않은 경우 애플리케이션은 메일을 송신할 수 있는 사용자 ID를 입력하도록 프롬프트를 표시합니다.

애플리케이션을 시작할 때 메일 큐가 열립니다. 이 큐가 존재하지 않는 경우 애플리케이션에서 사용자를 위해 큐를 작성합니다. 메일 큐는 CSQ4SAMP.MAILMGR. *userid* 양식의 이름을 갖습니다. 여기서 *userid*는 환경에 따라 달라집니다.

TSO에서

사용자의 TSO ID

CICS에

사용자의 CICS 사인온 또는 메일 관리자를 시작할 때 프롬프트되어 사용자가 입력한 사용자 ID

메일 관리자가 사용하는 큐 이름의 모든 부분은 대문자여야 합니다.

그런 다음 애플리케이션은 다음 옵션이 포함된 메뉴 패널을 표시합니다.

- 수신 메일 읽기
- 메일 발송
- 별명 작성

메뉴 패널은 메일 큐에서 대기 중인 메시지의 수도 표시합니다. 각 메뉴 옵션은 추가 패널을 표시합니다.

수신 메일 읽기

메일 관리자는 메일 큐에 있는 메시지의 목록을 표시합니다. (큐에 있는 처음 99개 메시지만 표시됩니다.) 이 패널의 예는 [1110 페이지의 그림 159](#)의 내용을 참조하십시오. 이 목록에서 메시지를 선택하면 메시지의 컨텐츠가 표시됩니다([1111 페이지의 그림 160](#) 참조).

메일 발송

다음은 입력하도록 패널이 프롬프트됩니다.

- 메시지를 송신하려는 사용자의 이름
- 메일 큐를 소유하는 큐 관리자의 이름
- 메시지의 텍스트

사용자 이름 필드에 사용자 ID 또는 메일 관리자를 사용하여 작성한 별명을 입력할 수 있습니다. 사용 중인 동일한 큐 관리자가 사용자의 메일 큐를 소유하는 경우에는 큐 관리자 이름 필드를 공백으로 남겨 둘 수 있으며 사용자 이름 필드에 별명을 입력한 경우에는 반드시 공백으로 두어야 합니다.

- 사용자 이름만 지정한 경우 프로그램은 우선 이름이 별명이며, 해당 이름에서 정의한 오브젝트에 메시지를 송신한다고 가정합니다. 별명이 없는 경우 프로그램은 해당 이름의 로컬 큐로 메시지를 송신하려고 시도합니다.
- 사용자 이름 및 큐 관리자 이름을 둘 다 지정한 경우 프로그램은 이러한 두 가지 이름에서 정의한 메일 큐로 메시지를 송신합니다.

예를 들어, 리모트 큐 관리자 QM12의 사용자 JONESM에 메시지를 송신하려는 경우 다음 두 가지 방법 중 하나를 사용하여 메시지를 보낼 수 있습니다.

- 두 필드를 모두 사용하여 큐 관리자 QM12의 사용자 JONESM을 지정합니다.
- 해당 사용자에 대한 별명(예: MARY)을 정의하고 사용자 이름 필드에 MARY를 넣고 큐 관리자 이름 필드는 공백으로 두어 메시지를 보냅니다.

별명 작성

자주 연락하는 다른 사용자에게 메시지를 송신할 때 사용할 수 있는 기억하기 쉬운 이름을 정의할 수 있습니다. 다른 사용자의 사용자 ID 및 메일 큐를 소유하는 큐 관리자의 이름을 입력하도록 프롬프트가 표시됩니다.

별명은 CSQ4SAMP.MAILMGR. *userid.nickname* 양식의 이름을 갖는 큐입니다. 여기서 *userid*는 고유 사용자 ID이고 *nickname*은 사용하려는 닉네임입니다. 이런 방식으로 구조화된 이름을 사용하여 사용자는 각각 고유한 별명 세트를 가질 수 있습니다.

프로그램이 작성하는 큐의 유형은 별명 작성 패널의 필드를 완료하는 방법에 따라 달라집니다.

- 사용자 이름만 지정했거나 큐 관리자 이름이 메일 관리자가 연결되는 큐 관리자 이름과 동일한 경우 프로그램은 알리어스 큐를 작성합니다.
- 사용자 이름과 큐 관리자 이름을 모두 지정한 경우(및 큐 관리자가 메일 관리자가 연결되는 큐 관리자가 아닌 경우) 프로그램은 리모트 큐의 로컬 정의를 작성합니다. 프로그램은 이 정의를 해석하는 큐가 있는지, 심지어 리모트 큐 관리자가 있는지 확인하지 않습니다.

예를 들어, 고유 사용자 ID가 SMITHK이고 사용자 JONESM(리모트 큐 관리자 QM12를 사용하는)에 대해 MARY라는 별명을 작성하는 경우 별명 프로그램은 CSQ4SAMP.MAILMGR.SMITHK.MARY라는 리모트 큐의 로컬 정의를 작성합니다. 이 정의는 큐 관리자 QM12에서 CSQ4SAMP.MAILMGR.JONESM인 Mary의 메일 큐로 해석됩니다. 사용자가 QM12 큐 관리자를 직접 사용 중인 경우 프로그램은 대신 동일한 이름의 알리어스 큐를 작성합니다(CSQ4SAMP.MAILMGR.SMITHK.MARY).

TSO 애플리케이션의 C 버전은 COBOL 버전 보다 ISPF의 메시지 핸들링 기능을 더 잘 이용합니다. C와 COBOL 버전에서 다른 오류 메시지가 표시된다는 것을 알 수 있습니다.

z/OS z/OS에서의 메일 관리자 샘플 디자인

다음 절은 메일 관리자 샘플 애플리케이션을 구성하는 각 프로그램에 대해 설명합니다.

애플리케이션이 사용하는 프로그램과 패널 간의 관계는 TSO 버전의 경우 1108 페이지의 [그림 157](#)에 표시되고 z/OS 용 CICS Transaction Server 버전의 경우 1109 페이지의 [그림 158](#)에 표시됩니다.

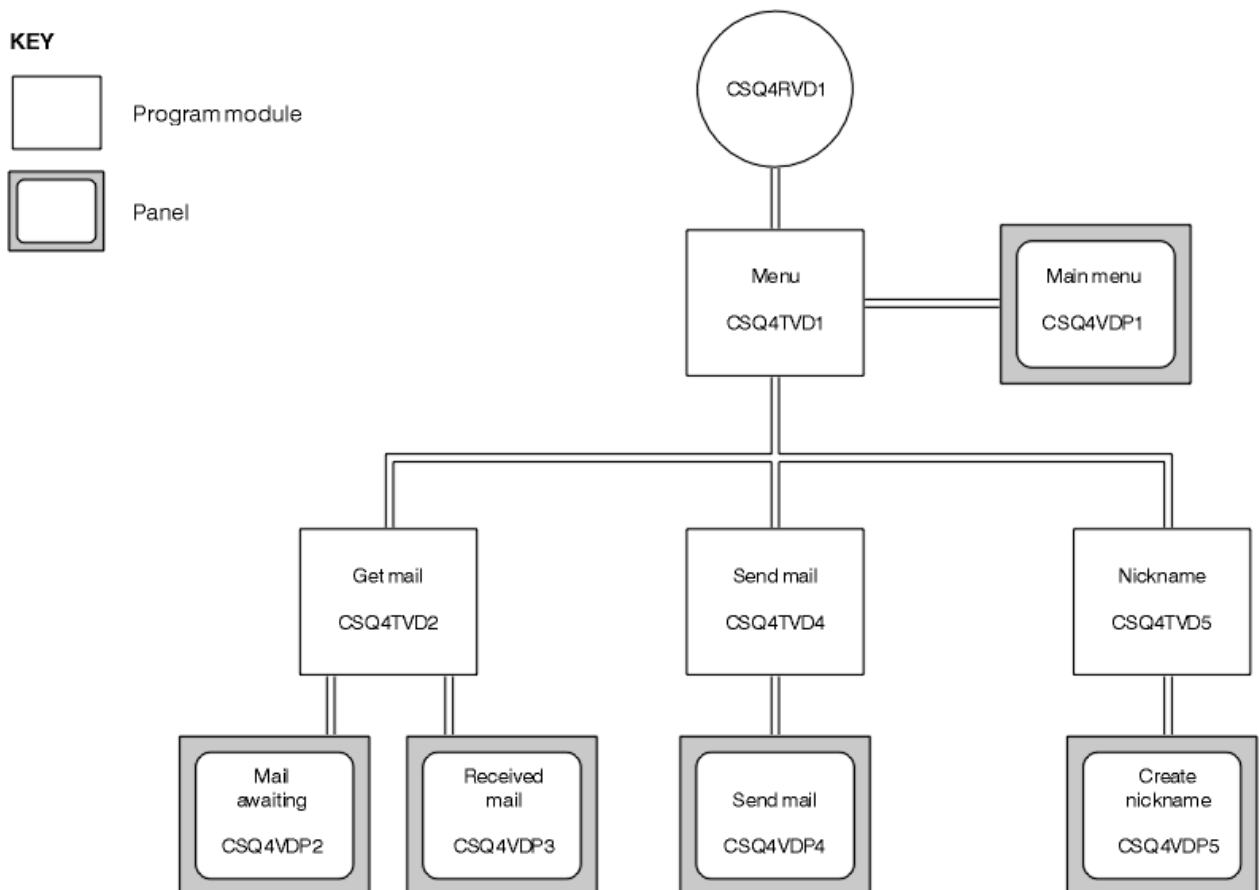


그림 157. 메일 관리자의 TSO 버전에 대한 프로그램 및 패널

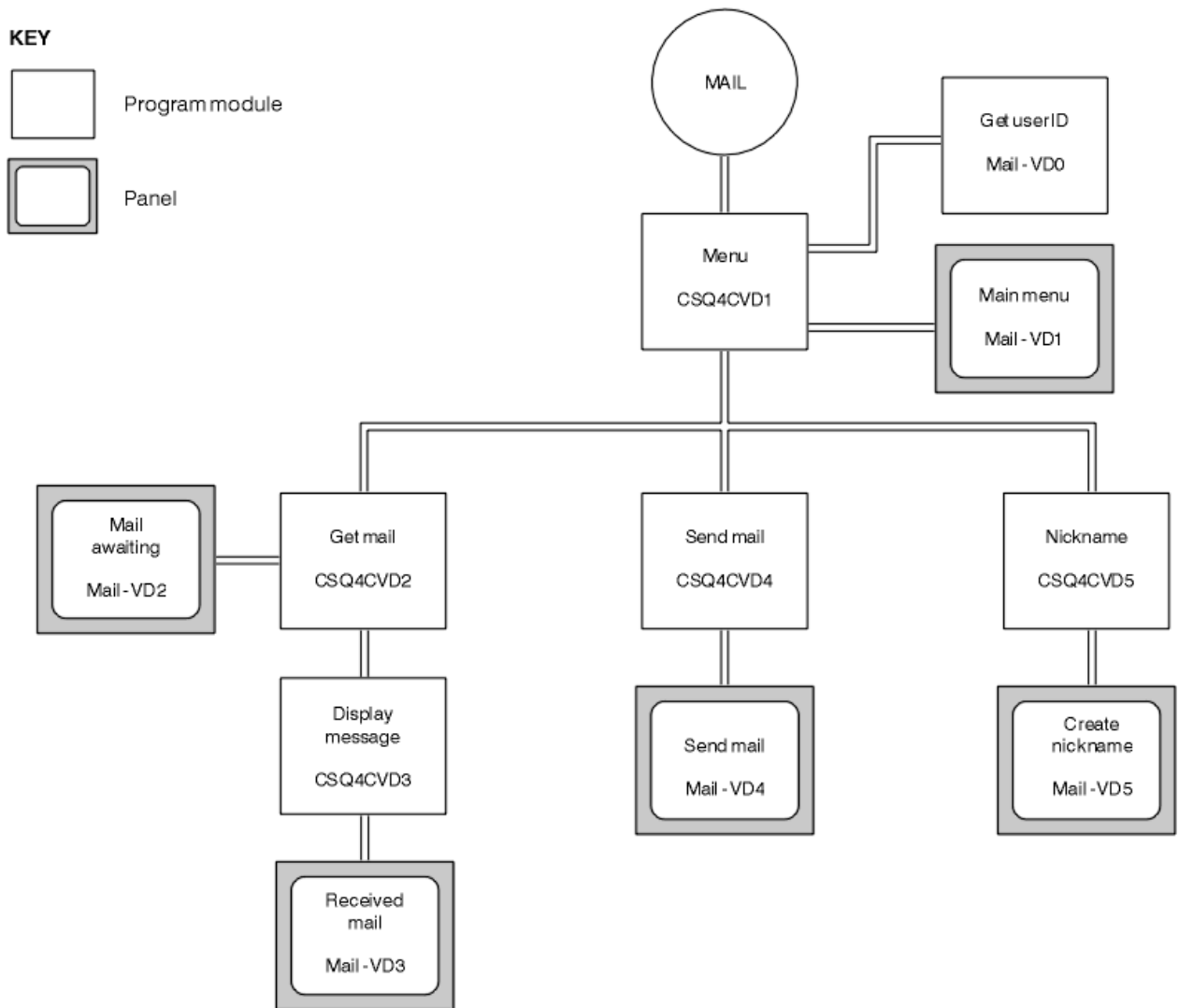


그림 158. 메일 관리자의 CICS 버전에 대한 프로그램 및 패널

z/OS z/OS의 메뉴 프로그램

TSO 환경에서 메뉴 프로그램은 CLIST에 의해 호출됩니다. CICS 환경에서 프로그램은 트랜잭션 MAIL을 통해 호출됩니다.

메뉴 프로그램(TSO용 CSQ4TVDP1, CICS용 CSQ4CVD1)은 스위트의 초기 프로그램입니다. 메뉴(TSO용 CSQ4VDP1, CICS용 VD1)를 표시하고 메뉴에서 선택되는 경우 다른 프로그램을 호출합니다.

프로그램은 먼저 사용자 ID를 확보합니다.

- 프로그램의 CICS 버전에서 사용자가 CICS에 사인온한 경우 CICS 명령 ASSIGN USERID를 사용하여 사용자 ID를 확보합니다. 사용자가 사인온하지 않은 경우에는 프로그램에서 사용자 ID를 입력하기 위한 프롬프트를 표시하도록 사인온 패널(CSQ4VD0)을 표시합니다. 이 프로그램에서는 보안 처리가 수행되지 않습니다. 사용자는 사용자 ID를 제공할 수 있습니다.
- TSO 버전에서 사용자의 ID는 CLIST의 TSO에서 확보됩니다. ISPF 공유 풀에서 변수로 메뉴 프로그램에 전달됩니다.

프로그램이 사용자 ID를 확보하고 나면, 사용자가 메일 큐(CSQ4SAMP.MAILMGR.userid). 메일 큐가 없는 경우 프로그램은 시스템 명령 입력 큐에 메시지를 넣어 큐를 작성합니다. 메시지에는 IBM MQ for z/OS 명령 DEFINE QLOCAL이 포함되어 있습니다. 이 명령이 사용하는 오브젝트 정의는 큐의 최대 용량을 9,999개 메시지로 설정합니다.

또한 프로그램은 시스템 명령 입력 큐의 응답을 처리하기 위해 임시 동적 큐를 작성합니다. 이를 수행하기 위해 프로그램은 MQOPEN 호출을 사용하고 동적 큐에 대한 템플릿으로 SYSTEM.DEFAULT.MODEL.QUEUE를 지정합

니다. 큐 관리자는 CSQ4SAMP 접두부가 있는 이름으로 임시 동적 큐를 작성합니다. 이름의 나머지 부분은 큐 관리자에서 생성합니다.

그런 다음 프로그램은 사용자의 메일 큐를 열고 현재 큐의 용량에 대해 조회하여 큐에 있는 메시지의 수를 찾습니다. 이를 수행하기 위해, 프로그램은 MQIA_CURRENT_Q_DEPTH 선택자를 지정하는 MQINQ 호출을 사용합니다.

프로그램은 메뉴를 표시하고 사용자가 작성한 선택사항을 처리하는 루프를 수행합니다. 사용자가 PF3 키를 누르면 루프가 중지됩니다. 올바른 선택을 작성한 경우 적절한 프로그램이 시작됩니다. 그렇지 않으면 오류 메시지가 표시됩니다.

z/OS에서 메일 가져오기 및 메시지 표시 프로그램

애플리케이션의 TSO 버전에서 메일 가져오기 및 메시지 표시 기능은 동일한 프로그램에 의해 수행됩니다 (CSQ4TVD2). 애플리케이션의 CICS 버전에서는, 이러한 기능이 별도의 프로그램(CSQ4CVD2 및 CSQ4CVD3)을 통해 수행됩니다.

대기 메일 패널(TSO용 CSQ4VDP2, CICS용 VD2, 예제는 [1110 페이지의 그림 159](#) 참조)은 사용자의 메일 큐에 있는 모든 메시지를 표시합니다. 이 목록을 작성하기 위해, 프로그램은 MQGET 호출을 사용하여 큐에서 모든 메시지를 찾아보며 각 메시지에 대한 정보를 저장합니다. 표시된 정보뿐만 아니라, 프로그램은 각 메시지의 MsgId 및 CorrelId를 기록합니다.

```
----- IBM MQ for z/OS Sample Programs ----- ROW 16 OF 29
COMMAND ==>
Scroll ==> PAGE
USERID - NTSFV02
Mail Manager System      QMGR - VC4
Mail Awaiting

Msg   Mail   Date   Time
No    From   Sent   Sent
16
16    Deleted
17    JOHNS   01/06/1993 12:52:02
18    JOHNS   01/06/1993 12:52:02
19    JOHNS   01/06/1993 12:52:03
20    JOHNS   01/06/1993 12:52:03
21    JOHNS   01/06/1993 12:52:03
22    JOHNS   01/06/1993 12:52:04
23    JOHNS   01/06/1993 12:52:04
24    JOHNS   01/06/1993 12:52:04
25    JOHNS   01/06/1993 12:52:05
26    JOHNS   01/06/1993 12:52:05
27    JOHNS   01/06/1993 12:52:05
28    JOHNS   01/06/1993 12:52:06
29    JOHNS   01/06/1993 12:52:06
```

그림 159. 대기 메시지 목록을 표시하는 패널의 예

대기 메일 패널에서 사용자는 하나의 메시지를 선택하고 메시지의 콘텐츠를 표시할 수 있습니다(예제는 [1111 페이지의 그림 160](#) 참조). 프로그램이 모든 메시지를 찾아볼 때 기록해 놓은 MsgId 및 CorrelId를 사용하여 큐에서 이 메시지를 제거하기 위해 프로그램은 MQGET 호출을 사용합니다. 이 MQGET 호출은 MQGMO_SYNCPOINT 옵션을 사용하여 수행됩니다. 프로그램은 메시지의 콘텐츠를 표시한 후 동기점을 선언합니다. 이는 MQGET 호출을 커밋하므로 이제 메시지는 더 이상 존재하지 않습니다.

또한 프로그램은 시스템 명령 입력 큐의 응답을 처리하기 위해 임시 동적 큐를 작성합니다.

큐 관리자가 프로그램이 예상하는 이유로 (예를 들어, 큐가 이미 있는 경우) 별명 큐를 작성할 수 없다면 프로그램은 고유 오류 메시지를 표시합니다. 큐 관리자가 프로그램이 예상하지 못한 이유로 큐를 작성할 수 없는 경우에는 프로그램은 명령 서버에 의해 프로그램에 리턴되는 최대 두 개의 오류 메시지를 표시합니다.

참고: 각 별명에 대해, 별명 프로그램은 알리어스 큐 또는 리모트 큐의 로컬 정의만 작성합니다. 이러한 큐 이름이 해석되는 로컬 큐는 별명에 포함되는 사용자 ID가 메일 관리자 애플리케이션을 시작하는 데 사용되는 경우에만 작성됩니다.

▶ z/OS z/OS에서의 신용 조회 샘플

신용 조회 샘플 애플리케이션은 IBM MQ for z/OS에서 제공한 여러 기능의 사용 방법을 보여주는 프로그램의 스위트입니다. 이는 애플리케이션의 여러 컴포넌트 프로그램이 메시지 큐잉 기술을 사용하여 서로에게 메시지를 전달할 수 있는 방법을 보여줍니다.

샘플은 독립형 CICS 애플리케이션으로 실행할 수 있습니다. 그러나 CICS 및 IMS 환경 모두에서 제공한 기능을 사용하는 메시지 큐잉 애플리케이션의 설계 방법을 보여주기 위해, IMS 배치 메시지 처리 프로그램으로 하나의 모듈이 제공됩니다. 샘플에 대한 이 확장은 1121 페이지의 『z/OS의 신용 조회 샘플에 대한 IMS 확장』에서 설명됩니다.

또한 둘 이상의 큐 관리자에서 샘플을 실행할 수 있으며, 애플리케이션의 각 인스턴스 간에 메시지를 송신할 수 있습니다. 이렇게 하려면 1121 페이지의 『z/OS에서 다중 큐 관리자가 있는 신용 조회 샘플』의 내용을 참조하십시오.

CICS 프로그램은 C 및 COBOL로 전달됩니다. 단일 IMS 프로그램은 C로만 전달됩니다. 제공된 데이터 세트는 1093 페이지의 『CICS 신용 조회 샘플』 및 1095 페이지의 『신용 조회 IMS 샘플에 대한 소스 및 JCL(C 전용)』에 표시됩니다.

애플리케이션은 은행 고객이 대출을 요청하는 경우 위험 평가 방법을 보여줍니다. 애플리케이션은 은행이 대출 요청을 처리하기 위해 두 가지 방법으로 작업하는 방법을 보여줍니다.

- 고객과 직접 거래하는 경우 은행 직원은 계좌 및 신용 위험 정보에 즉시 액세스할 수 있기를 원합니다.
- 서면 애플리케이션을 처리하는 경우 은행 직원은 계좌 및 신용 위험 정보에 대한 요청 세트를 제출하고 나중에 응답을 처리할 수 있습니다.

메시지 큐잉 기술이 지워지도록 애플리케이션의 재무 및 보안 세부사항은 단순하게 유지됩니다.

▶ z/OS z/OS에서의 신용 조회 샘플 준비 및 실행

신용 조회 샘플을 준비하고 실행하려면, 다음 단계를 수행하십시오.

1. 일부 예제 계정에 대한 정보를 보유하는 VSAM 데이터 세트를 작성하십시오. 데이터 세트 CSQ4FILE에 제공된 JCL을 편집하고 실행하여 이를 수행하십시오.
2. 1091 페이지의 『환경에 CICS 대한 샘플 애플리케이션 준비 z/OS』의 단계를 수행하십시오. (샘플에 대한 IMS 확장을 사용하려는 경우 반드시 수행해야 하는 추가 단계는 1121 페이지의 『z/OS의 신용 조회 샘플에 대한 IMS 확장』에서 설명됩니다.)
3. CKTI 트리거 모니터 시작 (IBM MQ for z/OS와 함께 제공됨) 큐 CSQ4SAMP.INITIATION.QUEUE, CICS 트랜잭션 CKQC 사용.
4. 애플리케이션을 시작하려면, CICS 시스템을 시작하고 MVB1 트랜잭션을 사용하십시오.
5. 첫 번째 패널에서 **즉시** 또는 **배치** 조회를 선택하십시오.

즉시 및 배치 조회 패널은 유사합니다. 1113 페이지의 그림 161에서 즉시 조회 패널을 표시합니다.

```

CSQ4VB2      IBM MQ for z/OS Sample Programs

Credit Check - Immediate Inquiry

Specify details of the request, then press Enter.
Name . . . . .
Social security number ____
Bank account name . . ____
Account number . . . ____
Amount requested : . . 012345
Response from CHECKING ACCOUNT for name : ____
Account information not found
Credit worthiness index - NOT KNOWN
..
..
..
..
..
..
..
..
..
MESSAGE LINE
F1=Help F3=Exit F5=Make another inquiry

```

그림 161. 신용 조회 샘플 애플리케이션에 대한 즉시 조회 패널

6. 적절한 필드에서 은행 계좌 번호 및 대출 금액을 입력하십시오. 이러한 필드에 입력할 정보에 대한 안내는 [1113 페이지의 『조회 패널에 정보 입력』](#)의 내용을 참조하십시오.

조회 패널에 정보 입력

신용 조회 샘플 애플리케이션은 조회 패널의 **요청 금액** 필드에 입력한 데이터가 정수 양식인지 확인합니다.

다음 계좌 번호 중 하나를 입력하는 경우 애플리케이션은 VSAM 데이터 세트 CSQ4BAQ에서 해당 계좌 이름, 평균 계좌 잔액 및 신용도 색인을 찾습니다.

- 2222222222
- 3111234329
- 3256478962
- 3333333333
- 3501676212
- 3696879656
- 4444444444
- 5555555555
- 6666666666
- 7777777777

다른 필드에는 임의의 정보를 입력하거나 정보를 입력하지 않을 수 있습니다. 애플리케이션은 사용자가 입력하는 정보를 보유하며 애플리케이션에서 생성하는 보고서에 동일한 정보를 리턴합니다.

▶ z/OS z/OS에서의 신용 조회 샘플 디자인

이 절은 신용 조회 샘플 애플리케이션을 구성하는 각 프로그램의 디자인을 설명합니다.

애플리케이션을 설계하는 동안 고려되는 일부 기술에 대한 자세한 정보는 [1119 페이지의 『z/OS에서 신용 조회 샘플의 디자인 고려사항』](#)의 내용을 참조하십시오.

[1114 페이지의 그림 162](#)에서는 애플리케이션을 구성하는 프로그램 및 이러한 프로그램이 제공하는 큐도 표시합니다. 이 그림에서 그림을 보다 쉽게 이해할 수 있도록 CSQ4SAMP 접두부가 모든 큐 이름에서 생략되었습니다.

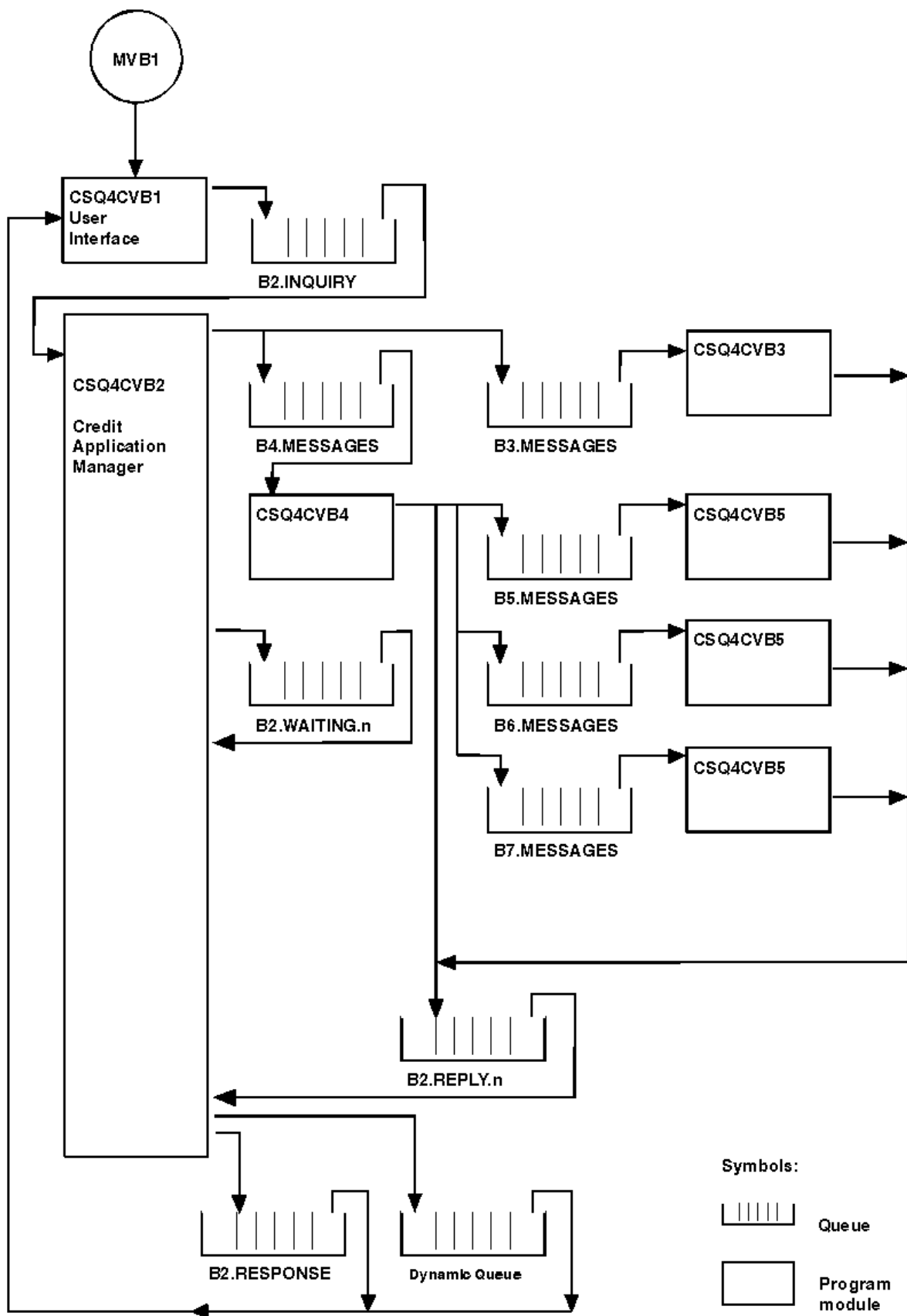


그림 162. 신용 조회 샘플 애플리케이션에 대한 프로그램 및 큐(COBOL 프로그램 전용)

▶ z/OS

z/OS에서의 사용자 인터페이스 프로그램(CSQ4CVB1)

대화식 모드 CICS 트랜잭션 MVB1을 시작하면 애플리케이션에 대한 사용자 인터페이스 프로그램이 시작됩니다.

이 프로그램은 조회 메시지를 CSQ4SAMP.B2.INQUIRY 큐에 넣고 조회를 작성할 때 지정하는 응답 대상 큐에서 이러한 조회에 대한 응답을 가져옵니다. 사용자 인터페이스에서 즉시 또는 배치 조회를 제출할 수 있습니다.

- 즉시 조회의 경우 프로그램은 응답 대상 큐로 사용하는 임시 동적 큐를 작성합니다. 이는 각 조회에 고유 응답 큐가 있음을 의미합니다.
- 배치 조회의 경우, 사용자 인터페이스 프로그램은 CSQ4SAMP.B2.RESPONSE 큐에서 응답을 가져옵니다. 단 순성을 위해 프로그램은 이 하나의 응답 대상 큐에서 해당 조회 전체에 대한 응답을 가져옵니다. 은행은 시작했던 조회에 대해서만 각각 응답을 볼 수 있도록 MVB1의 각 사용자에 대한 별도의 응답 대상 큐를 사용하려고 할 수 있습니다.

배치 및 즉시 모드일 때 애플리케이션에 사용된 메시지 특성 간의 중요한 차이점은 다음과 같습니다.

- 배치 작동의 경우, 메시지의 우선순위가 낮으므로 즉시 모드로 입력된 대출 요청이 있다면 요청 후에 처리됩니다. 또한 메시지가 지속적이므로 애플리케이션 또는 큐 관리자를 재시작해야 하는 경우 복구됩니다.
- 즉시 작동의 경우, 메시지의 우선순위가 높으므로 배치 모드로 입력된 대출 요청이 있다면 요청 전에 처리됩니다. 또한 메시지가 지속적이지 않으므로 애플리케이션 또는 큐 관리자를 재시작해야 하는 경우 제거됩니다.

그러나 모든 경우에 대출 요청 메시지의 특성은 애플리케이션 전체에 전파됩니다. 따라서, 예를 들어, 우선순위가 높은 요청의 결과로 발생하는 모든 메시지도 높은 우선순위를 갖습니다.

▶ z/OS

z/OS에서의 CAM(Credit Application Manager)(CSQ4\$CVB2)

CAM(Credit Application Manager) 프로그램은 신용 조회 애플리케이션에 대한 대부분의 처리를 수행합니다.

트리거 이벤트가 CSQ4SAMP.B2.INQUIRY 큐 또는 CSQ4SAMP.B2.REPLY 큐에서 발생하는 경우 (IBM MQ for z/OS와 함께 제공된) CKTI 트리거 모니터에 의해 CAM이 시작됩니다. n 여기서 n 은 일련의 응답 큐 중 하나를 식별하는 정수입니다. 트리거 메시지에는 트리거 이벤트가 발생한 큐의 이름을 포함하는 데이터가 포함되어 있습니다.

CAM은 처리 중인 조회에 대한 정보를 저장하기 위해 CSQ4SAMP.B2.WAITING. n 양식의 이름으로 큐를 사용합니다. 응답 대상 큐와 각가 쌍을 이루도록 큐의 이름이 지정됩니다. 예를 들어, CSQ4SAMP.B2.WAITING.3 큐에는 특정 조회에 대한 입력 데이터가 포함되고 CSQ4SAMP.B2.REPLY.3 큐에는 해당 조회와 관련된 모든 응답 메시지 세트가 포함됩니다. 이렇게 설계된 이유에 대해 살펴보려면 [1119 페이지의 『별도의 조회와 CAM의 응답 큐』](#)의 내용을 참조하십시오.

시작 논리

트리거 이벤트가 CSQ4SAMP.B2.INQUIRY 큐에서 발생하는 경우 CAM은 공유 액세스를 위해 큐를 엽니다. 그런 다음 빈 큐를 찾을 때까지 각 응답 큐를 열려고 시도합니다. 빈 응답 큐를 찾을 수 없는 경우 CAM은 사실을 기록하고 정상적으로 종료합니다.

트리거 이벤트가 CSQ4SAMP.B2.REPLY. n 큐에서 발생하는 경우 CAM은 독점 액세스를 위해 큐를 엽니다. 오브젝트가 이미 사용 중이라는 리턴 코드가 보고되면 CAM은 정상적으로 종료됩니다. 다른 오류가 발생하는 경우에는 CAM은 오류를 기록하고 종료합니다. CAM은 해당하는 대기 큐 및 조회 큐를 연 후 메시지 가져오기 및 처리를 시작합니다. 대기 큐에서 CAM은 부분적으로 완료된 조회의 세부사항을 복구합니다.

이 샘플의 단순성을 위해 사용된 큐의 이름이 프로그램에 보관됩니다. 비즈니스 환경에서 큐 이름은 프로그램에서 액세스한 파일에 보관됩니다.

조회 큐에서 메시지 가져오기

CAM은 먼저 MQGMO_SET_SIGNAL 옵션과 함께 MQGET 호출을 사용하여 조회 큐에서 메시지를 가져오려고 시도합니다. 메시지가 바로 사용 가능한 경우 메시지가 처리됩니다. 메시지를 사용할 수 없는 경우 신호가 설정됩니다.

그런 다음 CAM은 동일한 옵션과 함께 MQGET 호출을 다시 사용하여 응답 큐에서 메시지를 가져오려고 시도합니다. 메시지가 바로 사용 가능한 경우 메시지가 처리됩니다. 그렇지 않은 경우 신호가 설정됩니다.

두 신호가 모두 설정되는 경우 프로그램은 신호 중 하나가 게시될 때까지 대기합니다. 메시지가 사용 가능하다는 것을 표시하기 위해 신호가 게시되면 메시지가 검색되고 처리됩니다. 신호가 만료되거나 큐 관리자가 종료되는 경우 프로그램이 종료됩니다.

CAM에서 검색하는 메시지 처리

CAM에서 검색한 메시지는 다음 네 가지 유형 중 하나일 수 있습니다.

- 조회 메시지
- 응답 메시지
- 전파 메시지
- 예상치 못한 또는 원치 않는 메시지

CAM은 [1116 페이지](#)의 『z/OS에서 CAM이 검색한 메시지 처리』에서 설명한 대로 이러한 메시지를 처리합니다.

응답 송신

CAM에서 모든 응답을 수신한 경우 조회에서 응답을 처리하고 단일 응답 메시지를 작성하기를 예상합니다. 동일한 *CorrelId*를 갖는 모든 응답 메시지의 모든 데이터를 하나의 메시지로 통합합니다. 이 응답을 원래 대출 요청에 지정된 응답 대상 큐에 넣습니다. 마지막 응답 메시지의 검색을 포함하는 동일한 작업 단위 내에 응답 메시지를 넣습니다. 이는 CSQ4SAMP.B2.WAITING.n 큐에 완료된 메시지가 없는지 확인하여 복구를 간소화하는 것입니다.

부분적으로 완료된 조회의 복구

CAM은 큐 CSQ4SAMP.B2.WAITING.n에 수신하는 모든 메시지를 복사합니다. 메시지 디스크립터의 필드를 다음과 같이 설정합니다.

- *Priority*는 메시지의 유형에 의해 판별됩니다.
 - 요청 메시지의 경우, *priority* = 3
 - 데이터그램의 경우, *priority* = 2
 - 응답 메시지의 경우, *priority* = 1
- *CorrelId*는 대출 요청 메시지의 *MsgId*로 설정됩니다.
- 다른 MQMD 필드는 수신된 메시지의 필드에서 복사됩니다.

조회가 완료된 경우 특정 조회에 대한 메시지는 응답을 처리하는 동안 대기 큐에서 제거됩니다. 따라서 언제든지 대기 큐는 처리 중인 조회와 관련된 모든 메시지를 포함하고 있습니다. 이러한 메시지는 프로그램을 재시작해야 하는 경우 진행 중인 조회의 세부사항을 복구하는 데 사용됩니다. 전파 또는 응답 메시지 전에 조회 메시지가 복구되도록 다른 우선순위가 설정됩니다.

z/OS에서 CAM이 검색한 메시지 처리

CAM(Credit Application Manager)이 검색한 메시지는 다음 네 가지 유형 중 하나가 될 수 있습니다. CAM이 메시지를 처리하는 방식은 유형에 따라 달라집니다.

CAM에서 검색한 메시지는 다음 네 가지 유형 중 하나일 수 있습니다.

- 조회 메시지
- 응답 메시지
- 전파 메시지
- 예상치 못한 또는 원치 않는 메시지

CAM은 다음과 같이 이러한 메시지를 처리합니다.

조회 메시지

조회 메시지는 사용자 인터페이스 프로그램에서 옵니다. 각 대출 요청에 대한 조회 메시지를 작성합니다.

모든 대출 요청의 경우, CAM은 고객의 당좌 예금 구좌의 평균 잔액을 요청합니다. 알리어스 큐 CSQ4SAMP.B2.OUTPUT.ALIAS에 요청 메시지를 넣어 이를 수행합니다. 이 큐 이름은 CSQ4SAMP.B3.MESSAGES 큐로 해석되며 당좌 예금 계좌 프로그램 CSQ4CVB3에서 처리합니다. CAM이 이 알리어스 큐에 메시지를 넣는 경우 응답 대상 큐에 대해 적절한 CSQ4SAMP.B2.REPLY.n 큐를 지정합니다. CSQ4CVB3 프로그램이 다른 이름의 기본 큐를 처리하는 다른 프로그램에 의해 쉽게 대체될 수 있도록 여기서 알리어스 큐가 사용됩니다. 이를 수행하기 위해, 해당 이름이 새 큐로 해석되도록 알리어스 큐를 재정의합니다. 또한 알리어스 큐 및 기본 큐에 서로 다른 액세스 권한을 지정할 수 있습니다.

사용자가 10,000 단위보다 더 큰 대출을 요청하는 경우 CAM은 다른 데이터베이스에 대한 확인을 시작합니다. CSQ4SAMP.B4.MESSAGES 큐에 요청 메시지를 넣어 이를 수행하며 분산 프로그램 CSQ4CVB4에 의해 처리됩니다. 이 큐를 제공하는 프로세스는 신용카드 내역, 예금 및 부동산 담보 대출 납부액과 같은 다른 레코드에 액세스 권한이 있는 프로그램에서 제공한 큐에 메시지를 전파합니다. 이러한 프로그램의 데이터가 Put 조작에서 지정된 응답 대상 큐로 리턴됩니다. 또한 얼마나 많은 전파 메시지가 송신되었는지 지정하기 위해 이 프로그램에 의해 전파 메시지가 응답 대상 큐로 송신됩니다.

비즈니스 환경에서 분산 프로그램은 은행 계정의 각기 다른 유형에 필요한 형식과 일치하도록 제공된 데이터를 재형식화합니다.

참조하는 큐는 원격 시스템에 있을 수 있습니다.

각 조회 메시지의 경우 CAM은 메모리 상주 IRT(Inquiry Record Table)의 입력 항목을 시작합니다. 이 레코드는 다음을 포함합니다.

- 조회 메시지의 MsgId
- ReplyExp 필드에서, 예상 응답 수(송신된 메시지 수와 같음)
- ReplyRec 필드에서, 수신된 응답 수((이 단계에서는 0)
- PropsOut 필드에서, 전파 메시지가 예상되는지 표시

CAM은 조회 메시지를 대기 큐에 복사합니다.

- Priority를 3으로 설정함
- CorrelId가 조회 메시지의 MsgId로 설정됨
- 다른 메시지 디스크립터 필드는 조회 메시지의 필드로 설정됩니다.

전파 메시지

전파 메시지는 분산 프로그램이 조회를 전달한 대상 큐의 수를 포함합니다. 해당 메시지는 다음과 같이 처리됩니다.

1. 송신된 메시지의 수를 IRT에 있는 적절한 레코드의 ReplyExp 필드에 추가하십시오. 이 정보는 메시지에 있습니다.
2. IRT에 있는 레코드의 ReplyRec 필드가 1씩 증가합니다.
3. IRT에 있는 레코드의 PropsOut 필드가 1씩 감소합니다.
4. 대기 큐로 메시지를 복사하십시오. CAM은 Priority를 2로 설정하고 메시지 디스크립터의 다른 필드를 전파 메시지의 필드로 설정합니다.

응답 메시지

응답 메시지에는 당좌 예금 계좌 프로그램 또는 에이전시 조회 프로그램에 대한 요구 중 하나에 대한 응답을 포함합니다. 응답 메시지는 다음과 같이 처리됩니다.

1. IRT에 있는 레코드의 ReplyRec 필드가 1씩 증가합니다.
2. Priority를 1로 설정하고 메시지 디스크립터의 다른 필드를 응답 메시지의 필드로 설정하여 대기 큐에 메시지를 복사하십시오.
3. ReplyRec = ReplyExp 이고 PropsOut = 0인 경우 MsgComplete 플래그를 설정하십시오.

문제점 진단을 위해

애플리케이션은 다른 메시지를 예상하지 않습니다. 그러나 애플리케이션은 시스템에 의해 브로드캐스트되는 메시지를 수신하거나 알 수 없는 CorrelIds를 가진 메시지에 응답할 수 있습니다.

CAM은 이러한 메시지를 메시지를 조사할 수 있는 CSQ4SAMP.DEAD.QUEUE 큐에 넣습니다. 이 넣기 조작에 실패하는 경우 메시지가 유실되고 프로그램은 계속됩니다. 프로그램의 이 파트에 대한 설계 관련 자세한 정보는 1120 페이지의 『샘플이 예상치 못한 메시지를 처리하는 방법』의 내용을 참조하십시오.

▶ z/OS z/OS의 당좌 예금 프로그램(CSQ4CVB3)

CSQ4SAMP.B3.MESSAGES 큐의 트리거 이벤트에 의해 당좌 예금 프로그램이 시작됩니다. 큐를 열고 나서, 이 프로그램은 대기 옵션과 함께 대기 간격을 30초로 설정해서 MQGET 호출을 사용하여 큐에서 메시지를 가져옵니다.

프로그램은 대출 요청 메시지에서 은행 계좌 번호에 대해 VSAM 데이터 세트 CSQ4BAQ를 검색합니다. 해당 계좌 이름, 평균 잔액 및 신용도 색인을 검색하고 계좌 번호가 데이터 세트에 없는지 참고합니다.

그런 다음 프로그램은 (MQPUT1 호출을 사용하여) 응답 메시지를 대출 요청 메시지에서 이름 지정된 응답 대상 큐에 넣습니다. 이 응답 메시지의 경우 프로그램은 다음을 수행합니다.

- 대출 요청 메시지의 CorrelId 복사
- MQPMO_PASS_IDENTITY_CONTEXT 옵션 사용

대기 간격이 만료될 때까지 프로그램은 큐에서 계속 메시지를 가져옵니다.

▶ z/OS z/OS의 분산 프로그램(CSQ4CVB4)

분산 프로그램은 CSQ4SAMP.B4.MESSAGES 큐의 트리거 이벤트에 의해 시작됩니다.

신용카드 내역, 예금 및 부동산 담보 대출 납부액과 같은 레코드에 액세스 권한이 있는 다른 에이전시로 대출 요청의 분산을 시뮬레이션하기 위해, 프로그램은 이름 목록 CSQ4SAMP.B4.NAMELIST에 있는 모든 큐에 동일한 메시지의 사본을 넣습니다. 이름이 CSQ4SAMP.B *n*.MESSAGES 양식인 3개의 큐가 있습니다(여기서 *n*은 5, 6 또는 7). 비즈니스 애플리케이션에서는 에이전시가 별도의 위치에 있을 수 있으므로 이러한 큐는 리모트 큐일 수 있습니다. 샘플 애플리케이션을 이를 표시하도록 수정하려는 경우 1121 페이지의 『z/OS에서 다중 큐 관리자가 있는 신용 조회 샘플』의 내용을 참조하십시오.

분산 프로그램은 다음 단계를 수행합니다.

1. 이름 목록에서 프로그램이 사용하기 위한 큐의 이름을 가져옵니다. 프로그램은 MQINQ 호출로 이를 수행하여 이름 목록 오브젝트의 속성에 대해 조회합니다.
2. 이러한 큐 및 CSQ4SAMP.B4.MESSAGES를 엽니다.
3. CSQ4SAMP.B4.MESSAGES 큐에 더 이상 메시지가 없을 때까지 다음 루프를 수행합니다.
 - a. 대기 옵션과 30초로 대기 간격을 설정한 MQGET 호출을 사용하여 메시지를 가져오십시오.
 - b. 이름 목록에 나열된 각 큐에 메시지를 넣고 응답 대상 큐에 대한 적절한 CSQ4SAMP.B2.REPLY.*n* 큐의 이름을 지정합니다. 프로그램은 대출 요청 메시지의 CorrelId를 이러한 복사 메시지로 복사하고 MQPUT 호출에서 MQPMO_PASS_IDENTITY_CONTEXT 옵션을 사용합니다.
 - c. 성공적으로 넣은 메시지 수를 표시하기 위해 데이터그램 메시지를 CSQ4SAMP.B2.REPLY.*n* 큐에 송신하십시오.
 - d. 동기점을 선언하십시오.

▶ z/OS z/OS의 에이전시 조회 프로그램(CSQ4CVB5/CSQ4CCB5)

에이전시 조회 프로그램은 COBOL 프로그램 및 C 프로그램 두 가지 모두로 제공됩니다. 두 프로그램은 동일한 디자인입니다. 이는 서로 다른 유형의 프로그램이 IBM MQ 애플리케이션 내에서 쉽게 공존할 수 있으며 이러한 애플리케이션을 구성하는 프로그램 모듈을 쉽게 대체할 수 있음을 보여줍니다.

프로그램의 인스턴스는 이러한 큐에서 트리거 이벤트에 의해 시작됩니다.

- COBOL 프로그램의 경우(CSQ4CVB5):
 - CSQ4SAMP.B5.MESSAGES
 - CSQ4SAMP.B6.MESSAGES
 - CSQ4SAMP.B7.MESSAGES
- C 프로그램의 경우(CSQ4CCB5), CSQ4SAMP.B8.MESSAGES 큐

참고: C 프로그램을 사용하려는 경우 CSQ4SAMP.B7.MESSAGES를 CSQ4SAMP.B8.MESSAGES로 바꾸려면 CSQ4SAMP.B4.NAMELIST 이름 목록의 정의를 대체해야 합니다. 이를 수행하기 위해 다음 중 하나를 사용할 수 있습니다.

- IBM MQ for z/OS 조작 및 제어판
- [ALTER NAMELIST 명령](#)
- [CSQUTIL 유틸리티](#)

적절한 큐를 얻 후에 이 프로그램은 대기 옵션과 30초로 대기 간격을 설정하여 MQGET 호출로 큐에서 메시지를 가져옵니다.

프로그램은 대출 요청 메시지에 전달된 계좌 번호에 대한 VSAM 데이터 세트 CSQ4BAQ를 검색하여 에이전시의 데이터베이스 검색을 시뮬레이션합니다. 그런 다음 제공되는 큐의 이름 및 신용도 색인을 포함하는 응답을 빌드합니다. 처리를 간소화하기 위해 신용도 색인이 무작위로 선택됩니다.

응답 메시지를 넣을 때 프로그램은 MQPUT1 호출과 다음을 사용합니다.

- 대출 요청 메시지의 CorrelId 복사
- MQPMO_PASS_IDENTITY_CONTEXT 옵션 사용

프로그램은 응답 메시지를 대출 요청 메시지에서 이름 지정된 응답 대상 큐로 송신합니다. (응답 대상 큐를 소유하는 큐 관리자의 이름도 대출 요청 메시지에 지정됩니다.)

z/OS에서 신용 조회 샘플의 디자인 고려사항

신용 조회 샘플에 대한 디자인 고려사항입니다.

이 주제에는 다음에 대한 정보가 포함되어 있습니다.

- [1119 페이지의 『별도의 조회와 CAM의 응답 큐』](#)
- [1119 페이지의 『샘플이 오류를 처리하는 방법』](#)
- [1120 페이지의 『샘플이 예상치 못한 메시지를 처리하는 방법』](#)
- [1120 페이지의 『샘플이 동기점을 사용하는 방법』](#)
- [1120 페이지의 『샘플이 메시지 컨텍스트 정보를 사용하는 방법』](#)
- [1121 페이지의 『CAM의 메시지 ID와 상관 ID 사용』](#)

별도의 조회와 CAM의 응답 큐

애플리케이션은 조회 및 응답 모두에 대해 단일 큐를 사용할 수 있지만 다음과 같은 이유로 별도의 큐를 사용하도록 설계되었습니다.

- 프로그램이 최대 조회 수를 처리 중인 경우 추가 조회가 큐에 남아 있을 수 있습니다. 단일 큐가 사용되는 경우 큐에서 제거되어야 하며 다른 곳에 저장됩니다.
- 메시지 트래픽이 충분히 높은 경우 CAM의 다른 인스턴스는 동일한 조회 서비스를 제공하기 위해 자동으로 시작될 수 있습니다. 그러나 프로그램은 진행 중인 조회를 추적해야 하고 이를 수행하기 위해서는 시작한 조회의 모든 응답을 다시 확인해야 합니다. 단 하나의 큐를 사용하는 경우 프로그램은 메시지가 이 프로그램에 대한 것인지 또는 다른 프로그램에 대한 것인지 확인하기 위해 메시지를 찾아봐야 합니다. 이는 조작을 덜 효율적이게 합니다.

애플리케이션은 여러 CAM을 지원할 수 있으며 쌍을 이룬 응답 대상 큐 및 대기 큐를 사용하여 진행 중인 조회를 효과적으로 복구할 수 있습니다.

- 프로그램은 신호보내기를 사용하여 여러 큐에서 효과적으로 대기할 수 있습니다.

샘플이 오류를 처리하는 방법

사용자 인터페이스 프로그램은 오류를 사용자에게 직접 보고하여 오류를 처리합니다.

다른 프로그램에는 사용자 인터페이스가 없으므로 다른 방식으로 오류를 처리해야 합니다. 또한 여러 상황에서 (예를 들어, MQGET 호출에 실패하는 경우) 이러한 다른 프로그램은 애플리케이션 사용자의 ID를 알지 못합니다.

다른 프로그램은 메시지를 CSQ4SAMP라는 CICS 임시 스토리지 큐에 넣습니다. CICS 제공 트랜잭션 CEBR을 사용하여 이 큐를 찾아볼 수 있습니다. 또한 프로그램은 오류 메시지를 CICS CSML 로그에 씁니다.

샘플이 예상치 못한 메시지를 처리하는 방법

메시지 큐잉 애플리케이션을 설계하는 경우 예상치 못하게 큐에 도착하는 메시지를 어떻게 처리할지 결정해야 합니다.

두 가지 기본 선택사항은 다음과 같습니다.

- 예상치 못한 메시지가 처리될 때까지 애플리케이션이 추가 작업을 수행하지 않습니다. 이는 아마도 애플리케이션이 운영자에게 이를 알리고, 자체 종료하며 자동으로 재시작되지 않는지 확인하는 것을 의미합니다(트리거링 설정을 꺼서 이를 수행할 수 있습니다). 이 선택사항은 애플리케이션에 대한 모든 처리가 예상치 못한 하나의 메시지로 중단될 수 있으며 애플리케이션을 재시작하기 위해 운영자의 개입이 필요하다는 것을 의미합니다.
- 애플리케이션이 제공하고 있는 큐에서 메시지를 제거하고 다른 위치에 메시지를 넣은 후 처리를 계속합니다. 이 메시지를 넣을 최적의 위치는 시스템 데드-레터 큐입니다.

두 번째 옵션을 선택하는 경우:

- 운영자 또는 다른 프로그램은 메시지가 어디에서 수신되는지 확인하기 위해 데드-레터 큐에 넣어진 메시지를 조사해야 합니다.
- 예상치 못한 메시지를 데드-레터 큐에 넣을 수 없는 경우 이 메시지는 유실됩니다.
- 예상치 못한 메시지가 데드-레터 큐의 메시지 한계보다 길거나 프로그램의 버퍼 크기보다 긴 경우 메시지가 잘립니다.

애플리케이션이 외부 활동에서 최소의 영향으로 모든 조회를 원활하게 처리하는지 확인하기 위해 신용 조회 샘플 프로그램은 두 번째 옵션을 사용합니다. 샘플을 동일한 큐 관리자를 사용하는 다른 애플리케이션과 별도로 유지할 수 있으려면, 신용 조회 샘플에서 시스템 데드-레터 큐를 사용하지 않습니다. 대신 고유 데드-레터 큐를 사용합니다. 이 큐는 CSQ4SAMP.DEAD.QUEUE로 이름이 지정됩니다. 샘플은 샘플 프로그램에 대해 제공된 버퍼 영역보다 긴 메시지를 자릅니다. 찾아보기 샘플 애플리케이션을 사용하여 이 큐에서 메시지를 찾아보거나 메시지 출력 샘플 애플리케이션을 사용하여 해당 메시지 디스크립터와 함께 메시지를 출력할 수 있습니다.

그러나 샘플을 둘 이상의 큐 관리자에서 실행하도록 확장하는 경우 예상치 못한 메시지 또는 전달할 수 없는 메시지를 큐 관리자에서 시스템 데드-레터 큐에 넣을 수 있습니다.

샘플이 동기점을 사용하는 방법

신용 조회 샘플 애플리케이션의 프로그램은 다음을 확인하기 위해 동기점을 선언합니다.

- 단 하나의 응답 메시지가 각각의 예쌍 메시지에 대한 응답으로 송신됩니다.
- 예상치 못한 메시지의 여러 사본을 샘플의 데드-레터 큐로는 넣을 수 없습니다.
- CAM은 대기 큐에서 지속 메시지를 가져와서 부분적으로 완료된 모든 조회의 상태를 복구할 수 있습니다.

이를 달성하기 위해 단일 작업 단위가 메시지 가져오기, 해당 메시지 처리 및 후속 Put 조작을 처리하는 데 사용됩니다.

샘플이 메시지 컨텍스트 정보를 사용하는 방법

사용자 인터페이스 프로그램(CSQ4CVB1)에서 메시지를 송신할 때, MQPMO_DEFAULT_CONTEXT 옵션을 사용합니다. 큐 관리자가 ID 및 원래 컨텍스트 정보를 모두 생성한다는 것을 의미합니다. 큐 관리자는 프로그램(MVB1)을 시작한 트랜잭션 및 트랜잭션을 시작한 사용자 ID에서 이 정보를 가져옵니다.

CAM에서 조회 메시지를 송신할 때 MQPMO_PASS_IDENTITY_CONTEXT 옵션을 사용합니다. 이는 넣을 메시지의 ID 컨텍스트 정보가 원래 조회 메시지의 ID 컨텍스트에서 복사된다는 것을 의미합니다. 이 옵션을 사용하여 원본 컨텍스트 정보가 큐 관리자에 의해 생성됩니다.

CAM에서 응답 메시지를 송신할 때 MQPMO_ALTERNATE_USER_AUTHORITY 옵션을 사용합니다. 이로 인해 CAM에서 응답 대상 큐를 열 때 큐 관리자는 보안 검사를 위해 대체 사용자 ID를 사용하게 됩니다. CAM은 원래 조회 메시지 제출자의 사용자 ID를 사용합니다. 이는 사용자가 만든 조회에 해당하는 응답만 볼 수 있다는 것을 의미합니다. 대체 사용자 ID는 원래 조회 메시지의 메시지 디스크립터에 있는 ID 컨텍스트 정보에서 확보됩니다.

조회 프로그램(CSQ4CVB3/4/5)에서 응답 메시지를 송신할 때 MQPMO_PASS_IDENTITY_CONTEXT 옵션을 사용합니다. 이는 넣을 메시지의 ID 컨텍스트 정보가 원래 조회 메시지의 ID 컨텍스트에서 복사된다는 것을 의미합니다. 이 옵션을 사용하여 원본 컨텍스트 정보가 큐 관리자에 의해 생성됩니다.

참고: MVB3/4/5 트랜잭션과 연관된 사용자 ID에는 B2.REPLY.n 큐에 대한 액세스 권한이 필요합니다. 이러한 사용자 ID는 처리 중인 요청과 연관된 ID와 동일하지 않을 수 있습니다. 이 가능한 보안 노출을 해결하기 위해, 조회 프로그램은 응답을 넣을 때 MQPMO_ALTERNATE_USER_AUTHORITY 옵션을 사용할 수 있습니다. 이는 MVB1의 개별 사용자 각각 B2.REPLY.n 큐를 열기 위한 권한이 필요하다는 것을 의미합니다.

CAM의 메시지 ID와 상관 ID 사용

애플리케이션은 한 번에 처리되는 모든 라이브 조회의 진행상태를 모니터해야 합니다. 이를 수행하기 위해, 각 조회가 있는 모든 정보와 연관된 각 대출 요청 메시지의 고유한 메시지 ID를 사용합니다.

CAM은 조회 메시지의 MsgId를 해당 조회에 대해 송신하는 모든 요청 메시지의 CorrelId에 복사합니다. 샘플(CSQ4CVB3 - 5)의 다른 프로그램은 수신하는 각 메시지의 CorrelId를 응답 메시지의 CorrelId에 복사합니다.

z/OS에서 다중 큐 관리자가 있는 신용 조회 샘플

신용 조회 샘플 애플리케이션을 사용하여 두 개의 큐 관리자 및 CICS 시스템(각 큐 관리자가 다른 CICS 시스템에 연결됨)에 샘플을 설치해서 분산 큐잉을 시연할 수 있습니다.

샘플 프로그램이 설치되어 있고 트리거 모니터(CKTI)가 각 시스템에서 실행 중인 경우 다음을 수행해야 합니다.

1. 두 개의 큐 관리자 간에 통신 링크를 설정하십시오. 이를 수행하는 방법에 대한 정보는 [분산 큐잉 구성](#)을 참조하십시오.
2. 하나의 큐 관리자에서 사용하려는 (다른 큐 관리자의) 각 리모트 큐에 대한 로컬 정의를 작성하십시오. 이러한 큐는 CSQ4SAMP.B n.MESSAGES일 수 있으며 여기서 n은 3, 5, 6 또는 7입니다. (당좌 예금 계좌 프로그램 및 에이전시 조회 프로그램에서 제공하는 큐가 있습니다.) 이를 수행하는 방법에 대한 정보는 [DEFINE QREMOTE](#) 및 [DEFINE 큐의 내용](#)을 참조하십시오.
3. 사용하려는 리모트 큐의 이름을 포함하도록 이름 목록(CSQ4SAMP.B4.NAMELIST)의 정의를 변경하십시오. 이를 수행하는 방법에 대한 정보는 [DEFINE NAMELIST](#)의 내용을 참조하십시오.

z/OS의 신용 조회 샘플에 대한 IMS 확장

당좌 예금 계좌 프로그램의 버전이 IMS 배치 메시지 처리(BMP) 프로그램으로 제공됩니다. C 언어로 작성되어 있습니다.

이 프로그램은 버전과 CICS 동일한 기능을 수행하지만, 계정 정보를 가져올 때 VSAM 파일 대신 데이터베이스를 IMS 읽는다는 점이 다릅니다. 당좌 예금 계좌 프로그램의 CICS 버전을 IMS 버전으로 대체하는 경우 애플리케이션을 사용하는 방식에는 차이가 없습니다.

IMS 버전을 준비하고 실행하려면, 다음을 수행해야 합니다.

1. [1112 페이지의 『z/OS에서의 신용 조회 샘플 준비 및 실행』](#)의 단계를 수행하십시오.
2. [1094 페이지의 『환경에 IMS 대한 샘플 애플리케이션 준비 z/OS』](#)의 단계를 수행하십시오.
3. (CSQ4SAMP.B3.MESSAGES 대신에) CSQ4SAMP.B3.IMS.MESSAGES 큐로 해석하기 위해 알리어스 큐 CSQ4SAMP.B2.OUTPUT.ALIAS의 정의를 대체하십시오. 이를 수행하기 위해 다음 중 하나를 사용할 수 있습니다.

- IBM MQ for z/OS 조작 및 제어판
- [ALTER QALIAS](#) 명령

IMS 당좌 예금 계좌 프로그램을 사용하는 다른 방법은 분산 시스템에서 메시지를 수신하는 큐 중 하나를 제공하도록 하는 것입니다. 신용 조회 샘플 애플리케이션의 전달되는 양식으로 이러한 큐 중 세 개는 모두 에이전시 조회 프로그램에서 제공합니다. 이 프로그램은 VSAM 데이터 세트를 검색합니다. VSAM 데이터 세트와 IMS 데이터베이스의 사용을 비교하기 위해 IMS 당좌 예금 계좌 프로그램이 이러한 큐 중 하나를 대신 제공하게 할 수 있습니다. 이를 수행하려면, CSQ4SAMP.B n.MESSAGES 큐 중 하나를 CSQ4SAMP.B3.IMS.MESSAGES 큐로 대체하기 위해 이름 목록 CSQ4SAMP.B4.NAMELIST의 정의를 대체해야 합니다. 다음 중 하나를 사용할 수 있습니다.

- IBM MQ for z/OS 조작 및 제어판

- ALTER NAMELIST 명령

그런 다음 CICS 트랜잭션 MVB1에서 샘플을 실행할 수 있습니다. 사용자는 조작 또는 응답에서 어떠한 차이도 느끼지 못합니다. 중지 메시지를 수신한 후 또는 5분 동안 비활성화된 후 IMS BMP가 중지됩니다.

IMS 당좌 예금 계좌 프로그램(CSQ4ICB3)의 디자인

이 프로그램은 BMP로 실행됩니다. IBM MQ 메시지가 송신되기 전에 해당 JCL을 사용하여 프로그램을 시작하십시오.

프로그램은 대출 요청 메시지에서 계좌 번호에 대해 IMS 데이터베이스를 검색합니다. 해당하는 계좌 이름, 평균 잔액 및 신용도 색인을 검색합니다.

프로그램은 데이터베이스 검색의 결과를 처리 중인 IBM MQ 메시지에서 이름 지정된 응답 대상 큐로 송신합니다. 리턴된 메시지는 응답을 빌드하는 트랜잭션이 올바른 조회가 처리 중임을 확인할 수 있도록 계좌 유형 및 검색 결과를 수신된 메시지에 추가합니다. 메시지는 다음과 같이 세 개의 79자 그룹 양식으로 되어 있습니다.

```
'Response from CHECKING ACCOUNT for name : JONES J B'  
'   Opened 870530, 3-month average balance = 000012.57'  
'   Credit worthiness index - BBB'
```

메시지 지향 BMP로 실행 중인 경우 프로그램은 IMS 메시지 큐를 제거한 후 IBM MQ for z/OS 큐에서 메시지를 읽고 이를 처리합니다. IMS 메시지 큐에서 수신되는 정보는 없습니다. 프로그램은 핸들이 닫혔기 때문에 각 체크포인트 후 큐 관리자에 다시 연결합니다.

배치 지향 BMP를 실행 중인 경우 프로그램은 핸들이 닫히지 않았기 때문에 각 체크포인트 후 큐 관리자에 계속 연결되어 있습니다.

▶ z/OS z/OS에서의 메시지 핸들러 샘플

메시지 핸들러 샘플 TSO 애플리케이션을 통해 큐에서 메시지를 찾아보고, 전달 및 삭제할 수 있습니다. 샘플은 C 및 COBOL로 사용 가능합니다.

샘플 준비 및 실행

다음 단계를 수행하십시오.

1. 1089 페이지의 『z/OS에서 TSO 환경에 대한 샘플 애플리케이션 준비』에서 설명된 대로 샘플을 준비하십시오.
2. 패널의 위치, 메시지 파일의 위치 및 로드 모듈의 위치를 정의하려면 샘플에 제공된 CLIST(CSQ4RCH1)를 조정하십시오.

CLIST CSQ4RCH1을 사용하여 샘플의 C 및 COBOL 버전 모두를 실행할 수 있습니다. 제공된 CSQ4RCH1 버전은 C 버전을 실행하고 COBOL 버전에 필요한 조정 관련 지시사항을 포함합니다.

참고:

1. 샘플에 제공된 샘플 큐 정의는 없습니다.
2. VS COBOL II는 ISPF에서 멀티태스킹을 지원하지 않으므로 분할 화면의 양쪽 사이드에서 메시지 핸들러 샘플 애플리케이션을 사용하지 마십시오. 사용하는 경우 결과는 예측할 수 없습니다.

▶ z/OS z/OS에서 메시지 핸들러 샘플 사용

샘플을 설치하고 조정된 CLIST CSQ4RCH1에서 호출하면 1123 페이지의 그림 163에 표시된 화면이 표시됩니다.

```

----- IBM MQ for z/OS -- Samples -----
COMMAND ==>
User Id : JOHNS

Enter information. Press ENTER :

Queue Manager Name : _____ :
Queue Name          : _____ :

F1=HELP  F2=SPLIT  F3=END  F4=RETURN  F5=RFIND  F6=RCHANGE
F7=UP    F8=DOWN  F9=SWAP  F10=LEFT  F11=RIGHT F12=RETRIEVE

```

그림 163. 메시지 핸들러 샘플에 대한 초기 화면

확인할 큐 관리자 큐 이름을 입력하면(대소문자 구분) 메일 목록 화면이 표시됩니다([1123 페이지의 그림 164 참조](#)).

```

----- IBM MQ for z/OS -- Samples ----- Row 1 to 4 of 4
COMMAND ==>

Queue Manager : VM03
Queue         : MQEI.IMS.BRIDGE.QUEUE

Message number 01 of 04

Msg Put Date Put Time Format User Put Application
No MM/DD/YYYY HH:MM:SS Name Identifier Type Name
01 10/16/1998 13:51:19 MQIMS NTSFV02 00000002 NTSFV02A
02 10/16/1998 13:55:45 MQIMS JOHNS 00000011 EDIT\CLASSES\BIN\PROGTS
03 10/16/1998 13:54:01 MQIMS NTSFV02 00000002 NTSFV02B
04 10/16/1998 13:57:22 MQIMS johnj 00000011 EDIT\CLASSES\BIN\PROGTS
***** Bottom of data *****

```

그림 164. 메시지 핸들러 샘플에 대한 메시지 목록 화면

이 화면은 큐에 있는 처음 99개의 메시지를 표시하며 각 메시지에 대해 다음 필드가 표시됩니다.

메시지 번호

메시지 번호

Put 날짜 MM/DD/YYYY

큐에 메시지를 넣은 날짜(GMT)

Put 시간 HH:MM:SS

큐에 메시지를 넣은 시간(GMT)

형식 이름

MQMD.Format 필드

UserIdentifier

MQMD.UserIdentifier 필드

Put 애플리케이션 유형

MQMD.PutApplType 필드

Put 애플리케이션 이름

MQMD.PutApplName 필드

이 화면에서 커서 위치가 아닌 번호 기준으로 메시지를 선택한 후 표시할 수 있습니다. 예를 들어 [1124 페이지의 그림 165](#)의 내용을 참조하십시오.

그림 165. 선택한 메시지가 표시됩니다.

메시지가 MsgId 및 CorrelId를 키로 검색되기 때문에 샘플 디자인을 통해 고유 MsgId / CorrelId가 결합된 메시지만 선택하고 표시할 수 있습니다. 키가 고유하지 않은 경우 샘플은 확신을 가지고 선택한 메시지를 검색할 수 없습니다.

참고: 메시지를 찾아보는 데 SCSQCLST(CSQ4RCH1) 샘플을 사용하는 경우 각 호출로 인해 메시지 백아웃 수가 증가하게 됩니다. 이 샘플의 동작을 변경하려는 경우 샘플을 복사하고 필요에 따라 콘텐츠를 수정하십시오. 이 백아웃 수에 의존하는 다른 애플리케이션이 이 늘어난 수에 의해 영향을 받을 수 있다는 점을 알고 있어야 합니다.

z/OS z/OS에서의 메시지 핸들러 샘플 디자인

이 주제는 메시지 핸들러 샘플 애플리케이션을 구성하는 각 프로그램의 디자인에 대해 설명합니다.

오브젝트 유효성 검증 프로그램

이는 올바른 큐 및 큐 관리자 이름을 요청합니다.

큐 관리자 이름을 지정하지 않는 경우 가능하다면 기본 큐 관리자가 사용됩니다. 로컬 큐만 사용될 수 있습니다. 큐가 로컬이 아닌 경우 큐 유형 및 오류가 보고되었는지 확인하기 위해 MQINQ가 발행됩니다. 큐가 성공적으로 열리지 않거나 큐에서 MQGET 호출이 금지된 경우 완료 코드 및 이유 리턴 코드를 나타내는 오류 메시지가 리턴됩니다.

메시지 목록 프로그램

이는 Put 날짜, Put 시간 및 메시지 형식과 같은 메시지에 대한 정보와 함께 큐에 있는 메시지 목록을 표시합니다.

목록에 저장되는 최대 메시지 수는 99개입니다. 큐에 이 숫자보다 많은 메시지가 있는 경우 현재 큐 용량도 표시됩니다. 표시할 메시지를 선택하려면, 입력 항목 필드에 메시지 번호를 입력하십시오(기본값은 01임). 입력 항목이 올바르지 않은 경우 해당하는 오류 메시지를 수신합니다.

메시지 콘텐츠 프로그램

메시지 콘텐츠를 표시합니다.

콘텐츠는 형식화되어 있으며 다음 두 가지 파트로 나누어집니다.

1. 메시지 디스크립터
2. 메시지 버퍼

메시지 디스크립터는 별도의 행에 각 필드의 콘텐츠를 표시합니다.

메시지 버퍼는 해당 콘텐츠에 따라 형식화됩니다. 버퍼가 데드-레터 헤더(MQDLH) 또는 전송 큐 헤더(MQXQH)를 보유하는 경우 버퍼 자체 이전에 형식화되고 표시됩니다.

버퍼 데이터가 형식화되기 전에 제목 행은 메시지의 버퍼 길이를 바이트 단위로 표시합니다. 최대 버퍼 크기는 32,768바이트이며 이보다 긴 메시지는 잘립니다. 메시지의 처음 32,768바이트만 표시된다는 메시지와 함께 버퍼의 전체 크기가 표시됩니다.

버퍼 데이터는 다음 두 가지 방법으로 형식화됩니다.

1. 버퍼로의 오프셋이 출력된 후, 버퍼 데이터는 16진수로 표시됩니다.
2. 버퍼 데이터가 다시 EBCDIC 값으로 표시됩니다. EBCDIC 값을 출력할 수 없는 경우 마침표(.)로 대신 출력됩니다.

삭제의 경우 D, 전달의 경우 F를 조치 필드에 입력할 수 있습니다. 메시지를 전달하도록 선택하는 경우 forward-to queue 및 queue manager name을 올바르게 설정해야 합니다. 메시지 디스크립터 ReplyToQ 및 ReplyToQMgr 필드에서 이러한 필드에 대한 기본값을 읽습니다.

메시지를 전달하는 경우 버퍼에 저장된 헤더 블록은 스트립됩니다. 메시지가 성공적으로 전달되면 원래 큐에서 제거됩니다. 올바르지 않은 조치를 입력하면 오류 메시지가 표시됩니다.

CSQ4CHP9라는 예제 도움말 패널도 사용 가능합니다.

z/OS에서의 비동기 Put 샘플

비동기 Put 샘플 프로그램은 비동기 MQPUT 호출을 사용하여 큐에 메시지를 넣습니다. 또한 샘플은 MQSTAT 호출을 사용하여 상태 정보를 검색합니다.

비동기 Put 애플리케이션은 다음 MQI 호출을 사용합니다.

- MQCONN
- MQOPEN
- MQPUT
- MQSTAT
- MQCLOSE
- MQDISC

샘플 프로그램이 C 프로그래밍 언어로 제공됩니다.

비동기 Put 애플리케이션이 배치 환경에서 실행됩니다. 배치 애플리케이션은 [기타 샘플의 내용](#)을 참조하십시오.
이 주제는 비동기 이용 프로그램의 디자인에 대한 정보도 제공하며 CSQ4BCS2 샘플을 실행합니다.

- [1126 페이지의 『CSQ4BCS2 샘플 실행』](#)
- [1126 페이지의 『비동기 Put 샘플 프로그램의 디자인』](#)

CSQ4BCS2 샘플 실행

이 샘플 프로그램은 최대 6개 매개변수를 사용합니다.

1. 대상 큐의 이름(필수)
2. 큐 관리자의 이름(선택사항)
3. 열기 옵션(선택사항)
4. 닫기 옵션(선택사항)
5. 대상 큐 관리자의 이름(선택사항)
6. 동적 큐의 이름(선택사항)

큐 관리자가 지정되지 않은 경우 CSQ4BCS2는 기본 큐 관리자에 연결합니다. 메시지 콘텐츠가 표준 입력(**SYSDD**)을 통해 제공됩니다.

프로그램을 실행하기 위한 샘플 JCL이 있으며 CSQ4BCSP에 상주합니다.

비동기 Put 샘플 프로그램의 디자인

프로그램은 메시지를 넣기 위한 대상 큐를 열기 위해 제공된 출력 옵션 또는 MQOO_OUTPUT 및 MQOO_FAIL_IF_QUIESCING과 함께 MQOPEN 호출을 사용합니다.

프로그램이 큐를 열 수 없는 경우 프로그램은 MQOPEN 호출에서 리턴된 이유 코드를 포함하는 오류 메시지를 출력합니다. 이 호출 및 후속 MQI 호출에서 프로그램을 단순하게 유지하기 위해 여러 옵션에 대해 기본값이 사용됩니다.

입력의 각 행에 대해 프로그램을 텍스트를 버퍼로 읽고 해당 행의 텍스트를 포함하는 데이터그램 메시지를 작성하고 대상 큐에 비동기적으로 메시지를 넣기 위해 MQPMO_ASYNC_RESPONSE와 함께 MQPUT 호출을 사용합니다. 프로그램은 입력의 끝에 도달할 때까지 또는 MQPUT 호출에 실패할 때까지 계속됩니다. 프로그램이 입력의 끝에 도달하는 경우 MQCLOSE 호출을 사용하여 큐를 닫습니다.

그런 다음 프로그램은 MQSTS 구조를 리턴하는 MQSTAT 호출을 발행하고, 성공적으로 넣어진 메시지 수, 경고와 함께 넣어진 메시지 수 및 실패 수를 포함하는 메시지를 표시합니다.

참고: MQSTAT 호출에서 MQPUT 오류를 감지하는 경우 발생하는 상황을 관찰하려면, 대상 큐의 MAXDEPTH를 낮은 값으로 설정하십시오.

z/OS의 배치 비동기 이용 샘플

CSQ4BCS1 샘플 프로그램이 C로 제공되며 여러 큐에서 비동기적으로 메시지를 이용하기 위해 MQCB 및 MQCTL의 사용을 보여줍니다.

비동기 이용 샘플이 배치 환경에서 실행됩니다. 배치 애플리케이션은 [기타 샘플의 내용](#)을 참조하십시오.

CICS 환경에서 실행되는 COBOL 샘플도 있습니다. [1128 페이지의 『z/OS의 CICS Asynchronous Consumption and Publish/Subscribe 샘플』](#)의 내용을 참조하십시오.

애플리케이션은 이러한 MQI 호출을 사용합니다.

- MQCONN
- MQOPEN
- MQCLOSE
- MQDISC

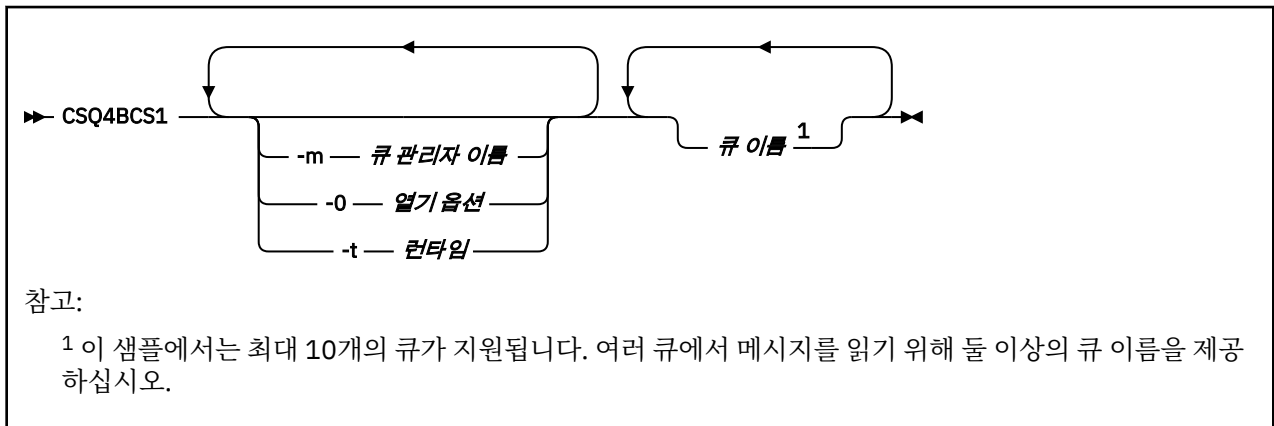
- MQCB
- MQCTL

이 주제는 다음 표제에 대한 정보도 제공합니다.

- [1127 페이지의 『CSQ4BCS1 샘플 실행』](#)
- [1127 페이지의 『배치 비동기 이용 샘플 프로그램의 디자인』](#)

CSQ4BCS1 샘플 실행

이 샘플 프로그램은 다음 구문을 따릅니다.



이 프로그램을 실행하기 위한 샘플 JCL이 있으며 CSQ4BCSC에 상주합니다.

배치 비동기 이용 샘플 프로그램의 디자인

샘플은 도착하는 순서대로 여러 큐에서 메시지를 읽는 방법을 표시합니다. 동기 MQGET을 사용하여 더 많은 코드가 필요하게 됩니다. 비동기 이용을 사용하면, 폴링은 필요하지 않으며 IBM MQ에서 스레드 및 스토리지 관리를 수행합니다. 샘플 프로그램에서 오류는 콘솔에 작성됩니다.

샘플 코드에는 다음 단계가 있습니다.

1. 단일 메시지 이용 콜백 함수를 정의하고,

```
void MessageConsumer(MQHCONN hConn,
MQMD * pMsgDesc,
MQGMO * pGetMsgOpts,
MQBYTE * Buffer,
MQCBC * pContext)
{ ... }
```

2. 큐 관리자에 연결하십시오.

```
MQCONN(QMName,&Hcon,&CompCode,&CReason);
```

3. 입력 큐를 열고 각 큐를 MessageConsumer 콜백 함수와 연관시키십시오.

```
MQOPEN(Hcon,&od,0_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

cbd.CallbackFunction은 각 큐에 대해 설정할 필요가 없습니다. 입력 전용 필드입니다. 다른 콜백 함수를 각 큐와 연관시킬 수 있습니다.

4. 메시지 이용을 시작하십시오.

```
MQCTL(Hcon,MQOP_START,&cto,&CompCode,&Reason);
```

5. 사용자가 Enter를 누를 때까지 기다린 후 메시지 이용을 중지하십시오.

```
MQCTL(Hcon,MQOP_STOP,&ctl0,&CompCode,&Reason);
```

6. 마지막으로 큐 관리자에서 연결을 끊으십시오.

```
MQDISC(&Hcon,&CompCode,&Reason);
```

z/OS의 CICS Asynchronous Consumption and Publish/Subscribe 샘플

비동기 이용 및 발행/구독 샘플 프로그램은 비동기 이용의 사용 및 CICS 내의 발행 및 구독 기능을 보여줍니다.

등록 클라이언트 프로그램은 세 개의 콜백 핸들러(이벤트 핸들러 및 두 개의 메시지 이용자)를 등록하고 비동기 이용을 시작합니다. 메세징 클라이언트 프로그램은 메시지를 큐에 넣거나, 두 개의 메시지 이용자(CSQ4CVCN 및 CSQ4CVCT)에 의한 이용을 위해 CICS 콘솔에서 적당한 메시지를 발행합니다.

샘플 동작에 대해 런타임 제어를 제공하려면 수신하는 메시지를 사용하여 콜백 핸들러를 SUSPEND, RESUME 또는 DEREGISTER하기 위해 메시지 이용자 중 하나를 지시할 수 있습니다. 또한 제어 하에 비동기 이용을 종료하기 위해 MQCTL STOP을 발행하는 데 사용할 수 있습니다. 주제에 구독하기 위해 다른 메시지 이용자가 등록됩니다.

각 프로그램은 샘플의 동작을 표시하기 위해 적절한 지점에서 COBOL DISPLAY 명령문을 발행합니다.

애플리케이션은 이러한 MQI 호출을 사용합니다.

- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQCB
- MQCTL

프로그램은 COBOL 언어로 제공됩니다. CICS 애플리케이션에 대한 [CICS 비동기 이용 및 발행/구독 샘플](#) 을 참조하십시오.

또한 이 토픽에서는 다음 정보도 제공합니다.

- [1128 페이지의 『설정』](#)
- [1128 페이지의 『등록 클라이언트 CSQ4CVRG』](#)
- [1129 페이지의 『이벤트 핸들러 CSQ4CVEV』](#)
- [1129 페이지의 『단순한 메시지 이용자 CSQ4CVCN』](#)
- [1129 페이지의 『제어 메시지 이용자 CSQ4CVCT』](#)
- [1129 페이지의 『메시징 클라이언트 CSQ4CVPT』](#)

설정

메시지 이용자가 사용한 큐 및 주제의 이름은 등록 및 메세징 클라이언트 프로그램에서 하드코딩됩니다.

큐, **SAMPLE.CONTROL.QUEUE**를 샘플을 실행하기 전에 CICS 영역과 연관된 큐 관리자에 정의해야 합니다. 필요한 경우 뉴스/미디어/영화 주제를 정의할 수 있으며 주제가 없는 경우 기본 관리 오브젝트 아래에서 런타임 시작됩니다.

CSQ4SAMP 그룹 설치에 의해 CICS 프로그램 및 트랜잭션 정의를 설치할 수 있습니다.

등록 클라이언트 CSQ4CVRG

CICS 트랜잭션 MVRG 아래에서 등록 클라이언트 프로그램을 시작해야 합니다. 입력을 사용하지 않습니다.

시작되면 등록 클라이언트는 MQCB를 사용하여 다음 콜백 핸들러를 등록합니다.

- 이벤트 핸들러로 CSQ4CVEV
- 뉴스/미디어/영화 주제에 대한 메시지 이용자로 CSQ4CVCN
- **SAMPLE.CONTROL.QUEUE** 큐의 메시지 이용자로 CSQ4CVCT.

등록 클라이언트는 CSQ4CVCT에 등록된 세 가지 모든 콜백 핸들러의 이름과 두 개 메시지 이용자와 연관된 오브젝트 핸들을 포함하는 데이터 구조를 전달합니다.

콜백 핸들러를 등록하면 등록 클라이언트에서 MQCTL START_WAIT를 발행하여 비동기 이용을 시작하고 제어가 리턴될 때까지 일시중단합니다(예를 들어, MQCTL STOP을 발행하는 콜백 핸들러 중 하나에 의해).

이벤트 핸들러 CSQ4CVEV

구동되면 이벤트 핸들러가 호출 유형을 나타내는 메시지를 표시합니다(예: START). IBM MQ 이유 코드 CONNECTION_QUIESCING에 대해 구동되면 이벤트 핸들러가 MQCTL STOP을 발행하여 비동기 이용을 종료하고 등록 클라이언트에 제어를 리턴합니다.

단순한 메시지 이용자 CSQ4CVCN

구동되면 이 메시지 이용자는 호출 유형을 나타내는 메시지를 표시합니다(예: REGISTER). MSG_REMOVED 호출 유형에 대해 구동되면, 메시지 이용자는 인바운드 메시지를 검색하고 CICS 작업 로그에 출력합니다.

제어 메시지 이용자 CSQ4CVCT

구동되면 이 메시지 이용자는 호출 유형을 나타내는 메시지를 표시합니다(예: START). MSG_REMOVED 호출 유형에 대해 구동되면 메시지 이용자가 인바운드 메시지 및 등록 클라이언트가 전달한 데이터 구조를 검색합니다. 메시지 콘텐츠를 기반으로 다음 중 하나에 적절한 MQCB 또는 MQCTL 명령을 발행합니다.

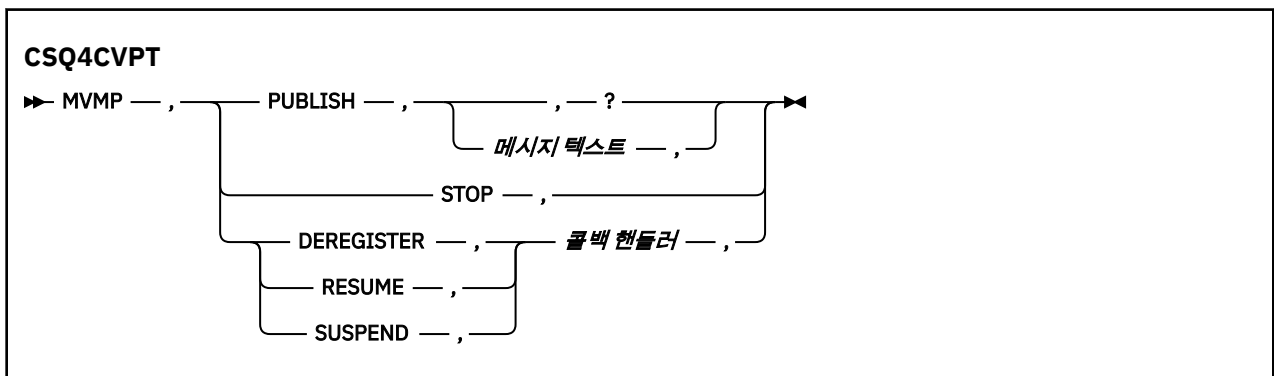
- 비동기 이용 STOP(등록 클라이언트에 제어를 리턴).
- 이름 지정된 콜백 핸들러 SUSPEND, RESUME 또는 Deregister(자체 포함).

메시징 클라이언트 CSQ4CVPT

메시징 클라이언트는 두 가지 기능을 가지고 있습니다.

- 메시지 이용자 CSQ4CVCN에서 이용하도록 주제에 메시지를 발행합니다.
- 제어 메시지 이용자 CSQ4CVCT에서 이용하도록 큐에 제어 메시지를 넣고 이에 따라 샘플의 동작에 잠재적인 변화가 발생합니다.

메시징 클라이언트 프로그램은 CICS 트랜잭션 아래의 CICS 콘솔에서 시작해야 하며 다음 구문으로 명령행 입력을 사용합니다.



발행

단순 메시지 이용자에서 이용하도록 보유된 메시지로 메시지 텍스트(또는 기본 메시지)를 발행합니다.

STOP

비동기 이용을 중지합니다.

DEREGISTER

이름 지정된 콜백 핸들러를 등록 취소합니다.

RESUME

이름 지정된 콜백 핸들러를 재개합니다.

SUSPEND

이름 지정된 콜백 핸들러를 일시중단합니다.

입력 필드는 위치와 관련되며 쉼표로 구분됩니다. 키워드 및 콜백 핸들러 이름은 대소문자를 구분하지 않습니다.

예:

표 193. 입력 예	
예	설명
MVMP,PUBLISH,,	기본 메시지 발행
MVMP,게시, A short message,	지정된 텍스트 발행
MVMP,STOP,	비동기 이용 중지
MVMP,DEREGISTER,CSQ4CVEV,	이벤트 핸들러 등록 취소
MVMP,resume,csq4cvcn,	단순 메시지 이용자 재개
MVMP,SUSPEND,CSQ4CVEV,	이벤트 핸들러 일시중단

여기서 MVMP는 메시징 클라이언트 프로그램 CSQ4CVPT와 연관된 CICS 트랜잭션입니다.

참고:

- 모든 콜백 핸들러를 일시중단 또는 등록 취소하면 등록 클라이언트에서 발행한 START_WAIT이 종료되고, 제어가 리턴되며 태스크가 종료됩니다.
- 제어 콜백 핸들러의 일시중단 또는 등록 취소는 의도적으로 방지되지만, 샘플의 동작을 추가 제어하기 위한 기능이 제거됩니다.

▶ z/OS z/OS에서의 발행/구독 샘플

발행/구독 샘플 프로그램은 IBM MQ의 발행 및 구독 기능의 사용을 보여줍니다.

IBM MQ 발행/구독 인터페이스에 프로그래밍하는 방법을 보여주는 네 개의 C 및 두 개의 COBOL 프로그래밍 언어 샘플 프로그램이 있습니다. 프로그램은 C 및 COBOL 언어로 제공됩니다. 애플리케이션은 배치 환경에서 실행됩니다. 배치 애플리케이션은 발행/구독 샘플을 참조하십시오.

CICS 환경에서 실행되는 COBOL 샘플도 있습니다. 1128 페이지의 『z/OS의 CICS Asynchronous Consumption and Publish/Subscribe 샘플』의 내용을 참조하십시오.

이 주제는 또한 발행/구독 샘플 프로그램을 실행하는 방법에 대한 정보를 제공합니다. 이러한 샘플 프로그램은 다음을 포함합니다.

- 1130 페이지의 『CSQ4BCP1 샘플 실행』
- 1131 페이지의 『CSQ4BCP2 샘플 실행』
- 1131 페이지의 『CSQ4BCP3 샘플 실행』
- 1131 페이지의 『CSQ4BCP4 샘플 실행』
- 1132 페이지의 『CSQ4BVP1 샘플 실행』
- 1132 페이지의 『CSQ4BVP2 샘플 실행』

CSQ4BCP1 샘플 실행

이 프로그램은 C로 작성되며, 토픽으로 메시지를 발행합니다. 이 프로그램을 실행하기 전에 구독자 샘플 중 하나를 시작하십시오.

이 프로그램은 최대 4개의 매개변수를 사용합니다.

1. 대상 토픽 문자열의 이름(필수)
2. 큐 관리자의 이름(선택사항)
3. 열기 옵션(선택사항)
4. 닫기 옵션(선택사항)

큐 관리자가 지정되지 않은 경우 CSQ4BCP1는 기본 큐 관리자에 연결합니다. 프로그램을 실행하기 위한 샘플 JCL이 있으며 CSQ4BCPP에 상주합니다.

메시지 콘텐츠가 표준 입력(SYSIN DD)을 통해 제공됩니다.

CSQ4BCP2 샘플 실행

이 프로그램은 C로 작성되며, 토픽을 구독하고 수신된 메시지를 인쇄합니다.

이 프로그램은 최대 3개의 매개변수를 사용합니다.

1. 대상 토픽 문자열의 이름(필수)
2. 큐 관리자의 이름(선택사항)
3. MQSD 구독 옵션(선택사항)

큐 관리자가 지정되지 않은 경우 CSQ4BCP2는 기본 큐 관리자에 연결합니다. 프로그램을 실행하기 위한 샘플 JCL이 있으며 CSQ4BCPS에 상주합니다.

CSQ4BCP3 샘플 실행

이 프로그램은 C로 작성되며, 사용자가 지정한 목적지 큐를 사용하여 토픽을 구독하고 수신된 메시지를 인쇄합니다.

이 프로그램은 최대 4개의 매개변수를 사용합니다.

1. 대상 토픽 문자열의 이름(필수)
2. 대상의 이름(필수)
3. 큐 관리자의 이름(선택사항)
4. MQSD 구독 옵션(선택사항)

큐 관리자가 지정되지 않은 경우 CSQ4BCP3은 기본 큐 관리자에 연결합니다. 프로그램을 실행하기 위한 샘플 JCL이 있으며 CSQ4BCPD에 상주합니다.

CSQ4BCP4 샘플 실행

이 프로그램은 C로 작성되며, MQSUB 호출에서 확장된 옵션의 사용을 허용하는 주제를 구독하고 메시지를 가져 오며, 더 간단한 MQSUN 샘플: CSQ4BCP2에서 사용 가능한 옵션으로 확장합니다. 메시지 페이로드(payload)뿐만 아니라 각 메시지의 메시지 특성이 수신되고 표시됩니다.

이 프로그램은 매개변수의 변수 세트를 사용합니다.

- **-t** *Topic string*.
 - **-o** *Topic object name*.
- 중요사항:** **-t** 또는 **-o** 중 하나 또는 둘 다 필요합니다.
- **-m** *Queue manager name* (선택사항).
 - **-b** *Connection binding type* (선택적), 여기서 *type* 은 다음 값 중 하나를 가질 수 있습니다.
 - *standard* : MQCNO_STANDARD_BINDING (기본값)
 - 공유: MQCNO_SHARED_BINDING
 - 빠른 경로: MQCNO_FASTPATH_BINDING
 - 분리됨: MQCNO_ISOLATED_BINDING
 - **-q** *Destination queue name* (선택사항).

- **-w** *Wait interval on MQGET in seconds* (선택사항). 여기서 *seconds* 의 값은 다음 중 하나입니다.
 - unlimited: MQWI_UNLIMITED
 - none: 대기하지 않음
 - n: 대기 간격(초)
 - 지정된 값 없음: 값이 지정되지 않은 경우 기본값은 30초입니다.
- **-d** *Subscription name* (선택사항). 이름이 지정된 지속 가능한 구독을 작성하거나 계속합니다.
- **-k** (선택사항). MQCLOSE에서 지속 가능한 구독을 유지합니다.

큐 관리자가 지정되지 않은 경우 CSQ4BCP4는 기본 큐 관리자에 연결합니다. 프로그램을 실행하기 위한 샘플 JCL이 있으며 CSQ4BCPE에 상주합니다.

CSQ4BVP1 샘플 실행

이 프로그램은 COBOL로 작성되며 주제로 메시지를 발행합니다. 이 프로그램을 실행하기 전에 구독자 샘플 중 하나를 시작하십시오.

이 프로그램은 매개변수를 사용하지 않습니다. **SYSIN DD**는 입력 주제 이름, 큐 관리자 이름 및 메시지 콘텐츠를 제공합니다.

큐 관리자가 지정되지 않은 경우 CSQ4BVP1은 기본 큐 관리자에 연결합니다. 프로그램을 실행하기 위한 샘플 JCL이 있으며 CSQ4BVPP에 상주합니다.

CSQ4BVP2 샘플 실행

이 프로그램은 COBOL로 작성되며 주제를 구독하고 수신된 메시지를 출력합니다.

이 프로그램은 매개변수를 사용하지 않습니다. **SYSIN DD**는 주제 이름 및 큐 관리자 이름에 대한 입력을 제공합니다.

큐 관리자가 지정되지 않은 경우 CSQ4BVP1은 기본 큐 관리자에 연결합니다. 프로그램을 실행하기 위한 샘플 JCL이 있으며 CSQ4BVPP에 상주합니다.

z/OS에서 메시지 특성 샘플의 설정 및 조회

메시지 특성 샘플 프로그램은 메시지 핸들에 사용자 정의 특성의 추가 및 해당 메시지와 연관된 특성의 조사를 보여줍니다.

애플리케이션은 이러한 MQI 호출을 사용합니다.

- MQCONN
- MQOPEN
- MQPUT
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP
- MQSETMP

프로그램은 C 언어로 제공됩니다. 애플리케이션은 배치 환경에서 실행됩니다. 배치 애플리케이션은 [기타 샘플](#)의 내용을 참조하십시오.

CSQ4BCM1 프로그램은 메시지 큐에서 메시지 핸들의 특성을 조회하는 데 사용되며 MQINQMP API 호출의 사용 예입니다. 샘플은 큐에서 하나의 메시지를 가져온 후 모든 메시지 핸들 특성을 출력합니다.

CSQ4BCM2 프로그램은 메시지 큐에 메시지 핸들의 특성을 설정하는 데 사용되며 MQSETMP API 호출의 사용 예입니다. 샘플은 메시지 핸들을 작성하여 MQGMO 구조의 MsgHandle 필드에 넣습니다. 그런 다음 메시지를 큐에 넣습니다.

메시지 특성 조회 및 출력의 기타 예는 CSQ4BCG1 및 CSQ4BCP4 샘플 프로그램에 포함됩니다.

이 주제는 또한 다음 표제 아래의 설정 및 조회 메시지 특성 샘플 실행에 대한 정보를 제공합니다.

- [1133 페이지의 『CSQ4BCM1 샘플 실행』](#)
- [1133 페이지의 『CSQ4BCM2 샘플 실행』](#)

CSQ4BCM1 샘플 실행

이 프로그램은 최대 4개의 매개변수를 사용합니다.

1. 대상 큐의 이름(필수)
2. 큐 관리자의 이름(선택사항)
3. 열기 옵션(선택사항)
4. 닫기 옵션(선택사항)

CSQ4BCM2 샘플 실행

이 프로그램은 최대 6개 매개변수를 사용합니다.

1. 대상 큐의 이름(필수)
2. 큐 관리자의 이름(선택사항)
3. 열기 옵션(선택사항)
4. 닫기 옵션(선택사항)
5. 대상 큐 관리자의 이름(선택사항)
6. 동적 큐의 이름(선택사항)

특성 이름, 값 및 메시지 콘텐츠는 표준 입력 (**SYSIN DD**) 을 통해 제공됩니다. 프로그램을 실행하기 위한 샘플 JCL이 있으며 CSQ4BCMP에 상주합니다.

절차에 따른 프로그램 오류 핸들링

이 정보는 호출할 때 또는 메시지를 최종 목적지에 전달할 때 애플리케이션 MQI 호출과 관련된 오류를 설명합니다.

가능할 때마다 MQI를 호출하는 즉시 큐 관리자에서 오류를 리턴합니다. 이러한 오류는 로컬에서 판별한 오류입니다.

리모트 큐에 메시지를 보낼 때 MQI를 호출하면 오류가 표시되지 않을 수도 있습니다. 이 경우 오류를 식별하는 큐 관리자가 원본 프로그램에 다른 메시지를 보내 오류를 보고합니다. 이러한 오류는 리모트로 판별한 오류입니다.

로컬에서 판별한 오류

MQI 호출 실패, 시스템 인터럽트, 잘못된 데이터가 포함된 메시지 등을 포함하는 로컬에서 감지된 오류에 대한 정보.

큐 관리자가 즉시 보고할 수 있는 오류의 가장 일반적인 원인은 다음 세 가지입니다.

- MQI 호출 실패(예: 큐가 가득차기 때문).
- 애플리케이션이 종속된 시스템의 일부를 실행할 때 인터럽트 발생(예: 큐 관리자)
- 성공적으로 처리할 수 없는 데이터를 포함하는 메시지

비동기 Put 기능을 사용하는 경우 오류가 즉시 보고되지 않습니다. MQSTAT 호출을 사용하여 이전 비동기 Put 조작에 관한 상태 정보를 검색하십시오.

MQI 호출 실패

큐 관리자는 `MQI` 호출의 코딩에 오류가 있을 경우 즉시 이를 보고할 수 있습니다. 이 작업은 사전정의된 리턴 코드 세트를 사용하여 수행합니다. 리턴 코드 세트는 완료 코드와 이유 코드로 구분됩니다.

호출이 성공했는지 여부를 표시하기 위해, 큐 관리자는 호출이 완료될 때 완료 코드를 반환합니다. 성공, 부분 완료 및 호출 실패를 표시하는 세 개의 완료 코드가 있습니다. 큐 관리자는 또한 호출의 부분적 완료 또는 실패 사유를 나타내는 사유 코드를 반환합니다.

각 호출의 완료 코드와 이유 코드가 리턴 코드에 있는 해당 호출의 설명과 함께 나열됩니다. 더 자세한 정보(시정 조치 방안 포함)는 을 참조하십시오 메시지 및 오류 코드.

각 호출에서 생성될 수 있는 모든 리턴 코드를 핸들링하도록 프로그램을 디자인하십시오.

시스템 중단

연결된 큐 관리자를 시스템 실패에서 복구해야 하는 경우 애플리케이션에서 인터럽트를 인식하지 못할 수도 있습니다. 그러나 이러한 인터럽트가 발생하는 경우 데이터가 유실되지 않도록 애플리케이션을 디자인해야 합니다.

데이터가 일관되도록 유지하는 데 사용할 수 있는 메소드는 큐 관리자가 실행 중인 플랫폼에 따라 달라집니다.

ALW **AIX, Linux, and Windows 시스템**

이러한 환경에서 일반적인 방식으로 MQPUT과 MQGET을 호출할 수 있지만, MQCMIT 및 MQBACK 호출을 사용하여 동기점을 선언해야 합니다(863 페이지의 『작업 단위 커밋 및 백아웃』 참조). CICS 환경에서는 CICS에서 관리하는 작업 단위에서 MQPUT과 MQGET을 호출할 수 있으므로 MQCMIT와 MQBACK 명령을 사용하지 않게 설정할 수 있습니다.

IBM i **IBM i**

IBM i 커밋 제어를 통해 관리하는 글로벌 작업 단위에서 MQPUT 및 MQGET을 호출할 수 있습니다. 고유 IBM i COMMIT 및 ROLLBACK 명령 또는 언어 특정 명령을 사용하여 동기점을 선언할 수 있습니다. 로컬 작업 단위는 MQCMIT 및 MQBACK 호출을 사용하여 IBM MQ가 관리합니다.

z/OS **z/OS**

CICS 및 IMS 환경에서 CICS 또는 IMS가 관리하는 작업 단위에서 MQPUT 및 MQGET을 호출할 수 있습니다. 배치 환경에서 동일한 방식으로 MQPUT과 MQGET을 호출할 수 있지만 다음을 사용하여 동기점을 선언해야 합니다.

- IBM MQ for z/OS MQCMIT 및 MQBACK 호출(863 페이지의 『작업 단위 커밋 및 백아웃』 참조) 또는
- 2단계 동기점 지원을 제공하는 z/OS 트랜잭션 관리 및 복구 가능한 자원 관리자 서비스(RRS). RRS를 사용하면 작업의 단일 논리 단위에서 IBM MQ 및 기타 RRS 사용 제품 자원(예: Db2 스토어드 프로시저 자원)을 모두 업데이트할 수 있습니다. RRS 동기점 지원에 대한 정보는 867 페이지의 『트랜잭션 관리 및 복구 가능한 자원 관리자 서비스』의 내용을 참조하십시오.

손실되지 않아야 하는 모든 데이터를 이동시키는 데 지속 메시지를 사용하십시오. 큐 관리자를 실패에서 복원구해야 하는 경우 지속 메시지는 큐에 복구됩니다.

- **ALW** 이 AIX, Linux, and Windows IBM MQ 옵션이 활성화된 상태에서 애플리케이션 내 MQGET 또는 MQPUT 호출은 모든 로그 파일을 채우는 시점에서 MQRC_RESOURCE_PROBLEM 메시지와 함께 실패합니다. 로그 파일에 대한 자세한 내용은 AIX, Linux, and Windows 로그 기록: 메시지 손실 방지 방법을 참조하십시오.
- **z/OS** 참조하십시오 로그 환경 계획 수립 z/OSz/OS.

애플리케이션이 실행 중인 동안 운영자가 큐 관리자를 중지하는 경우 일반적으로 일시정지 옵션을 사용합니다. 큐 관리자가 정지 중 상태가 되면, 애플리케이션이 계속 작동할 수 있지만 가능한 빨리 종료해야 합니다. 단시간에 실행되는 작은 애플리케이션에서는 정지 중 상태를 무시하고 정상적으로 종료할 때까지 계속될 수 있습니다. 오래 실행되는 애플리케이션이나 메시지가 도착할 때까지 기다리는 애플리케이션은 MQOPEN, MQPUT, MQPUT1 및 MQGET을 호출할 때 일시정지하는 경우 실패 옵션을 사용해야 합니다. 이러한 옵션은 큐 관리자가 일시정지되면 호출에 실패함을 나타내지만 애플리케이션에서 정지 중 상태를 무시하는 호출을 실행하면 여전히 시간을 갖고 문제없이 종료될 수 있습니다. 이러한 애플리케이션은 변경한 내용을 커밋하거나 백아웃한 다음 종료할 수 있습니다.

큐 관리자가 강제로 중지되면(즉, 일시정지하지 않고 중지) 애플리케이션이 MQI를 호출할 때 MQRC_CONNECTION_BROKEN 이유 코드를 수신합니다. 애플리케이션을 종료하거나 IBM MQ for IBM i, AIX, Linux, and Windows 시스템에서 MQDISC 호출을 실행하십시오.

잘못된 데이터를 포함하는 메시지

애플리케이션에서 작업 단위를 사용할 때 프로그램이 큐에서 검색한 메시지를 처리할 수 없으면 MQGET 호출이 백아웃됩니다.

큐 관리자가 메시지 디스크립터의 *BackoutCount* 필드에 백아웃이 발생한 횟수를 유지합니다. 영향받는 각 메시지의 디스크립터에서 이 수를 유지합니다. 이 수는 애플리케이션 효율성에 대한 중요한 정보를 제공할 수 있습니다. 백아웃 수가 시간 경과에 따라 증가하는 메시지는 반복적으로 거부됩니다. 그 이유를 분석하고 해당 메시지를 적절하게 핸들링하도록 애플리케이션을 디자인하십시오.

z/OS IBM MQ for z/OS에서 큐 관리자를 재시작할 때 백아웃 수가 유지되게 하려면 **HardenGetBackout** 속성을 MQQA_BACKOUT_HARDENED로 설정하십시오. 그렇지 않으면 큐 관리자를 재시작해야 하는 경우 각 메시지의 정확한 백아웃 수를 유지하지 않습니다. 이런 방법으로 속성을 설정하면 추가 처리의 부담이 추가됩니다.

시스템에서 IBM MQ **IBM i** IBM i, AIX, Linux, and Windows 백아웃 카운트는 큐 매니저 재시작 후에도 항상 유지됩니다.

z/OS 또한 IBM MQ for z/OS에서는 작업 단위의 큐에서 메시지를 제거할 때, 애플리케이션이 작업 단위를 백아웃하는 경우 다시 사용 가능하게 되지 않도록 한 메시지를 표시할 수 있습니다. 표시된 메시지는 새 작업 단위에서 검색된 것처럼 처리됩니다. MQGMO_MARK_SKIP_BACKOUT 옵션을 사용하여 백아웃을 건너뛴 메시지를 표시합니다.(MQGMO 구조) MQGET 호출을 사용하는 경우에 해당합니다. 이 기술에 대한 자세한 정보는 813 페이지의 『백아웃 건너뛰기』의 내용을 참조하십시오.

문제점 판별을 위해 보고 메시지 사용

리모트 큐 관리자가 MQI 호출 시 큐에 메시지를 넣지 못하는 등의 오류를 보고할 수 없지만 메시지가 처리된 방식을 알리는 보고 메시지는 사용자에게 보낼 수 있습니다.

애플리케이션에서 (MQPUT) 보고 메시지를 작성할 수 있을 뿐 아니라 해당 메시지를 수신할 옵션도 선택할 수 있습니다(이 경우 다른 애플리케이션이나 큐 관리자가 송신함).

보고 메시지 작성

보고 메시지를 사용하면 애플리케이션이 송신된 메시지를 처리할 수 없음을 다른 애플리케이션에 알릴 수 있습니다.

그러나 메시지를 송신한 애플리케이션이 문제점 정보를 받으려는지 판별하기 위해 처음에 *Report* 필드를 분석해야 합니다. 보고 메시지가 필요하다고 판별하면 다음을 결정해야 합니다.

- 포함할 데이터(전체 원본 메시지 포함, 처음 100바이트의 데이터만 포함 또는 원본 메시지를 전혀 포함하지 않음).
- 원본 메시지에서 수행할 작업. 제거하거나 데드-레터 큐로 이동하게 할 수 있습니다.
- *MsgId* 및 *CorrelId* 필드의 콘텐츠도 필요한지 여부.

Feedback 필드를 사용하여 보고 메시지를 생성하는 이유를 표시하십시오. 애플리케이션의 응답 대상 큐에 보고 메시지를 넣으십시오. 자세한 정보는 [피드백](#)을 참조하십시오.

(MQGET) 보고 메시지 요청 및 수신

필요한 피드백을 표시하도록 *Report* 필드를 완료하지 않으면, 다른 애플리케이션에 메시지를 송신할 때 문제점이 보고되지 않습니다. 사용 가능한 옵션은 [보고서 필드의 구조](#)를 참조하십시오.

큐 관리자가 항상 애플리케이션의 응답 대상 큐에 보고 메시지를 넣습니다. 고유 애플리케이션에서도 동일한 작업을 수행하는 것이 좋습니다. 보고 메시지 기능을 사용할 때 메시지의 메시지 디스크립터에 응답 대상 큐의 이름을 지정하십시오. 그렇지 않으면 MQPUT 호출에 실패합니다.

애플리케이션에 응답 대상 큐를 모니터하고 도착하는 모든 메시지를 처리하는 프로시저가 있어야 합니다. 보고 메시지에 모든 원본 메시지 또는 원본 메시지의 처음 100바이트가 포함되거나 원본 메시지가 전혀 포함되지 않을 수 있습니다.

큐 관리자가 오류의 원인(예: 대상 큐가 없음)을 표시하기 위해 보고 메시지의 *Feedback* 필드를 설정합니다. 프로그램에서도 동일하게 설정해야 합니다.

보고 메시지에 대한 자세한 정보는 [20 페이지의 『보고 메시지』](#)의 내용을 참조하십시오.

리모트로 판별된 오류

리모트 큐에 메시지를 송신할 때 로컬 큐 관리자에서 오류를 발견하지 않고 MQI를 처리한 경우에도 다른 요인이 리모트 큐 관리자가 메시지를 핸들링하는 방식에 영향을 미칠 수 있습니다.

예를 들어 대상 큐가 가득 찼거나 없을 수도 있습니다. 대상 큐까지의 라우트에 있는 다른 중간 큐 관리자가 메시지를 핸들링해야 하는 경우 오류를 발견할 수 있습니다.

메시지 전달 문제

MQPUT 호출에 실패하면 큐에 다시 메시지를 넣거나 송신자에게 리턴하거나 데드-레터 큐에 넣을 수 있습니다.

각 옵션에는 장점이 있지만 MQPUT에 실패한 이유가 목적지 큐가 가득 찼기 때문인 경우 메시지를 다시 넣고 싶지 않을 수 있습니다. 이 경우 데드-레터 큐에 넣으면 나중에 올바른 목적지 큐에 전달될 수 있습니다.

메시지 전달 재시도

데드-레터 큐에 메시지를 넣기 전에 채널에 *MsgRetryCount* 및 *MsgRetryInterval* 속성이 설정되었거나 채널에서 사용할 재시도 엑시트 프로그램(채널 속성 *MsgRetryExitId* 필드에 보유된 이름)이 있으면 리모트 큐 관리자가 큐에 다시 메시지 넣기를 시도합니다.

MsgRetryExitId 필드가 비어 있으면 *MsgRetryCount* 및 *MsgRetryInterval* 속성의 값이 사용됩니다.

MsgRetryExitId 필드가 비어 있지 않으면 이 이름의 엑시트 프로그램이 실행됩니다. 고유 엑시트 프로그램을 사용하는 데 관한 자세한 정보는 [710 페이지의 『메시지 채널에 대한 채널 엑시트 프로그램』](#)의 내용을 참조하십시오.

송신자에게 메시지 리턴

생성될 보고 메시지에 원래 메시지를 모두 포함하도록 요청하여 송신자에게 메시지를 리턴합니다.

보고 메시지 옵션에 대한 자세한 내용은 [20 페이지의 『보고 메시지』](#)의 내용을 참조하십시오.

데드 레터(미전달 메시지) 큐 사용

큐 관리자가 메시지를 전달할 수 없는 경우 큐 관리자는 이 메시지를 데드-레터 큐에 넣기 시도합니다. 이 큐는 큐 관리자가 설치될 때 정의되어야 합니다.

프로그램은 큐 관리자가 사용하는 것과 같은 방식으로 데드-레터 큐를 사용할 수 있습니다. 큐 관리자 오브젝트를 열고(MQOPEN 호출 사용) **DeadLetterQName** 속성을 조회하여(MQING 호출 사용) 데드-레터 큐의 이름을 찾을 수 있습니다.

큐 관리자가 이 큐에 메시지를 넣을 때 메시지에 헤더를 추가합니다. 해당 형식은 데드-레터 헤더(MQDLH) 구조에 설명되어 있습니다. [MQDLH - 데드-레터 헤더](#)를 참조하십시오. 이 헤더에는 대상 큐의 이름과 메시지를 데드-레터 큐에 넣는 이유가 포함됩니다. 이는 제거되어야 하며 메시지를 예정된 큐에 넣기 전에 이 문제를 해결해야 합니다. 또한 큐 관리자는 메시지에 MQDLH 구조가 포함됨을 표시하도록 메시지 디스크립터(MQMD)의 *Format* 필드를 변경합니다.

MQDLH 구조

데드-레터 큐에 넣은 모든 메시지에 MQDLH 구조를 추가하는 것이 좋습니다. 그러나 특정 IBM MQ 제품에서 제공한 데드-레터 핸들러를 사용하려는 경우 메시지에 MQDLH 구조를 추가해야 합니다.

메시지에 헤더를 추가하면 메시지가 데드-레터 큐에 대해 너무 길어질 수 있으므로 MQ_MSG_HEADER_LENGTH 상수 값을 사용하여 메시지가 데드-레터 큐에 허용된 최대 크기보다 작은지 항상

확인하십시오. 큐에 허용된 메시지의 최대 크기는 큐의 **MaxMsgLength** 속성 값으로 판별됩니다. 데드-레터 큐의 경우, 이 속성이 큐 관리자가 허용하는 최대값으로 설정되는지 확인하십시오. 애플리케이션이 메시지를 전달할 수 없고 메시지가 데드-레터 큐에 넣기에 너무 긴 경우 MQDLH 구조의 설명에 제공된 어드바이스에 따르십시오.

데드-레터 큐가 모니터되고 그에 도착하는 메시지가 처리되는지 확인하십시오. 데드-레터 큐 핸들러는 배치 유틸리티로 실행되며 데드-레터 큐의 선택된 메시지에 대해 다양한 조치를 수행하는 데 사용될 수 있습니다. 자세한 정보는 [1137 페이지의 『데드-레터 큐 처리』](#)의 내용을 참조하십시오.

데이터 변환이 필요한 경우 MOGET 호출에 MQGMO_CONVERT 옵션을 사용할 때 큐 관리자가 헤더 정보를 변환합니다. 메시지를 넣는 프로세스가 MCA인 경우 헤더 뒤에 원래 메시지의 모든 텍스트가 표시됩니다.

데드-레터 큐에 넣은 메시지가 이 큐에 대해 너무 긴 경우 이 메시지는 잘릴 수 있습니다. 이러한 상황은 큐의 **MaxMsgLength** 속성 값과 같은 길이인 데드-레터 큐에 메시지로 표시될 수 있습니다.

데드-레터 큐 처리

이 정보는 데드-레터 큐 처리를 사용할 때 범용 프로그래밍 인터페이스 정보를 포함합니다.

데드-레터 큐 처리는 로컬 시스템 요구사항에 따라 달라지지만 스펙을 작성할 때 다음을 고려하십시오.

- MQMD의 형식 필드 값이 MQFMT_DEAD_LETTER_HEADER이므로 메시지에 데드-레터 큐 헤더가 있는 것으로 식별될 수 있습니다.
- CICS를 사용하는 IBM MQ for z/OS 에서 MCA가 이 메시지를 데드-레터 큐에 넣으면 *PutApplType* 필드는 MQAT_CICS이고 *PutApplName* 필드는 MCA의 트랜잭션 이름이 뒤에 오는 CICS 시스템의 *ApplId* 입니다.
- 메시지가 데드-레터 큐로 라우팅되는 이유는 데드-레터 큐 헤더의 *Reason* 필드에 포함되어 있습니다.
- 데드-레터 큐 헤더에는 목적지 큐 이름과 큐 관리자 이름의 자세한 내용이 포함되어 있습니다.
- 메시지를 목적지 큐에 넣기 전에 메시지에 복원되어야 하는 필드가 데드-레터 큐 헤더에 포함되어 있습니다. 이러한 항목은 다음과 같습니다.

1. Encoding

2. CodedCharSetId

3. Format

- 메시지 디스크립터는 원본 애플리케이션에서 수행하는 PUT과 동일합니다. 단, 표시된 세 필드(Encoding, CodedCharSetId 및 Format)는 제외입니다.

데드-레터 큐 애플리케이션에서 다음 중 하나 이상을 수행해야 합니다.

- *Reason* 필드를 조사하십시오. 다음과 같은 이유로 인해 MCA에서 메시지를 넣을 수 있습니다.
 - 메시지가 채널의 최대 메시지 크기보다 큼니다.
이유는 MQRC_MSG_TOO_BIG_FOR_CHANNEL입니다.
 - 목적지 큐에 메시지를 넣을 수 없습니다.
이유는 MQPUT 조작을 통해 리턴할 수 있는 모든 MQRC_* 이유 코드입니다.
 - 사용자 엑시트에서 이 조치를 요청했습니다.
이유 코드는 사용자 엑시트 또는 기본 MQRC_SUPPRESSED_BY_EXIT에서 제공합니다.
- 가능한 경우 예정 목적지에 메시지를 전달하십시오.
- 전환 이유가 판별되었지만 즉시 정정할 수 없는 경우 제거하기 전에 일정 기간 동안 메시지를 보유하십시오.
- 문제점이 판별된 경우 문제점을 정정하도록 관리자에게 지시사항을 제공하십시오.
- 손상되었거나 처리할 수 없는 메시지를 제거하십시오.

데드-레터 큐에서 복구한 메시지를 처리하는 방법은 다음 두 가지입니다.

1. 메시지가 로컬 큐용인 경우 다음을 수행하십시오.

- 애플리케이션 데이터를 추출하는 데 필요한 코드 변환을 수행합니다.
- 로컬 함수인 경우 해당 데이터에서 코드 변환을 수행합니다.

- 메시지 디스크립터의 모든 세부 정보를 복원하여 로컬 큐에 결과 메시지를 넣습니다.

2. 메시지가 리모트 큐용인 경우 큐에 메시지를 넣습니다.

분산 큐잉 환경에서 미전달 메시지를 핸들링하는 방법에 관한 정보는 [메시지를 전달할 수 없을 때 발생하는 사항을 참조하십시오](#).

멀티캐스트 프로그래밍

이 정보를 사용하여 큐 관리자에 연결 및 예외 보고와 같은 IBM MQ Multicast 프로그래밍 태스크에 대해 알아보십시오.

IBM MQ Multicast는 사용자에게 최대한 투명하지만 기존 애플리케이션과 여전히 호환 가능하도록 디자인되었습니다. COMMINFO 오브젝트를 정의하고 TOPIC 오브젝트 **MCAST** 및 **COMMINFO** 매개변수를 설정하면 기존 IBM MQ 애플리케이션에서 멀티캐스트를 사용하기 위해 상당 부분을 다시 작성하지 않아도 됩니다. 그러나 몇 가지 제한사항(자세한 정보는 [1138 페이지의 『멀티캐스트 및 MQI』 참조](#))와 고려할 보안 문제(자세한 정보는 [Multicast 보안 참조](#))가 있을 수 있습니다.

멀티캐스트 및 MQI

이 정보를 사용하면 주요 MQI(Message Queue Interface) 개념 및 이들이 IBM MQ 멀티캐스트와 관련되는 방식을 이해할 수 있습니다.

멀티캐스트 subscription은 비지속적입니다. 연관된 물리적 큐가 없으므로 지속적 subscription에서 작성된 오프라인 메시지를 저장할 수 없습니다.

애플리케이션이 멀티캐스트 토픽을 구독한 후 큐에 대한 핸들인 것처럼 이용하거나 MQGET할 수 있는 오브젝트 핸들에 다시 제공됩니다. 이는 관리 대상 멀티캐스트 구독(MQSO_MANAGED로 작성된 구독)만 지원된다는 의미입니다. 즉, 구독을 작성하고 하나의 큐에 있는 메시지를 '지정할' 수 없습니다. 이렇게 되면 구독 호출 시 리턴되는 오브젝트 핸들에서 메시지를 이용해야 합니다. 클라이언트에서 메시지는 클라이언트가 이용할 때까지 메시지 버퍼에 저장됩니다. 자세한 정보는 [클라이언트 구성 파일의 MessageBuffer 스탠자](#)를 참조하십시오. 클라이언트가 발행 속도를 따르지 못하는 경우, 필요에 따라 오래된 메시지를 먼저 제거하는 순서로 메시지가 제거됩니다.

애플리케이션이 멀티캐스트를 사용하는지 여부는 일반적으로 TOPIC 오브젝트의 MCAST 속성을 설정하여 지정된 관리 의사결정입니다. 발행 애플리케이션이 멀티캐스트가 사용되지 않게 해야 하는 경우, MQOO_NO_MULTICAST 옵션을 사용할 수 있습니다. 마찬가지로, 구독 애플리케이션은 MQSO_NO_MULTICAST 옵션을 사용하여 구독함으로써 멀티캐스트가 사용되지 않게 할 수 있습니다.

IBM MQ 멀티캐스트에서는 메시지 선택자 사용을 지원합니다. 애플리케이션은 선택자를 사용하여 선택 문자열이 나타내는 SQL92 조회를 충족시키는 특성을 가진 해당 메시지에 대한 관심만 등록할 수 있습니다. 메시지 선택자에 대한 자세한 정보는 [36 페이지의 『선택자』](#)의 내용을 참조하십시오.

다음 표에는 모든 주요 MQI 개념 및 해당 개념이 멀티캐스트와 관련되는 방식이 나열되어 있습니다.

표 194. MQI 개념 및 해당 개념이 멀티캐스트와 관련되는 방식		
MQI 개념	멀티캐스트를 사용하려 할 때의 조치	이유 코드
0 길이의 메시지 넣기	거부됨	2005 (07D5) (RC2005): MQRC_BUFFER_LENGTH_ERROR
그룹화	거부됨	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Segmentation	거부됨	2443 (098B) (RC2443): MQRC_SEGMENTATION_NOT_ALLOWED
분배 목록	거부됨	2154 (086A) (RC2154): MQRC_RECS_PRESENT_ERROR

표 194. MQI 개념 및 해당 개념이 멀티캐스트와 관련되는 방식 (계속)		
MQI 개념	멀티캐스트를 사용하려 할 때의 조치	이유 코드
MQINQ	토픽 핸들에 대해 거부됨: 토픽의 MQINQ 및 MQSET이 지원되지 않습니다.	2038 (07F6) (RC2038): MQRC_NOT_OPEN_FOR_INQUIRE
MQINQ	관리 대상 핸들에 대해 채택됩니다. 현재 용량을 조회할 수 있습니다.	- 값이 현재 용량인 경우, 적용 가능한 이유 코드가 없습니다. - 현재 용량 이외의 값인 경우 이유 코드는 2067 (0813) (RC2067): MQRC_SELECTOR_ERROR입니다.
MQSET	모든 핸들에 대해 거부됩니다.	2040 (07F8) (RC2040): MQRC_NOT_OPEN_FOR_SET
트랜잭션(XA 또는 아님)	거부됨	2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE
메시지 찾아보기	거부됨	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
메시지 잠금	거부됨	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
표시로 찾아보기	거부됨	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
컨텍스트 전달	거부됨	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPUT1	거부되었습니다. 멀티캐스트 전용 토픽을 시도하고 MQPUT1을 수행하는 작업은 올바르지 않습니다.	2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
지속 가능 구독	토픽이 "멀티캐스트 전용"으로 표시되는 경우 거부되고 그렇지 않으면 멀티캐스트가 아닌 구독이 작성됩니다.	2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
TopicString > 255	거부되었습니다. 토픽 문자열이 255자보다 큰 경우 클라이언트에서 거부됩니다.	2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR
관리 대상이 아닌 구독이 작성됨	토픽이 "멀티캐스트 전용"으로 표시되는 경우 거부되고 그렇지 않으면 멀티캐스트가 아닌 구독이 작성됩니다.	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPMO_NOT_OWN_SUBS	거부됨	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR

다음 항목이 이전 표의 MQI 개념 일부에서 확장되며 표에 없는 MQI 개념 일부에 대한 정보를 제공합니다.

메시지 지속성

비지속적 멀티캐스트 구독의 경우, 발행자의 지속 메시지를 복구 불가능한 방식으로 전달합니다.

메시지 자르기

메시지 자르기가 지원되며 이렇게 되면 애플리케이션이 다음을 수행할 수 있습니다.

1. MQGET을 실행합니다.
2. MQRC_TRUNCATED_MSG_FAILED를 가져옵니다.
3. 보다 큰 버퍼를 할당합니다.
4. MQGET을 다시 실행하여 메시지를 검색합니다.

구독 만기

subscription 만기가 지원되지 않습니다. 만기를 설정하려는 시도가 무시됩니다.

멀티캐스트의 고가용성

이 정보를 사용하여 IBM MQ Multicast 지속 피어-투-피어 조작을 이해합니다. IBM MQ가 IBM MQ 큐 관리자에 연결하지만 메시지는 해당 큐 관리자를 따라 이동하지 않습니다.

멀티캐스트 토픽 오브젝트를 MQOPEN 또는 MQSUB하기 위해 관리자에 연결해야 하지만 메시지가 큐 관리자를 통해 플로우되지 않습니다. 따라서 멀티캐스트 토픽 오브젝트에서 MQOPEN 또는 MQSUB를 완료한 후에는 큐 관리자에 대한 연결이 끊긴 후에도 멀티캐스트 메시지를 계속 전송할 수 있습니다. 조작 모드에는 두 가지가 있습니다.

큐 관리자에 정상적으로 연결됨

큐 관리자에 연결된 동안에는 멀티캐스트 통신이 가능합니다. 연결에 실패하면 일반 MQI 규칙이 적용됩니다. 예를 들어, 멀티캐스트 오브젝트 핸들에 대한 MQPUT은 [2009 \(07D9\) \(RC2009\)](#): [MQRC_CONNECTION_BROKEN](#)을 리턴합니다.

큐 관리자에 클라이언트 연결이 다시 연결됨

재연결 주기 중에도 멀티캐스트 통신이 가능합니다. 이렇게 되면 큐 관리자에 대한 연결이 끊기더라도 멀티캐스트 메시지 넣기와 이용에는 영향을 주지 않습니다. 클라이언트가 큐 관리자에 다시 연결하려고 시도하며, 재연결에 실패하는 경우 연결 핸들이 중단되고 멀티캐스트 호출을 포함한 모든 MQI 호출이 실패합니다. 자세한 정보는 [자동 클라이언트 다시 연결](#)을 참조하십시오.

모든 애플리케이션이 명확하게 MQDISC를 발행하는 경우 모든 멀티캐스트 구독 및 오브젝트 핸들이 종료됩니다.

멀티캐스트의 지속적인 피어 투 피어 조작

클라이언트 간 피어 투 피어 통신을 사용하면 메시지가 큐 관리자를 통해 플로우되지 않아도 됩니다. 따라서 큐 관리자에 대한 연결이 끊겨도 메시지 전송이 계속됩니다. 다음 제한사항이 이 모드의 지속적인 메시지 요구사항에 적용됩니다.

- 지속적인 조작을 위해 MQCNO_RECONNECT * 옵션 중 하나를 사용하여 연결해야 합니다. 이 프로세스에서는 통신 세션이 중단되더라도 실제 연결 핸들은 중단되지 않으며 대신 재연결 상태에 있게 됩니다. 재연결에 실패하면 모든 추가 MQI 호출을 방해하는 연결 핸들이 중단됩니다.
- MQPUT, MQGET, MQINQ 및 Async Consume이 이 모드에서 지원됩니다. 모든 MQOPEN, MQCLOSE 또는 MQDISC 동사에서는 큐 관리자에 대한 재연결이 완료되어야 합니다.
- 큐 관리자에 대한 상태 플로우가 중지됩니다. 따라서 큐 관리자의 모든 상태는 오래되었거나 누락된 상태가 됩니다. 이렇게 되면 클라이언트가 메시지를 송신 및 수신할 수 있고 큐 관리자에서 알 수 없는 상태가 됩니다. 자세한 정보는 [멀티캐스트 애플리케이션 모니터링](#)을 참조하십시오.

멀티캐스트 메시징을 위한 MQI의 데이터 변환

이 정보를 사용하여 IBM MQ Multicast 메시징에서 데이터 변환이 작동하는 방식을 이해하십시오.

IBM MQ Multicast는 연결되지 않은 공유 프로토콜이므로, 각 클라이언트가 특정하게 데이터 변환을 요청할 수 없습니다. 동일한 멀티캐스트 스트림을 구독하는 모든 클라이언트가 동일한 2진 데이터를 수신합니다. 따라서 IBM MQ 데이터 변환이 필요한 경우 각 클라이언트에서 로컬로 변환을 수행합니다.

IBM MQ Multicast 트랙의 클라이언트에서 데이터가 변환됩니다. **MQGMO_CONVERT** 옵션을 지정하면 데이터 변환이 요청대로 수행됩니다. 사용자 정의 형식에는 클라이언트에 설치된 데이터 변환 엑시트가 필요합니다. 클라이언트 및 서버 패키지에 있는 라이브러리에 대한 정보는 [729 페이지의 『데이터 변환 엑시트 작성』](#)의 내용을 참조하십시오.

데이터 변환을 관리하는 데 대한 정보는 [Multicast 메시징에서 데이터 변환 사용](#)을 참조하십시오.

데이터 변환에 대한 자세한 정보는 [데이터 변환](#)을 참조하십시오.

데이터 변환 엑시트와 ClientExitPath에 대한 자세한 정보는 [클라이언트 구성 파일의 ClientExitPath 스탠자](#)를 참조하십시오.

멀티캐스트 예외 보고

이 정보를 사용하여 IBM MQ 멀티캐스트 이벤트 핸들러 및 보고 IBM MQ 멀티캐스트 예외에 대해 알아보십시오.

IBM MQ 멀티캐스트는 표준 IBM MQ 이벤트 핸들러 메커니즘을 사용하여 보고되는 멀티캐스트 이벤트를 보고하도록 이벤트 핸들러를 호출하여 문제점 판별을 지원합니다.

단일 멀티캐스트 이벤트는 동일한 멀티캐스트 송신기 또는 수신기를 사용하는 연결 핸들이 MQHCONN 여러 개 존재할 수 있으므로, 하나 이상의 IBM MQ 이벤트가 호출될 수 있습니다. 그러나 멀티캐스트 예외가 발생할 때마다 IBM MQ 연결당 하나의 이벤트 핸들러만 호출됩니다.

IBM MQ MQCBDO_EVENT_CALL 상수를 사용하면 애플리케이션이 IBM MQ 이벤트만 수신하도록 콜백을 등록할 수 있고 MQCBDO_MC_EVENT_CALL을 사용하면 애플리케이션이 멀티캐스트 이벤트만 수신하도록 콜백을 등록할 수 있습니다. 두 상수 모두 사용하는 경우, 두 가지 유형의 이벤트 모두 수신할 수 있습니다.

멀티캐스트 이벤트 요청

IBM MQ 멀티캐스트 이벤트는 cbd.Options 필드에서 상수를 MQCBDO_MC_EVENT_CALL 사용합니다. 다음 예는 멀티캐스트 이벤트 요청 방법을 보여줍니다.

```
cbd.CallbackType      = MQCBT_EVENT_HANDLER;
cbd.Options           = MQCBDO_MC_EVENT_CALL;
cbd.CallbackFunction = EventHandler;
MQCB(Hcon, MQOP_REGISTER, &cbd, MQHO_UNUSABLE_HOBJ, NULL, NULL, &CompCode, &Reason);
```

필드에 cbd.Options 옵션이 MQCBDO_MC_EVENT_CALL 지정된 경우, 이벤트 핸들러는 연결 수준 이벤트 대신 멀티캐스트 IBM MQ 이벤트만 수신합니다. 두 가지 유형의 이벤트를 이벤트 핸들러로 송신하도록 요청하려면 애플리케이션이 다음 예에 표시된 대로 MQCBDO_MC_EVENT_CALL 상수 및 cbd.Options 필드의 MQCBDO_EVENT_CALL 상수를 지정해야 합니다.

```
cbd.CallbackType      = MQCBT_EVENT_HANDLER;
cbd.Options           = MQCBDO_EVENT_CALL | MQCBDO_MC_EVENT_CALL;
cbd.CallbackFunction = EventHandler;
MQCB(Hcon, MQOP_REGISTER, &cbd, MQHO_UNUSABLE_HOBJ, NULL, NULL, &CompCode, &Reason);
```

이러한 상수를 사용하지 않으면 연결 레벨 이벤트만 이벤트 핸들러로 송신됩니다.

Options 필드의 값에 대한 자세한 정보는 [옵션\(MQLONG\)](#)을 참조하십시오.

멀티캐스트 이벤트 형식

IBM MQ 멀티캐스트 예외에는 콜백 함수의 **Buffer** 매개변수에 리턴된 지원 정보가 포함되어 있습니다.

Buffer 포인터는 포인터의 배열을 가리키고 MQCBC.DataLength 필드는 배열의 크기(바이트)를 지정합니다. 배열의 첫 번째 요소는 항상 이벤트의 짧은 텍스트 설명을 가리킵니다. 이벤트 유형에 따라 추가 매개변수가 제공될 수 있습니다. 다음 표에는 예외 목록이 있습니다.

표 195. 멀티캐스트 이벤트 코드 설명		
이벤트 코드	설명	추가 데이터
MQMCEV_PACKET_LOSS	복구할 수 없는 패킷 손실	손실된 패킷의 수

표 195. 멀티캐스트 이벤트 코드 설명 (계속)		
이벤트 코드	설명	추가 데이터
MQMCEV_HEARTBEAT_TIMEOUT	하트비트 제어 패킷의 장기 부재	해당사항 없음
MQMCEV_VERSION_CONFLICT	최신 프로토콜 버전 패킷의 수신	해당사항 없음
MQMCEV_RELIABILITY	전송자 및 수신자의 서로 다른 신뢰 모드	해당사항 없음
MQMCEV_CLOSED_TRANS	1개의 소스에 의해 종료된 토픽 전송	해당사항 없음
MQMCEV_STREAM_ERROR	스트림에서 발견된 오류	해당사항 없음
MQMCEV_NEW_SOURCE	토픽에서 전송을 시작하는 새 소스	소스 구조
MQMCEV_RECEIVE_QUEUE_TRIMMED	시간 또는 공간 만기로 인해 PacketQ에서 제거된 패킷	조정된 패킷의 수
MQMCEV_PACKET_LOSS_NACK_EXPIRE	NACK 만기로 인한 복구할 수 없는 패킷 손실	손실된 패킷의 수
MQMCEV_ACK_RETRIES_EXCEEDED	max_ack_retries 를 초과한 후 실행 기록에서 제거된 패킷의 수	제거된 패킷 수
MQMCEV_STREAM_SUSPEND_NACK	NACK이 이 토픽에서 채택된 스트림에서 일시중단됨	스트림 ID 일시중단 스트림이 일시중단된 시간 (밀리초)
MQMCEV_STREAM_RESUME_NACK	NACK이 스트림에서 일시중단된 후 재개됨	스트림 ID
MQMCEV_STREAM_EXPELLED	이 토픽에서 채택된 스트림이 강제 제거 요청으로 인해 거부됨	스트림 ID
MQMCEV_FIRST_MESSAGE	소스의 첫 번째 메시지	메시지 번호
MQMCEV_LATE_JOIN_FAILURE	늦은 조인 세션을 시작하는 데 실패함	해당사항 없음
MQMCEV_MESSAGE_LOSS	복구할 수 없는 메시지 손실	손실된 메시지의 수
MQMCEV_SEND_PACKET_FAILURE	멀티캐스트 전송자가 멀티캐스트 패킷을 송신하는 데 실패함	해당사항 없음
MQMCEV_REPAIR_DELAY	멀티캐스트 수신자가 미해결 NAK에 대한 복구 패킷을 수신하지 못함	해당사항 없음
MQMCEV_MEMORY_ALERT_ON	수신자 수신 버퍼를 채우고 있음	버퍼 풀 사용 백분율
MQMCEV_MEMORY_ALERT_OFF	수신자 수신 버퍼 수가 정상으로 내려감	버퍼 풀 사용 백분율
MQMCEV_NACK_ALERT_ON	수신자 복구 패킷 요청 비율이 최고에 도달함	초당 패킷의 현재 복구 요청 비율
MQMCEV_NACK_ALERT_OFF	수신자 복구 패킷 요청 비율이 정상으로 내려감	초당 패킷의 현재 복구 요청 비율
MQMCEV_REPAIR_ALERT_ON	전송자 복구 패킷 송신 비율이 최고에 도달함	해당사항 없음
MQMCEV_REPAIR_ALERT_OFF	전송자 복구 패킷 송신 비율이 정상으로 내려감	해당사항 없음

표 195. 멀티캐스트 이벤트 코드 설명 (계속)		
이벤트 코드	설명	추가 데이터
MQMCEV_SHM_DEST_UNUSABLE	전송자 토픽 목적지에서 사용되는 공유 메모리 영역이 사용할 수 없는 영역으로 감지됨	해당사항 없음
MQMCEV_SHM_PORT_UNUSABLE	수신자 인스턴스에서 사용되는 공유 메모리 포트가 사용할 수 없는 영역으로 감지됨	해당사항 없음
MQMCEV_CCT_GETTIME_FAILED	CCT(Coordinated Cluster Time)에서 시간 구하기에 실패함	해당사항 없음
MQMCEV_DEST_INTERFACE_FAILURE	전송자 토픽 목적지에서 사용되는 네트워크 인터페이스가 실패하고 백업 네트워크 인터페이스를 사용할 수 없음	
MQMCEV_DEST_INTERFACE_FAILOVER	전송자 토픽 목적지에서 사용되는 네트워크 인터페이스가 실패하고 다른 인터페이스에 대한 장애 복구에 성공함	
MQMCEV_PORT_INTERFACE_FAILURE	수신자 rmmPort에서 사용되는 네트워크 인터페이스가 실패하고 백업 네트워크 인터페이스를 사용할 수 없음(또는 실패함)	RMM 구성
MQMCEV_PORT_INTERFACE_FAILOVER	수신자 rmmPort에서 사용되는 네트워크 인터페이스가 실패하고 다른 인터페이스에 대한 장애 복구에 성공함	RMM 구성

C 언어로 ‘MQI’ 앱 개발하기

C로 IBM MQ 프로그램을 코딩할 때 다음 섹션의 정보를 참고하십시오.

- [1143 페이지의 『MQI 호출의 매개변수』](#)
- [1144 페이지의 『데이터 유형이 정의되지 않은 매개변수』](#)
- [1144 페이지의 『데이터 유형』](#)
- [1144 페이지의 『2진 문자열 조작』](#)
- [1144 페이지의 『문자열 조작』](#)
- [1144 페이지의 『구조의 초기값』](#)
- [1145 페이지의 『동적 구조의 초기값』](#)
- [1145 페이지의 『C++에서 사용』](#)

MQI 호출의 매개변수

입력 전용이며 MQHCONN, MQHOBJ, MQHMSG 또는 MQLONG 유형의 매개변수는 값에 의해 전달됩니다. 그 외 다른 매개변수의 경우 매개변수의 주소가 값에 의해 전달됩니다.

주소에 의해 전달되는 모든 매개변수를 함수가 호출될 때마다 지정해야 하는 것은 아닙니다. 특정 매개변수가 필요하지 않은 경우 매개변수 데이터의 주소 대신에 널 포인터를 함수 호출의 매개변수로 지정할 수 있습니다. 이러한 매개변수는 호출 설명에서 식별됩니다.

함수 값으로 리턴되는 매개변수는 없습니다. C 용어에서 이는 모든 함수가 void를 리턴한다는 것을 의미합니다.

함수의 속성은 MQENTRY 매크로 변수에서 정의합니다. 이 매크로 변수의 값은 환경에 따라 달라집니다.

데이터 유형이 정의되지 않은 매개변수

MQGET, MQPUT 및 MQPUT1 함수에는 각각 정의되지 않은 데이터 유형이 있는 **Buffer** 매개변수가 있습니다. 이 매개변수는 애플리케이션의 메시지 데이터를 송신하고 수신하는 데 사용됩니다.

이러한 종류의 매개변수는 MQBYTE의 배열로 C 예제에 표시됩니다. 이러한 방식으로 매개변수를 선언할 수 있지만 일반적으로 메시지의 데이터 레이아웃을 설명하는 구조로 매개변수를 선언하는 것이 더 편리합니다. 함수 매개변수는 void에 대한 포인터로 선언되므로 데이터의 주소를 함수 호출의 매개변수로 지정할 수 있습니다.

데이터 유형

모든 데이터 유형은 typedef 명령문으로 정의됩니다.

각 데이터 유형의 경우 해당하는 포인터 데이터 유형도 정의됩니다. 포인터 데이터 유형의 이름은 포인터를 표시하기 위해 문자 P가 접두부로 사용되는 기본 또는 구조 데이터 유형의 이름입니다. 포인터의 속성은 MQPOINTER 매크로 변수에 의해 정의됩니다. 이 매크로 변수의 값은 환경에 따라 달라집니다. 다음 코드는 포인터 데이터 유형을 선언하는 방법에 대해 설명합니다.

```
#define MQPOINTER /* depends on environment */
...
typedef MQLONG MQPOINTER PMQLONG; /* pointer to MQLONG */
typedef MQMD MQPOINTER PMQMD; /* pointer to MQMD */
```

2진 문자열 조작

2진 데이터의 문자열은 MQBYTEn 데이터 유형 중 하나로 선언됩니다.

이 유형의 필드를 복사, 비교 또는 설정할 때마다, C 함수 memcpy, memcmp 또는 memset를 사용하십시오.

```
#include <string.h>
#include "cmqc.h"

MQMD MyMsgDesc;

memcpy(MyMsgDesc.MsgId, /* set "MsgId" field to nulls */
       MQMI_NONE, /* ...using named constant */
       sizeof(MyMsgDesc.MsgId));

memset(MyMsgDesc.CorrelId, /* set "CorrelId" field to nulls */
       0x00, /* ...using a different method */
       sizeof(MQBYTE24));
```

MQBYTE24로 선언되는 데이터와 올바르게 작동하지 않기 때문에 문자열 함수 strcpy, strcmp, strncmp 또는 strncmp를 사용하지 마십시오.

문자열 조작

큐 관리자가 문자 데이터를 애플리케이션에 리턴할 때, 큐 관리자는 항상 정의된 필드 길이만큼 공백으로 문자 데이터를 채웁니다. 큐 관리자는 널(Null) 종로 문자열을 리턴하지 않지만, 사용자는 이를 입력에서 사용할 수 있습니다. 따라서 이러한 문자열을 복사, 비교 또는 연결하는 경우 문자열 함수 strncpy, strncmp 또는 strncat을 사용하십시오.

널에 의해 종료되는 문자열이 필요한 문자열 함수를 사용하지 마십시오(strcpy, strcmp 및 strcat). 또한 문자열의 길이를 판별하는 데 함수 strlen을 사용하지 마십시오. 대신 함수의 크기를 사용하여 필드의 길이를 판별하십시오.

구조의 초기값

포함 파일 <cmqc.h>는 이러한 구조의 인스턴스를 선언할 때 구조에 대한 초기값을 제공하는 데 사용할 수 있는 다양한 매크로 변수를 정의합니다. 이러한 매크로 변수에 MQxxx_DEFAULT 양식의 이름이 있으며 여기서 MQxxx는 구조의 이름을 나타냅니다. 다음과 같이 사용하십시오.

```
MQMD MyMsgDesc = {MQMD_DEFAULT};
MQPMO MyPutOpts = {MQPMO_DEFAULT};
```


일부 문자 필드의 경우 MQI는 올바른 특정 값을 정의합니다(예를 들어, MQMD의 *Format* 필드 또는 *StrucId* 필드의 경우). 각 올바른 값의 경우 두 개의 매크로 변수가 제공됩니다.

- 하나의 매크로 변수는 정의된 필드 길이와 정확히 일치하는 내재된 널을 제외하는 길이가 포함된 문자열로 값을 정의합니다. 다음 예시에서 기호는 단일 공백 문자를 나타냅니다:

```
#define MQMD_STRUC_ID "MD~"
#define MQFMT_STRING "MQSTR~"
```

memcpy 및 memcmp 함수와 함께 이 양식을 사용하지 마십시오.

- 다른 매크로 변수는 문자 배열로서 값을 정의합니다. 이 매크로 변수의 이름은 _ARRAY가 접미부로 사용된 문자열 양식의 이름입니다. 예를 들면, 다음과 같습니다.

```
#define MQMD_STRUC_ID_ARRAY 'M','D',' ',' ',' '
#define MQFMT_STRING_ARRAY 'M','Q','S','T','R',' ',' ',' ' "
```

구조의 인스턴스가 MQMD_DEFAULT 매크로 변수에서 제공하는 값과 다른 값으로 선언되는 경우 이 양식을 사용하여 필드를 초기화하십시오.

동적 구조의 초기값

구조의 인스턴스 변수 번호가 필요한 경우 일반적으로 인스턴스는 calloc 또는 malloc 함수를 수용하여 동적으로 확보한 기본 스토리지에 작성됩니다.

이러한 구조에서 필드를 초기화하려면 다음 기술을 사용하는 것이 좋습니다.

1. 구조를 초기화하기 위해 적절한 MQxxx_DEFAULT 매크로 변수를 사용하여 구조의 인스턴스를 선언하십시오. 이 인스턴스는 다른 인스턴스에 대한 모델이 됩니다.

```
MQMD ModelMsgDesc = {MQMD_DEFAULT};
/* declare model instance */
```

필요에 따라 모델 인스턴스에 정적 또는 동적 수명을 제공하기 위해 선언에서 정적 또는 자동 키워드를 코드화하십시오.

2. calloc 또는 malloc 함수를 사용하여 구조의 동적 인스턴스에 대한 스토리지를 확보하십시오.

```
PMQMD InstancePtr;
InstancePtr = malloc(sizeof(MQMD));
/* get storage for dynamic instance */
```

3. memcpy 함수를 사용하여 모델 인스턴스를 동적 인스턴스에 복사하십시오.

```
memcpy(InstancePtr,&ModelMsgDesc,sizeof(MQMD));
/* initialize dynamic instance */
```

C++에서 사용

C++ 프로그래밍 언어의 경우 C++ 컴파일러가 사용될 때만 포함되는 다음 추가 명령문이 헤더 파일에 포함됩니다.

```
#ifdef __cplusplus
extern "C" {
#endif

/* rest of header file */

#ifdef __cplusplus
}
#endif
```

관련 참조

972 페이지의 『C 및 COBOL MQI 샘플에 표시된 기능 AIX and Linux』
샘플 프로그램에서 시연한 기술은,에 IBM MQ 관한 AIX and Linux것입니다.

975 페이지의 『C 및 COBOL MQI 샘플에 표시된 기능 Windows』

샘플 프로그램에서 시연한 기술은.에 IBM MQ 관한 Windows것입니다.

977 페이지의 『C, COBOL 및 RPG MQI 샘플에 표시된 기능은 다음과 같습니다. IBM i』
샘플 프로그램에서 시연한 기법을 IBM MQIBM i 보여줍니다.

1080 페이지의 『MQI 샘플에 표시된 기능은 다음과 같습니다. z/OS』

각 샘플 애플리케이션에서 시연된 ‘MQI’ 기능에 대한 요약으로, 각 샘플이 작성된 프로그래밍 언어와 실행 환경이 포함됩니다.

C 예제: 큐 매니저에 연결하기

이 예에서는 MQCONN 호출을 사용하여 프로그램을 z/OS 일괄처리의 큐 관리자에 연결하는 방법을 보여줍니다.

이 발췌문은 Browse 샘플 애플리케이션에서 가져온 것입니다.  이것은 IBM MQ for z/OS 프로그램 CSQ4BCA1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

-  다중 플랫폼용 샘플 절차형 프로그램 사용
-  다음에 대한 예제 절차형 프로그램을 사용하여 z/OS

```
#include <cmqc.h>
:
static char Parm1[MQ_Q_MGR_NAME_LENGTH] ;

int main(int argc, char *argv[] )
{
    /*
    /*      Variables for MQ calls
    /*
    /*      MQHCONN Hconn;      /* Connection handle
    /*      MQLONG  CompCode;   /* Completion code
    /*      MQLONG  Reason;     /* Qualifying reason
    /*
    /* Copy the queue manager name, passed in the
    /* parm field, to Parm1
    strncpy(Parm1,argv[1],MQ_Q_MGR_NAME_LENGTH);

    /*
    /* Connect to the specified queue manager.
    /* Test the output of the connect call. If the
    /* call fails, print an error message showing the
    /* completion code and reason code, then leave the
    /* program.
    /*
    MQCONN(Parm1,
           &Hconn,
           &CompCode,
           &Reason);
    if ((CompCode != MQCC_OK) | (Reason != MQRC_NONE))
    {
        sprintf(pBuff, MESSAGE_4_E,
                ERROR_IN_MQCONN, CompCode, Reason);
        PrintLine(pBuff);
        RetCode = CSQ4_ERROR;
        goto AbnormalExit2;
    }
    :
}
```

C 예제: 큐 매니저에서 연결 해제하기

이 예에서는 MQDISC 호출을 사용하여 z/OS 일괄처리의 큐 관리자에서 프로그램의 연결을 끊는 방법을 보여줍니다.

이 코드 추출에서 사용된 변수는 1146 페이지의 『C 예제: 큐 매니저에 연결하기』에서 설정된 변수입니다. 이 발췌문은 Browse 샘플 애플리케이션(프로그램 CSQ4BCA1)에서 가져온 것입니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 다음에 대한 예제 절차형 프로그램을 사용하여 z/OS.

```

:
/*
/* Disconnect from the queue manager. Test the
/* output of the disconnect call. If the call
/* fails, print an error message showing the
/* completion code and reason code.
/*
MQDISC(&Hconn,
      &CompCode,
      &Reason);
if ((CompCode != MQCC_OK) || (Reason != MQRC_NONE))
{
    sprintf(pBuff, MESSAGE_4_E,
            ERROR_IN_MQDISC, CompCode, Reason);
    PrintLine(pBuff);
    RetCode = CSQ4_ERROR;
}
:

```

C 예제: 동적 큐 생성

이 예제는 MQOPEN 호출을 사용하여 동적 큐를 작성하는 방법을 보여줍니다.

이 발췌문은 메일 관리자 샘플 애플리케이션에서 가져온 것입니다. **z/OS** 이것은 IBM MQ for z/OS 프로그램 CSQ4TCD1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

- ▶ **Multi** 다중 플랫폼용 샘플 절차형 프로그램 사용
- ▶ **z/OS** 다음에 대한 예제 절차형 프로그램을 사용하여 z/OS

```

:
MQLONG HCONN = 0; /* Connection handle */
MQHOBJ HOBJ; /* MailQ Object handle */
MQHOBJ HobjTempQ; /* TempQ Object Handle */
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Qualifying reason */
MQOD ObjDesc = {MQOD_DEFAULT}; /* Object descriptor */
MQLONG OpenOptions; /* Options control MQOPEN */

/*-----*/
/* Initialize the Object Descriptor (MQOD) */
/* control block. (The remaining fields */
/* are already initialized.) */
/*-----*/
strncpy( ObjDesc.ObjectName,
        SYSTEM_REPLY_MODEL,
        MQ_Q_NAME_LENGTH );
strncpy( ObjDesc.DynamicQName,
        SYSTEM_REPLY_INITIAL,
        MQ_Q_NAME_LENGTH );
OpenOptions = MQOO_INPUT_AS_Q_DEF;
/*-----*/
/* Open the model queue and, therefore, */
/* create and open a temporary dynamic */
/* queue */
/*-----*/
MQOPEN( HCONN,
        &ObjDesc,
        OpenOptions,
        &HobjTempQ,
        &CompCode,
        &Reason );
if ( CompCode == MQCC_OK ) {
}
else {
    /*-----*/
    /* Build an error message to report the */
    /* failure of the opening of the model */
    /* queue */
    /*-----*/
    MQMErrorHandler( "OPEN TEMPQ", CompCode,
                    Reason );
}

```

```

        ErrorFound = TRUE;
    }
    return ErrorFound;
}
...

```

C 예제: 기존 큐 열기

이 예제는 MQOPEN 호출을 사용하여 이미 정의된 큐를 여는 방법을 설명합니다.

이 발췌문은 Browse 샘플 애플리케이션에서 가져온 것입니다.  이것은 IBM MQ for z/OS 프로그램 CSQ4BCA1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

-  [다중 플랫폼용 샘플 절차형 프로그램 사용](#)
-  [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#)

```

#include <cmqc.h>
...
static char Parm1[MQ_Q_MGR_NAME_LENGTH];
...
int main(int argc, char *argv[] )
{
    /*
     * Variables for MQ calls
     */
    MQHCONN Hconn ;           /* Connection handle */
    MQLONG  CompCode;         /* Completion code */
    MQLONG  Reason;           /* Qualifying reason */
    MQOD    ObjDesc = { MQOD_DEFAULT };
    /* Object descriptor */
    MQLONG  OpenOptions;      /* Options that control
     * the MQOPEN call */
    MQHOBJ  Hobj;             /* Object handle */
    ...
    /* Copy the queue name, passed in the parm field,
     * to Parm2 strncpy(Parm2,argv[2],
     * MQ_Q_NAME_LENGTH);
     */
    ...
    /* Initialize the object descriptor (MQOD) control
     * block. (The initialization default sets StrucId,
     * Version, ObjectType, ObjectQMgrName,
     * DynamicQName, and AlternateUserid fields)
     */
    strncpy(ObjDesc.ObjectName,Parm2,MQ_Q_NAME_LENGTH);
    ...
    /* Initialize the other fields required for the open
     * call (Hobj is set by the MQCONN call).
     */
    OpenOptions = MQOO_BROWSE;
    ...
    /*
     * Open the queue.
     * Test the output of the open call. If the call
     * fails, print an error message showing the
     * completion code and reason code, then bypass
     * processing, disconnect and leave the program.
     */
    MQOPEN(Hconn,
           &ObjDesc,
           OpenOptions,
           &Hobj,
           &CompCode,
           &Reason);

    if ((CompCode != MQCC_OK) || (Reason != MQRC_NONE))
    {
        sprintf(pBuff, MESSAGE_4_E,
                ERROR_IN_MQOPEN, CompCode, Reason);
        PrintLine(pBuff);
        RetCode = CSQ4_ERROR;
        goto AbnormalExit1; /* disconnect processing */
    }
}

```

```

    }
    :
    } /* end of main */

```

C 예제: 큐 닫기

이 예제는 MQCLOSE 호출을 사용하여 큐를 닫는 방법을 보여줍니다.

이 발췌문은 Browse 샘플 애플리케이션에서 가져온 것입니다. **z/OS** 이것은 IBM MQ for z/OS 프로그램 CSQ4BCA1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

- ▶ **Multi** [다중 플랫폼용 샘플 절차형 프로그램 사용](#)
- ▶ **z/OS** [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#)

```

:
/*                                     */
/* Close the queue.                  */
/* Test the output of the close call. If the call */
/* fails, print an error message showing the */
/* completion code and reason code.      */
/*                                     */
MQCLOSE(Hconn,
        &Hobj,
        MQCO_NONE,
        &CompCode,
        &Reason);
if ((CompCode != MQCC_OK) || (Reason != MQRC_NONE))
{
    sprintf(pBuff, MESSAGE_4_E,
            ERROR_IN_MQCLOSE, CompCode, Reason);
    PrintLine(pBuff);
    RetCode = CSQ4_ERROR;
}
:

```

C 예제: MQPUT을 사용하여 메시지 전송

이 예제는 MQPUT 호출을 사용하여 큐에 메시지를 넣는 방법을 보여줍니다.

이 발췌문은 함께 제공된 샘플 애플리케이션에서 가져온 것이 아닙니다. IBM MQ

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

- ▶ **Multi** [다중 플랫폼용 샘플 절차형 프로그램 사용](#)
- ▶ **z/OS** [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#)
- ▶ **Multi** [971 페이지의 『멀티플랫폼용 샘플 절차적 프로그램 사용』](#)
- ▶ **z/OS** [1079 페이지의 『다음의 샘플 절차적 프로그램을 사용하여 z/OS』](#)

```

:
qput()
{
    MQMD      MsgDesc;
    MQPMO     PutMsgOpts;
    MQLONG    CompCode;
    MQLONG    Reason;
    MQHCONN   Hconn;
    MQHOBJ    Hobj;
    char message_buffer[] = "MY MESSAGE";
    /*-----*/
    /* Set up PMO structure. */
    /*-----*/
    memset(&PutMsgOpts, '\0', sizeof(PutMsgOpts));
    memcpy(PutMsgOpts.StrucId, MQPMO_STRUC_ID,
           sizeof(PutMsgOpts.StrucId));
    PutMsgOpts.Version = MQPMO_VERSION_1;
    PutMsgOpts.Options = MQPMO_SYNCPOINT;
}

```

```

/*-----*/
/* Set up MD structure. */
/*-----*/
memset(&MsgDesc, '\0', sizeof(MsgDesc));
memcpy(MsgDesc.StrucId, MQMD_STRUC_ID,
       sizeof(MsgDesc.StrucId));
MsgDesc.Version      = MQMD_VERSION_1;
MsgDesc.Expiry       = MQEI_UNLIMITED;
MsgDesc.Report       = MQRO_NONE;
MsgDesc.MsgType      = MQMT_DATAGRAM;
MsgDesc.Priority     = 1;
MsgDesc.Persistence  = MQPER_PERSISTENT;
memset(MsgDesc.ReplyToQ,
       '\0',
       sizeof(MsgDesc.ReplyToQ));
/*-----*/
/* Put the message. */
/*-----*/
MQPUT(Hconn, Hobj, &MsgDesc, &PutMsgOpts,
      sizeof(message_buffer), message_buffer,
      &CompCode, &Reason);

```

```

/*-----*/
/* Check completion and reason codes. */
/*-----*/
switch (CompCode)
{
    case MQCC_OK:
        break;
    case MQCC_FAILED:
        switch (Reason)
        {
            case MQRC_Q_FULL:
            case MQRC_MSG_TOO_BIG_FOR_Q:
                break;
            default:
                break; /* Perform error processing */
        }
        break;
    default:
        break; /* Perform error processing */
}
}

```

C 예제: `MQPUT1`를 사용하여 메시지 전송하기

이 예제는 MQPUT1 호출을 사용하여 큐를 열고 단일 메시지를 큐에 넣은 후에 큐를 닫는 방법을 보여줍니다.

이 발췌문은 신용 조회 샘플 신청서에서 가져온 것입니다.  이것은 IBM MQ for z/OS 프로그램 CSQ4CCB5 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

-  다중 플랫폼용 샘플 절차형 프로그램 사용
-  다음에 대한 예제 절차형 프로그램을 사용하여 z/OS

```

:
MQLONG  Hconn;           /* Connection handle */
MQHOBJ  Hobj_CheckQ;    /* Object handle */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Qualifying reason */
MQOD    ObjDesc = {MQOD_DEFAULT};
MQMD    MsgDesc = {MQMD_DEFAULT};
MQLONG  OpenOptions;    /* Message descriptor */
MQLONG  OpenOptions;    /* Control the MQOPEN call */

MQGMO    GetMsgOpts = {MQGMO_DEFAULT};
MQLONG  MsgBufLen;      /* Get Message Options */
CSQ4BCAQ MsgBuffer;     /* Length of message buffer */
MQLONG  DataLen;        /* Message structure */
MQLONG  DataLen;        /* Length of message */

MQPMO    PutMsgOpts = {MQPMO_DEFAULT};

```



```

MQLONG Reason;          /* Qualifying reason      */
MQHOBJ Hobj;            /* Object handle      */
MQMD MsgDesc = { MQMD_DEFAULT };
                        /* Message descriptor */
MQLONG DataLength;      /* Length of the message */
MQCHAR Buffer[BUFFERLENGTH+1];
                        /* Area for message data */
MQGMO GetMsgOpts = { MQGMO_DEFAULT };
                        /* Options which control */
                        /* the MQGET call      */
MQLONG BufferLength = BUFFERLENGTH;
                        /* Length of buffer      */
:
/* No need to change the message descriptor */
/* (MQMD) control block because initialization */
/* default sets all the fields.              */
/*                                           */
/* Initialize the get message options (MQGMO) */
/* control block (the copy file initializes all */
/* the other fields).                        */
/*                                           */
GetMsgOpts.Options = MQGMO_NO_WAIT +
                    MQGMO_BROWSE_FIRST +
                    MQGMO_ACCEPT_TRUNCATED_MSG;

/* Get the first message.                    */
/* Test for the output of the call is carried out */
/* in the 'for' loop.                        */
MQGET(Hconn,
      Hobj,
      &MsgDesc,
      &GetMsgOpts,
      BufferLength,
      Buffer,
      &DataLength,
      &CompCode,
      &Reason);

```

```

/* Process the message and get the next message, */
/* until no messages remaining.                  */
:
/* If the call fails for any other reason,        */
/* print an error message showing the completion */
/* code and reason code.                        */
if ( (CompCode == MQCC_FAILED) &&
     (Reason == MQRC_NO_MSG_AVAILABLE) )
{
    :
}
else
{
    sprintf(pBuff, MESSAGE_4_E,
            ERROR_IN_MQGET, CompCode, Reason);
    PrintLine(pBuff);
    RetCode = CSQ4_ERROR;
}
:
} /* end of main */

```

C 예제: 대기 옵션을 사용하여 메시지 받기

이 예제는 MQGET 호출의 대기 옵션을 사용하는 방법을 보여줍니다.

이 코드는 잘린 메시지를 허용합니다. 이 발췌문은 신용 조회 샘플 신청서에서 가져온 것입니다. **z/OS** 이
것은 IBM MQ for z/OS 프로그램 CSQ4CCB5 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

- ▶ **Multi** 다중 플랫폼용 샘플 절차형 프로그램 사용
- ▶ **z/OS** 다음에 대한 예제 절차형 프로그램을 사용하여 z/OS

```

:
MQLONG Hconn; /* Connection handle */
MQHOBJ Hobj_CheckQ; /* Object handle */
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Qualifying reason */
MQOD ObjDesc = {MQOD_DEFAULT}; /* Object descriptor */
MQMD MsgDesc = {MQMD_DEFAULT}; /* Message descriptor */
MQLONG OpenOptions;
MQGMO GetMsgOpts = {MQGMO_DEFAULT}; /* Get Message Options */
MQLONG MsgBufLen; /* Length of message buffer */
CSQ4BCAQ MsgBuffer; /* Message structure */
MQLONG DataLen; /* Length of message */

```

```

:
void main(void)
{
:
/*
/* Initialize options and open the queue for input */
/*
:
/*
/* Get and process messages */
/*
/*
GetMsgOpts.Options = MQGMO_WAIT +
                    MQGMO_ACCEPT_TRUNCATED_MSG +
                    MQGMO_SYNCPOINT;
GetMsgOpts.WaitInterval = WAIT_INTERVAL;
MsgBufLen = sizeof(MsgBuffer);
memcpy(MsgDesc.MsgId, MQMI_NONE,
        sizeof(MsgDesc.MsgId));
memcpy(MsgDesc.CorrelId, MQCI_NONE,
        sizeof(MsgDesc.CorrelId));

/*
/* Make the first MQGET call outside the loop */
/*
/*
MQGET(Hconn,
      Hobj_CheckQ,
      &MsgDesc,
      &GetMsgOpts,
      MsgBufLen,
      &MsgBuffer,
      &DataLen,
      &CompCode,
      &Reason);

:
/*
/* Test the output of the MQGET call. If the call */
/* failed, send an error message showing the */
/* completion code and reason code, unless the */
/* reason code is NO_MSG_AVAILABLE. */
/*
/*
if (Reason != MQRC_NO_MSG_AVAILABLE)
{
    strncpy(TS_Operation, "MQGET", sizeof(TS_Operation));
    strncpy(TS_ObjName, ObjDesc.ObjectName,
            MQ_Q_NAME_LENGTH);
    Record_Call_Error();
}
:
}

```

C 예제: 신호 처리를 사용한 메시지 수신

MQGET 호출을 사용하여 신호를 설정하는 방법: 적합한 메시지가 큐에 도착했을 때 알림을 받도록 합니다. 신호는 다음을 통해서만 이용 가능합니다. IBM MQ for z/OS

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS.](#)

```

:
get_set_signal()

```

```

{
    MQMD      MsgDesc;
    MQGMO     GetMsgOpts;
    MQLONG    CompCode;
    MQLONG    Reason;
    MQHCONN   Hconn;
    MQHOBJ    Hobj;
    MQLONG    BufferLength;
    MQLONG    DataLength;
    char message_buffer[100];
    long int q_ecb, work_ecb;
    short int signal_sw, endloop;
    long int mask = 255;

    /*-----*/
    /* Set up GMO structure. */
    /*-----*/
    memset(&GetMsgOpts, '\0', sizeof(GetMsgOpts));
    memcpy(GetMsgOpts.StrucId, MQGMO_STRUC_ID,
           sizeof(GetMsgOpts.StrucId));
    GetMsgOpts.Version      = MQGMO_VERSION_1;
    GetMsgOpts.WaitInterval = 1000;
    GetMsgOpts.Options      = MQGMO_SET_SIGNAL +
                              MQGMO_BROWSE_FIRST;
    q_ecb                    = 0;
    GetMsgOpts.Signal1      = &q_ecb;
    /*-----*/
    /* Set up MD structure. */
    /*-----*/
    memset(&MsgDesc, '\0', sizeof(MsgDesc));
    memcpy(MsgDesc.StrucId, MQMD_STRUC_ID,
           sizeof(MsgDesc.StrucId));
    MsgDesc.Version      = MQMD_VERSION_1;
    MsgDesc.Report       = MQRO_NONE;
    memcpy(MsgDesc.MsgId, MQMI_NONE,
           sizeof(MsgDesc.MsgId));
    memcpy(MsgDesc.CorrelId, MQCI_NONE,
           sizeof(MsgDesc.CorrelId));

    /*-----*/
    /* Issue the MQGET call. */
    /*-----*/
    BufferLength = sizeof(message_buffer);
    signal_sw = 0;

    MQGET(Hconn, Hobj, &MsgDesc, &GetMsgOpts,
          BufferLength, message_buffer, &DataLength,
          &CompCode, &Reason);
    /*-----*/
    /* Check completion and reason codes. */
    /*-----*/
    switch (CompCode)
    {
        case (MQCC_OK):          /* Message retrieved */
            break;
        case (MQCC_WARNING):
            switch (Reason)
            {
                case (MQRC_SIGNAL_REQUEST_ACCEPTED):
                    signal_sw = 1;
                    break;
                default:
                    break; /* Perform error processing */
            }
            break;
        case (MQCC_FAILED):
            switch (Reason)
            {
                case (MQRC_Q_MGR_NOT_AVAILABLE):
                case (MQRC_CONNECTION_BROKEN):
                case (MQRC_Q_MGR_STOPPING):
                    break;
                default:
                    break; /* Perform error processing. */
            }
            break;
        default:
            break; /* Perform error processing. */
    }
    /*-----*/
    /* If the SET_SIGNAL was accepted, set up a loop to */

```

```

/* check whether a message has arrived at one second */
/* intervals. The loop ends if a message arrives or */
/* the wait interval specified in the MQGMO */
/* structure has expired. */
/* */
/* If a message arrives on the queue, another MQGET */
/* must be issued to retrieve the message. If other */
/* MQM calls have been made in the intervening */
/* period, this may necessitate reinitializing the */
/* MQMD and MQGMO structures. */
/* In this code, no intervening calls */
/* have been made, so the only change required to */
/* the structures is to specify MQGMO_NO_WAIT, */
/* since we now know the message is there. */
/* */
/* This code uses the EXEC CICS DELAY command to */
/* suspend the program for a second. A batch program */
/* may achieve the same effect by calling an */
/* assembler language subroutine which issues a */
/* z/OS STIMER macro. */
/*-----*/

```

```

if (signal_sw == 1)
{
    endloop = 0;
    do
    {
        EXEC CICS DELAY FOR HOURS(0) MINUTES(0) SECONDS(1);
        work_ecb = q_ecb & mask;
        switch (work_ecb)
        {
            case (MQEC_MSG_ARRIVED):
                endloop = 1;
                mqgmo_options = MQGMO_NO_WAIT;
                MQGET(Hconn, Hobj, &MsgDesc, &GetMsgOpts,
                    BufferLength, message_buffer,
                    &DataLength, &CompCode, &Reason);
                if (CompCode != MQCC_OK)
                    /* Perform error processing. */
                    break;
            case (MQEC_WAIT_INTERVAL_EXPIRED):
            case (MQEC_WAIT_CANCELED):
                endloop = 1;
                break;
            default:
                break;
        }
    } while (endloop == 0);
}
return;
}

```

C 예시: 객체의 속성에 대한 조회

이 예제는 MQINQ 호출을 사용하여 큐의 속성에 대해 조회하는 방법을 보여줍니다.

이 발췌문은 대기열 속성 샘플 애플리케이션에서 가져온 것입니다. **z/OS** 이것은 IBM MQ for z/OS 프로그램 CSQ4CCC1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

- ▶ **Multi** [다중 플랫폼용 샘플 절차형 프로그램 사용](#)
- ▶ **z/OS** [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#)

```

#include <cmqc.h>      /* MQ API header file      */
...
#define NUMBEROFSELECTORS 2

const MQHCONN Hconn = MQHC_DEF_HCONN;
...
static void InquireGetAndPut(char *Message,
                             PMQHOBJ pHobj,
                             char *Object)

```

```

{
/*      Declare local variables          */
/*                                          */
MQLONG SelectorCount = NUMBEROFSELECTORS;
/*      /* Number of selectors */
MQLONG IntAttrCount = NUMBEROFSELECTORS;
/*      /* Number of int attrs */
MQLONG CharAttrLength = 0;
/*      /* Length of char attribute buffer */
MQCHAR *CharAttrs ;
/*      /* Character attribute buffer */
MQLONG SelectorsTable[NUMBEROFSELECTORS];
/*      /* attribute selectors */
MQLONG IntAttrsTable[NUMBEROFSELECTORS];
/*      /* integer attributes */
MQLONG CompCode;
/*      /* Completion code */
MQLONG Reason;
/*      /* Qualifying reason */
/*                                          */
/*      Open the queue. If successful, do the inquire */
/*      call.                                          */
/*                                          */
/*      /*                                          */
/*      /* Initialize the variables for the inquire */
/*      /* call:                                          */
/*      /* - Set SelectorsTable to the attributes whose */
/*      /* status is                                          */
/*      /* required                                          */
/*      /* - All other variables are already set          */
/*      /*                                          */
SelectorsTable[0] = MQIA_INHIBIT_GET;
SelectorsTable[1] = MQIA_INHIBIT_PUT;
/*                                          */
/*      Issue the inquire call                          */
/*      Test the output of the inquire call. If the */
/*      call failed, display an error message          */
/*      showing the completion code and reason code, */
/*      otherwise display the status of the            */
/*      INHIBIT-GET and INHIBIT-PUT attributes          */
/*      /*                                          */
MQINQ(Hconn,
      *pHobj,
      SelectorCount,
      SelectorsTable,
      IntAttrCount,
      IntAttrsTable,
      CharAttrLength,
      CharAttrs,
      &CompCode,
      &Reason);
if (CompCode != MQCC_OK)
{
    sprintf(Message, MESSAGE_4_E,
            ERROR_IN_MQINQ, CompCode, Reason);
    SetMsg(Message);
}
else
{
    /* Process the changes */
} /* end if CompCode */

```

C 예제: 큐의 속성 설정

이 예제는 MQSET 호출을 사용하여 큐의 속성을 변경하는 방법을 보여줍니다.

이 발췌문은 대기열 속성 샘플 애플리케이션에서 가져온 것입니다. **z/OS** 이것은 IBM MQ for z/OS 프로그램 CSQ4CCC1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

- ▶ **Multi** 다중 플랫폼용 샘플 절차형 프로그램 사용
- ▶ **z/OS** 다음에 대한 예제 절차형 프로그램을 사용하여 z/OS

```

#include <mqc.h>      /* MQ API header file      */
:
#define NUMBEROFSELECTORS 2

```



```

const MQHCONN Hconn = MQHC_DEF_HCONN;

static void InhibitGetAndPut(char *Message,
                           PMQHOBj pHobj,
                           char *Object)
{
    /*
    /* Declare local variables
    /*
    /*
    MQLONG SelectorCount = NUMBEROFSELECTORS;
    /* Number of selectors
    MQLONG IntAttrCount = NUMBEROFSELECTORS;
    /* Number of int attrs
    MQLONG CharAttrLength = 0;
    /* Length of char attribute buffer
    MQCHAR *CharAttrs ;
    /* Character attribute buffer
    MQLONG SelectorsTable[NUMBEROFSELECTORS];
    /* attribute selectors
    MQLONG IntAttrsTable[NUMBEROFSELECTORS];
    /* integer attributes
    MQLONG CompCode;
    /* Completion code
    MQLONG Reason;
    /* Qualifying reason
    :
    /*
    /* Open the queue. If successful, do the
    /* inquire call.
    /*
    /*
    :
    /*
    /* Initialize the variables for the set call:
    /* - Set SelectorsTable to the attributes to be
    /* set
    /* - Set IntAttrsTable to the required status
    /* - All other variables are already set
    /*
    /*
    SelectorsTable[0] = MQIA_INHIBIT_GET;
    SelectorsTable[1] = MQIA_INHIBIT_PUT;
    IntAttrsTable[0] = MQQA_GET_INHIBITED;
    IntAttrsTable[1] = MQQA_PUT_INHIBITED;
    :

```

```

    /*
    /* Issue the set call.
    /* Test the output of the set call. If the
    /* call fails, display an error message
    /* showing the completion code and reason
    /* code; otherwise move INHIBITED to the
    /* relevant screen map fields
    /*
    /*
    MQSET(Hconn,
          *pHobj,
          SelectorCount,
          SelectorsTable,
          IntAttrCount,
          IntAttrsTable,
          CharAttrLength,
          CharAttrs,
          &CompCode,
          &Reason);
    if (CompCode != MQCC_OK)
    {
        sprintf(Message, MESSAGE_4_E,
                ERROR_IN_MQSET, CompCode, Reason);
        SetMsg(Message);
    }
    else
    {
        /* Process the changes */
    }
    /* end if CompCode */

```

C 예제: MQSTAT을 사용한 상태 정보 조회

이 예제는 비동기 MQPUT를 실행하고 MQSTAT으로 상태 정보를 검색하는 방법을 예시합니다.

이 발췌문은 Calling MQSTAT 샘플 애플리케이션에서 가져온 것입니다. **z/OS** 이것은 IBM MQ for z/OS 프로그램 amqsapt0 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

- ▶ **Multi** 다중 플랫폼용 샘플 절차형 프로그램 사용
- ▶ **z/OS** 다음에 대한 예제 절차형 프로그램을 사용하여 z/OS

```

/*****
/*
/* Program name: AMQSAPT0
/*
/*
/* Description: Sample C program that asynchronously puts messages
/*               to a message queue (example using MQPUT & MQSTAT).
/*
/*
/* Licensed Materials - Property of IBM
/*
/* 63H9336
/* (c) Copyright IBM Corp. 2006, 2026. All Rights Reserved.
/*
/* US Government Users Restricted Rights - Use, duplication or
/* disclosure restricted by GSA ADP Schedule Contract with
/* IBM Corp.
/*
*****/
/*
/* Function:
/*
/* AMQSAPT0 is a sample C program to put messages on a message
/* queue with asynchronous response option, querying the success
/* of the put operations with MQSTAT.
/*
/* -- messages are sent to the queue named by the parameter
/*
/* -- gets lines from StdIn, and adds each to target
/* queue, taking each line of text as the content
/* of a datagram message; the sample stops when a null
/* line (or EOF) is read.
/* New-line characters are removed.
/* If a line is longer than 99 characters it is broken up
/* into 99-character pieces. Each piece becomes the
/* content of a datagram message.
/* If the length of a line is a multiple of 99 plus 1, for
/* example, 199, the last piece will only contain a
/* new-line character so will terminate the input.
/*
/* -- writes a message for each MQI reason other than
/* MQRC_NONE; stops if there is a MQI completion code
/* of MQCC_FAILED
/*
/* -- summarizes the overall success of the put operations
/* through a call to MQSTAT to query MQSTAT_TYPE_ASYNC_ERROR*/
/*
/* Program logic:
/* MQOPEN target queue for OUTPUT
/* while end of input file not reached,
/* . read next line of text
/* . MQPUT datagram message with text line as data
/* MQCLOSE target queue
/* MQSTAT connection
/*
*****/
/*
/* AMQSAPT0 has the following parameters
/* required:
/* (1) The name of the target queue
/* optional:
/* (2) Queue manager name
/* (3) The open options
/* (4) The close options
/* (5) The name of the target queue manager
/* (6) The name of the dynamic queue
/*
*****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

```

    /* includes for MQI */
#include <cmqc.h>

int main(int argc, char **argv)
{
    /* Declare file and character for sample input */
    FILE *fp;

    /* Declare MQI structures needed */
    MQOD      od = {MQOD_DEFAULT}; /* Object Descriptor */
    MQMD      md = {MQMD_DEFAULT}; /* Message Descriptor */
    MQPMO     pmo = {MQPMO_DEFAULT}; /* put message options */
    MQSTS     sts = {MQSTS_DEFAULT}; /* status information */
    /** note, sample uses defaults where it can */
    MQHCONN   Hcon; /* connection handle */
    MQHOBJ    Hobj; /* object handle */
    MQLONG    O_options; /* MQOPEN options */
    MQLONG    C_options; /* MQCLOSE options */
    MQLONG    CompCode; /* completion code */
    MQLONG    OpenCode; /* MQOPEN completion code */
    MQLONG    Reason; /* reason code */
    MQLONG    CReason; /* reason code for MQCONN */
    MQLONG    messlen; /* message length */
    char      buffer[100]; /* message buffer */
    char      QMName[50]; /* queue manager name */

    printf("Sample AMQSAPT0 start\n");
    if (argc < 2)
    {
        printf("Required parameter missing - queue name\n");
        exit(99);
    }

    /*****
    /*
    /* Connect to queue manager
    /*
    /*****
    QMName[0] = 0; /* default */
    if (argc > 2)
        strcpy(QMName, argv[2]);
    MQCONN(QMName, /* queue manager
            &Hcon, /* connection handle
            &Compcode, /* completion code
            &Reason); /* reason code
    /* report reason and stop if it failed */
    if (CompCode == MQCC_FAILED)
    {
        printf("MQCONN ended with reason code %d\n", CReason);
        exit( (int)CReason );
    }

    /*****
    /*
    /* Use parameter as the name of the target queue
    /*
    /*****
    strncpy(od.ObjectName, argv[1], (size_t)MQ_Q_NAME_LENGTH);
    printf("target queue is %s\n", od.ObjectName);

    if (argc > 5)
    {
        strncpy(od.ObjectQMGrName, argv[5], (size_t) MQ_Q_MGR_NAME_LENGTH);
        printf("target queue manager is %s\n", od.ObjectQMGrName);
    }

    if (argc > 6)
    {
        strncpy(od.DynamicQName, argv[6], (size_t) MQ_Q_NAME_LENGTH);
        printf("dynamic queue name is %s\n", od.DynamicQName);
    }

    /*****
    /*
    /* Open the target message queue for output
    /*
    /*****
    if (argc > 3)
    {
        O_options = atoi( argv[3] );
        printf("open options are %d\n", O_options);
    }

```

```

else
{
    O_options = MQOO_OUTPUT          /* open queue for output      */
                | MQOO_FAIL_IF_QUIESCING /* but not if MQM stopping */
                ;                    /* = 0x2010 = 8208 decimal */
}

MQOPEN(Hcon,                      /* connection handle      */
        &od,                      /* object descriptor for queue */
        O_options,                /* open options           */
        &Hobj,                   /* object handle          */
        &OpenCode,               /* MQOPEN completion code */
        &Reason);               /* reason code            */

/* report reason, if any; stop if failed */
if (Reason != MQRC_NONE)
{
    printf("MQOPEN ended with reason code %d\n", Reason);
}

if (OpenCode == MQCC_FAILED)
{
    printf("unable to open queue for output\n");
}

/*****
/*
/* Read lines from the file and put them to the message queue */
/* Loop until null line or end of file, or there is a failure */
/*
*****/
CompCode = OpenCode; /* use MQOPEN result for initial test */
fp = stdin;

memcpy(md.Format, /* character string format */
        MQFMT_STRING, (size_t)MQ_FORMAT_LENGTH);

/*****
/* These options specify that put operation should occur */
/* asynchronously and the application will check the success */
/* using MQSTAT at a later time. */
*****/
md.Persistence = MQPER_NOT_PERSISTENT;
pmo.Options |= MQPMO_ASYNC_RESPONSE;

/*****
/* These options cause the MsgId and CorrelId to be replaced, so */
/* that there is no need to reset them before each MQPUT */
*****/
pmo.Options |= MQPMO_NEW_MSG_ID;
pmo.Options |= MQPMO_NEW_CORREL_ID;

while (CompCode != MQCC_FAILED)
{
    if (fgets(buffer, sizeof(buffer), fp) != NULL)
    {
        messlen = (MQLONG)strlen(buffer); /* length without null */
        if (buffer[messlen-1] == '\n') /* last char is a new-line */
        {
            buffer[messlen-1] = '\0'; /* replace new-line with null */
            --messlen; /* reduce buffer length */
        }
    }
    else messlen = 0; /* treat EOF same as null line */

    /*****
    /*
    /* Put each buffer to the message queue */
    /*
    *****/
    if (messlen > 0)
    {
        MQPUT(Hcon, /* connection handle */
              Hobj, /* object handle */
              &md, /* message descriptor */
              &pmo, /* default options (datagram) */
              messlen, /* message length */
              buffer, /* message buffer */
              &CompCode, /* completion code */
              &Reason); /* reason code */

        /* report reason, if any */
    }
}

```

```

        if (Reason != MQRC_NONE)
        {
            printf("MQPUT ended with reason code %d\n", Reason);
        }
    }
    else /* satisfy end condition when empty line is read */
        CompCode = MQCC_FAILED;
}

/*****
/*
/*   Close the target queue (if it was opened)
/*
/*
/* *****/
if (OpenCode != MQCC_FAILED)
{
    if (argc > 4)
    {
        C_options = atoi( argv[4] );
        printf("close options are %d\n", C_options);
    }
    else
    {
        C_options = MQCO_NONE;          /* no close options          */
    }

    MQCLOSE(Hcon,                      /* connection handle          */
            &Hobj,                     /* object handle              */
            C_options,
            &CompCode,                 /* completion code            */
            &Reason);                  /* reason code                 */

    /* report reason, if any          */
    if (Reason != MQRC_NONE)
    {
        printf("MQCLOSE ended with reason code %d\n", Reason);
    }
}

/*****
/*
/*   Query how many asynchronous puts succeeded
/*
/*
/* *****/
MQSTAT(&Hcon,                        /* connection handle          */
       MQSTAT_TYPE_ASYNC_ERROR,     /* status type                */
       &Sts,                         /* MQSTS structure            */
       &CompCode,                    /* completion code            */
       &Reason);                     /* reason code                 */

/* report reason, if any          */
if (Reason != MQRC_NONE)
{
    printf("MQSTAT ended with reason code %d\n", Reason);
}
else
{
    /* Display results */
    printf("Succeeded putting %d messages\n",
           sts.PutSuccessCount);
    printf("%d messages were put with a warning\n",
           sts.PutWarningCount);
    printf("Failed to put %d messages\n",
           sts.PutFailureCount);

    if(sts.CompCode == MQCC_WARNING)
    {
        printf("The first warning that occurred had reason code %d\n",
               sts.Reason);
    }
    else if(sts.CompCode == MQCC_FAILED)
    {
        printf("The first error that occurred had reason code %d\n",
               sts.Reason);
    }
}

/*****
/*
/*   Disconnect from MQM if not already connected
/*
/*
/* *****/

```

```

if (CReason != MQRC_ALREADY_CONNECTED)
{
    MQDISC(&Hcon,                /* connection handle */
          &CompCode,            /* completion code */
          &Reason);             /* reason code */

    /* report reason, if any */
    if (Reason != MQRC_NONE)
    {
        printf("MQDISC ended with reason code %d\n", Reason);
    }
}

/*****
/*
/* END OF AMQSAPT0
/*
/*
*****/
printf("Sample AMQSAPT0 end\n");
return(0);
}

```

COBOL 에서 MQI 앱 개발하기

COBOL로 IBM MQ 프로그램을 코딩하는 경우 다음 섹션의 정보를 참고하십시오.

이름 지정된 상수

상수의 이름은 이름의 일부로 밑줄 문자(_)를 포함하는 것으로 표시됩니다. COBOL에서는 밑줄 대신에 하이픈 문자(-)를 사용해야 합니다. 문자열 값이 있는 상수는 문자열 구분 기호로 작은따옴표(')를 사용합니다. 컴파일러에서 이 문자를 허용하게 하려면, 컴파일러 옵션 APOST를 사용하십시오.

복사 파일 CMQV는 레벨-10 항목으로 이름 지정된 상수의 선언을 포함합니다. 상수를 사용하려면, 명시적으로 레벨-01 항목을 선언한 후 상수의 선언에서 복사할 COPY 명령문을 사용하십시오.

```

WORKING-STORAGE SECTION.
01 MQM-CONSTANTS.
COPY CMQV.

```

그러나 이 방법은 상수를 참조하지 않더라도 프로그램에서 스토리지를 차지하게 합니다. 상수가 동일한 실행 단위 내에서 여러 개의 별도 프로그램에 포함되는 경우 상수의 여러 사본이 존재하게 됩니다. 이는 결과적으로 상당한 양의 메인 스토리지가 사용될 수 있습니다. GLOBAL 절을 레벨-01 선언에 추가하여 이를 방지할 수 있습니다.

```

* Declare a global structure to hold the constants
01 MQM-CONSTANTS GLOBAL.
COPY CMQV.

```

이는 실행 단위 내에서 단 하나의 상수 세트에 스토리지를 할당합니다. 그러나 상수를 실행 단위 내에서 레벨-01 선언을 포함하는 프로그램만이 아닌 모든 프로그램에서 참조할 수 있습니다.

구조 정렬 확인

MQI 호출로 전달하기 전에 구조체를 IBM MQ 단어 경계에 맞춰 정렬하십시오. 단어 경계는 32비트 프로세스의 경우 4바이트, 64비트 프로세스의 경우 8바이트이며 128비트 프로세스의 경우 16바이트입니다(IBM i).

가능한 경우 모두 경계에 맞춰지도록 모든 IBM MQ 구조를 함께 넣으십시오.

관련 참조

972 페이지의 『C 및 COBOL MQI 샘플에 표시된 기능 AIX and Linux』
 샘플 프로그램에서 시연한 기술은,에 IBM MQ 관한 AIX and Linux것입니다.

975 페이지의 『C 및 COBOL MQI 샘플에 표시된 기능 Windows』
 샘플 프로그램에서 시연한 기술은,에 IBM MQ 관한 Windows것입니다.

977 페이지의 『C, COBOL 및 RPG MQI 샘플에 표시된 기능은 다음과 같습니다. IBM i』
 샘플 프로그램에서 시연한 기법을 IBM MQIBM i 보여줍니다.

1080 페이지의 『MQI 샘플에 표시된 기능은 다음과 같습니다. z/OS』

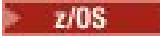
각 샘플 애플리케이션에서 시연된 ‘MQI’ 기능에 대한 요약으로, 각 샘플이 작성된 프로그래밍 언어와 실행 환경이 포함됩니다.

COBOL 예시: 큐 매니저에 연결하기

이 예에서는 MQCONN 호출을 사용하여 프로그램을 z/OS 일괄처리의 큐 관리자에 연결하는 방법을 보여줍니다.

이 발췌문은 Browse 샘플 애플리케이션에서 가져온 것입니다.  이것은 IBM MQ for z/OS 프로그램 CSQ4BVA1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

-  [다중 플랫폼용 샘플 절차형 프로그램 사용](#)
-  [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#)

```
* -----*
WORKING-STORAGE SECTION.
* -----*
*   W02 - Data fields derived from the PARM field
01  W02-MQM                PIC X(48) VALUE SPACES.
*   W03 - MQM API fields
01  W03-HCONN              PIC S9(9) BINARY.
01  W03-COMPCODE           PIC S9(9) BINARY.
01  W03-REASON             PIC S9(9) BINARY.
*
*   MQV contains constants (for filling in the control
*   blocks)
*   and return codes (for testing the result of a call)
*
01  W05-MQM-CONSTANTS.
    COPY CMQV SUPPRESS.
    :
*   Separate into the relevant fields any data passed
*   in the PARM statement
*
    UNSTRING PARM-STRING DELIMITED BY ALL ','
                INTO W02-MQM
                W02-OBJECT.
    :
*   Connect to the specified queue manager.
*
    CALL 'MQCONN' USING W02-MQM
                        W03-HCONN
                        W03-COMPCODE
                        W03-REASON.
*
*   Test the output of the connect call.  If the call
*   fails, print an error message showing the
*   completion code and reason code.
*
    IF (W03-COMPCODE NOT = MQCC-OK) THEN
        :
        END-IF.
    :
```

COBOL 예시: 큐 매니저에서 연결 해제

이 예에서는 MQDISC 호출을 사용하여 z/OS 일괄처리의 큐 관리자에서 프로그램의 연결을 끊는 방법을 보여줍니다.

이 코드 추출에서 사용된 변수는 1163 페이지의 『COBOL 예시: 큐 매니저에 연결하기』에서 설정된 변수입니다. 이 발췌문은 Browse 샘플 애플리케이션에서 가져온 것입니다.  이것은 IBM MQ for z/OS 프로그램 CSQ4BVA1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

-  [다중 플랫폼용 샘플 절차형 프로그램 사용](#)

- ▶ **z/OS** 다음에 대한 예제 절차형 프로그램을 사용하여 z/OS

```

:
*
* Disconnect from the queue manager
*
    CALL 'MQDISC' USING W03-HCONN
                        W03-COMPCODE
                        W03-REASON.
*
* Test the output of the disconnect call. If the
* call fails, print an error message showing the
* completion code and reason code.
*
    IF (W03-COMPCODE NOT = MQCC-OK) THEN
:
        END-IF.
:

```

COBOL 예시: 동적 큐 생성

이 예제는 MQOPEN 호출을 사용하여 동적 큐를 작성하는 방법을 보여줍니다.

이 발췌문은 신용 조회 샘플 신청서에서 가져온 것입니다. ▶ **z/OS** 이것은 IBM MQ for z/OS 프로그램 CSQ4CVB1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

- ▶ **Multi** 다중 플랫폼용 샘플 절차형 프로그램 사용
- ▶ **z/OS** 다음에 대한 예제 절차형 프로그램을 사용하여 z/OS

```

:
* -----*
WORKING-STORAGE SECTION.
* -----*
*
* W02 - Queues processed in this program
*
01 W02-MODEL-QNAME PIC X(48) VALUE
   'CSQ4SAMP.B1.MODEL'
01 W02-NAME-PREFIX PIC X(48) VALUE
   'CSQ4SAMP.B1.*'
01 W02-TEMPORARY-Q PIC X(48).
*
* W03 - MQM API fields
*
01 W03-HCONN PIC S9(9) BINARY VALUE ZERO.
01 W03-OPTIONS PIC S9(9) BINARY.
01 W03-HOBBJ PIC S9(9) BINARY.
01 W03-COMPCODE PIC S9(9) BINARY.
01 W03-REASON PIC S9(9) BINARY.
*
* API control blocks
*
01 MQM-OBJECT-DESCRIPTOR.
   COPY CMQODV.
*
* CMQV contains constants (for setting or testing
* field values) and return codes (for testing the
* result of a call)
*
01 MQM-CONSTANTS.
   COPY CMQV SUPPRESS.
* -----*
PROCEDURE DIVISION.
* -----*
:
* -----*
OPEN-TEMP-RESPONSE-QUEUE SECTION.
* -----*

```

```

*
* This section creates a temporary dynamic queue

```

```

* using a model queue
*
* -----*
*
* Change three fields in the Object Descriptor (MQOD)
* control block. (MQODV initializes the other fields)
*
      MOVE MQOT-Q          TO MQOD-OBJECTTYPE.
      MOVE W02-MODEL-QNAME TO MQOD-OBJECTNAME.
      MOVE W02-NAME-PREFIX TO MQOD-DYNAMICQNAME.
*
      COMPUTE W03-OPTIONS = MQ00-INPUT-EXCLUSIVE.
*
      CALL 'MQOPEN' USING W03-HCONN
                        MQOD
                        W03-OPTIONS
                        W03-HOBJ-MODEL
                        W03-COMPCODE
                        W03-REASON.
*
      IF W03-COMPCODE NOT = MQCC-OK
          MOVE 'MQOPEN'      TO M01-MSG4-OPERATION
          MOVE W03-COMPCODE TO M01-MSG4-COMPCODE
          MOVE W03-REASON   TO M01-MSG4-REASON
          MOVE M01-MESSAGE-4 TO M00-MESSAGE
      ELSE
          MOVE MQOD-OBJECTNAME TO W02-TEMPORARY-Q
      END-IF.
*
      OPEN-TEMP-RESPONSE-QUEUE-EXIT.
*
      Return to performing section.
*
      EXIT.
      EJECT
*

```

COBOL 예시: 기존 큐 열기

이 예제는 MQOPEN 호출을 사용하여 기존 큐를 여는 방법을 보여줍니다.

이 발췌문은 Browse 샘플 애플리케이션에서 가져온 것입니다.  이것은 IBM MQ for z/OS 프로그램 CSQ4BVA1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

-  [다중 플랫폼용 샘플 절차형 프로그램 사용](#)
-  [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#)

```

:
* -----*
* WORKING-STORAGE SECTION.
* -----*
*
* W01 - Fields derived from the command area input
*
01 W01-OBJECT          PIC X(48).
*
* W02 - MQM API fields
*
01 W02-HCONN          PIC S9(9) BINARY VALUE ZERO.
01 W02-OPTIONS        PIC S9(9) BINARY.
01 W02-HOBJ           PIC S9(9) BINARY.
01 W02-COMPCODE        PIC S9(9) BINARY.
01 W02-REASON         PIC S9(9) BINARY.
*
* CMQODV defines the object descriptor (MQOD)
*
01 MQM-OBJECT-DESCRIPTOR.
   COPY CMQODV.
*
* CMQV contains constants (for setting or testing
* field values) and return codes (for testing the
* result of a call)
*
01 MQM-CONSTANTS.

```

```

COPY CMQV SUPPRESS.
* -----*
E-OPEN-QUEUE SECTION.
* -----*
*
* This section opens the queue
*
* Initialize the Object Descriptor (MQOD) control
* block
* (The copy file initializes the remaining fields.)
*
MOVE MQOT-Q          TO MQOD-OBJECTTYPE.
MOVE W01-OBJECT      TO MQOD-OBJECTNAME.
*
* Initialize W02-OPTIONS to open the queue for both
* inquiring about and setting attributes
*
COMPUTE W02-OPTIONS = MQ00-INQUIRE + MQ00-SET.

*
* Open the queue
*
CALL 'MQOPEN' USING W02-HCONN
                   MQOD
                   W02-OPTIONS
                   W02-HOBJ
                   W02-COMPCODE
                   W02-REASON.
*
* Test the output from the open
*
* If the completion code is not OK, display a
* separate error message for each of the following
* errors:
*
* Q-MGR-NOT-AVAILABLE - MQM is not available
* CONNECTION-BROKEN  - MQM is no longer connected to CICS
* UNKNOWN-OBJECT-NAME - The queue does not exist
* NOT-AUTHORIZED     - The user is not authorized to open
*                     the queue
*
* For any other error, display an error message
* showing the completion and reason codes
*
IF W02-COMPCODE NOT = MQCC-OK
EVALUATE TRUE
*
    WHEN W02-REASON = MQRC-Q-MGR-NOT-AVAILABLE
        MOVE M01-MESSAGE-6 TO M00-MESSAGE
*
    WHEN W02-REASON = MQRC-CONNECTION-BROKEN
        MOVE M01-MESSAGE-6 TO M00-MESSAGE
*
    WHEN W02-REASON = MQRC-UNKNOWN-OBJECT-NAME
        MOVE M01-MESSAGE-2 TO M00-MESSAGE
*
    WHEN W02-REASON = MQRC-NOT-AUTHORIZED
        MOVE M01-MESSAGE-3 TO M00-MESSAGE
*
    WHEN OTHER
        MOVE 'MQOPEN'      TO M01-MSG4-OPERATION
        MOVE W02-COMPCODE  TO M01-MSG4-COMPCODE
        MOVE W02-REASON    TO M01-MSG4-REASON
        MOVE M01-MESSAGE-4 TO M00-MESSAGE
    END-EVALUATE
END-IF.
E-EXIT.
*
* Return to performing section
*
EXIT.
EJECT

```

COBOL 예시: 큐 닫기

이 예제는 MQCLOSE 호출을 사용하는 방법을 보여줍니다.

이 코드 추출에서 사용된 변수는 1163 페이지의 『COBOL 예시: 큐 매니저에 연결하기』에서 설정된 변수입니다. 이 발췌문은 Browse 샘플 애플리케이션에서 가져온 것입니다. **z/OS** 이것은 IBM MQ for z/OS 프로그램 CSQ4BVA1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

- ▶ **Multi** 다중 플랫폼용 샘플 절차형 프로그램 사용
- ▶ **z/OS** 다음에 대한 예제 절차형 프로그램을 사용하여 z/OS

```

:
*
*   Close the queue
*
*   MOVE MQCO-NONE TO W03-OPTIONS.
*
*   CALL 'MQCLOSE' USING W03-HCONN
                        W03-HOBJ
                        W03-OPTIONS
                        W03-COMPCODE
                        W03-REASON.
*
*   Test the output of the MQCLOSE call.  If the call
*   fails, print an error message showing the
*   completion code and reason code.
*
*   IF (W03-COMPCODE NOT = MQCC-OK) THEN
*       MOVE 'CLOSE'          TO W04-MSG4-TYPE
*       MOVE W03-COMPCODE     TO W04-MSG4-COMPCODE
*       MOVE W03-REASON       TO W04-MSG4-REASON
*       MOVE W04-MESSAGE-4    TO W00-PRINT-DATA
*       PERFORM PRINT-LINE
*       MOVE W06-CSQ4-ERROR TO W00-RETURN-CODE
*   END-IF.
*

```

COBOL 예시: MQPUT을 사용하여 메시지 전송

이 예제는 컨텍스트를 사용하여 MQPUT 호출을 사용하는 방법을 보여줍니다.

이 발췌문은 신용 조회 샘플 신청서에서 가져온 것입니다. **z/OS** 이것은 IBM MQ for z/OS 프로그램 CSQ4CVB1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

- ▶ **Multi** 다중 플랫폼용 샘플 절차형 프로그램 사용
- ▶ **z/OS** 다음에 대한 예제 절차형 프로그램을 사용하여 z/OS

```

:
* -----*
*   WORKING-STORAGE SECTION.
* -----*
*
*   W02 - Queues processed in this program
*
*   01 W02-TEMPORARY-Q          PIC X(48).
*
*   W03 - MQM API fields
*
*   01 W03-HCONN                PIC S9(9) BINARY VALUE ZERO.
*   01 W03-HOBJ-INQUIRY         PIC S9(9) BINARY.
*   01 W03-OPTIONS              PIC S9(9) BINARY.
*   01 W03-BUFFLEN              PIC S9(9) BINARY.
*   01 W03-COMPCODE             PIC S9(9) BINARY.
*   01 W03-REASON               PIC S9(9) BINARY.
*
*   01 W03-PUT-BUFFER.
*
*   05 W03-CSQ4BIIM.
*   COPY CSQ4VB1.
*
*   API control blocks
*

```

```

01 MQM-MESSAGE-DESCRIPTOR.
   COPY CMQMDV.
01 MQM-PUT-MESSAGE-OPTIONS.
   COPY CMQPMOV.
*
*   MQV contains constants (for filling in the
*   control blocks) and return codes (for testing
*   the result of a call).
*
01 MQM-CONSTANTS.
   COPY CMQV SUPPRESS.
* -----*
PROCEDURE DIVISION.
* -----*
:
*   Open queue and build message.
:

```

```

*
* Set the message descriptor and put-message options to
* the values required to create the message.
* Set the length of the message.
*
MOVE MQMT-REQUEST      TO MQMD-MSGTYPE.
MOVE MQCI-NONE         TO MQMD-CORRELID.
MOVE MQMI-NONE         TO MQMD-MSGID.
MOVE W02-TEMPORARY-Q   TO MQMD-REPLYTOQ.
MOVE SPACES            TO MQMD-REPLYTOQMGR.
MOVE 5                 TO MQMD-PRIORITY.
MOVE MQPER-NOT-PERSISTENT TO MQMD-PERSISTENCE.
COMPUTE MQPMO-OPTIONS  = MQPMO-NO-SYNCPOINT +
                        MQPMO-DEFAULT-CONTEXT.
MOVE LENGTH OF CSQ4BIIM-MSG TO W03-BUFFLEN.
*
CALL 'MQPUT' USING W03-HCONN
                  W03-HOBJ-INQUIRY
                  MQMD
                  MQPMO
                  W03-BUFFLEN
                  W03-PUT-BUFFER
                  W03-COMPCODE
                  W03-REASON.
IF W03-COMPCODE NOT = MQCC-OK
:
END-IF.

```

COBOL 예시: MQPUT1 를 사용하여 메시지 게시하기

이 예제는 MQPUT1 호출을 사용하는 방법을 보여줍니다.

이 발췌문은 신용 조회 샘플 신청서에서 가져온 것입니다.  이것은 IBM MQ for z/OS 프로그램 CSQ4CVB5 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

-  [다중 플랫폼용 샘플 절차형 프로그램 사용](#)
-  [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#)

```

:
* -----*
WORKING-STORAGE SECTION.
* -----*
*
*   W03 - MQM API fields
*
01 W03-HCONN          PIC S9(9) BINARY VALUE ZERO.
01 W03-OPTIONS        PIC S9(9) BINARY.
01 W03-COMPCODE       PIC S9(9) BINARY.
01 W03-REASON         PIC S9(9) BINARY.
01 W03-BUFFLEN        PIC S9(9) BINARY.
*
01 W03-PUT-BUFFER.
   05 W03-CSQ4BQRM.
   COPY CSQ4VB4.

```



```

*
*   API control blocks
*
*01 MQM-OBJECT-DESCRIPTOR.
*   COPY CMQODV.
*01 MQM-MESSAGE-DESCRIPTOR.
*   COPY CMQMDV.
*01 MQM-PUT-MESSAGE-OPTIONS.
*   COPY CMQPMOV.
*
* CMQV contains constants (for filling in the
* control blocks) and return codes (for testing
* the result of a call).
*
*01 MQM-MQV.
*   COPY CMQV SUPPRESS.
*-----*
*   PROCEDURE DIVISION.
*-----*
*
*       Get the request message.
*
*-----*
*   PROCESS-QUERY SECTION.
*-----*
*
*       Build the reply message.
*
*
*   Set the object descriptor, message descriptor and
*   put-message options to the values required to create
*   the message.
*   Set the length of the message.
*
*   MOVE MQMD-REPLYTOQ      TO MQOD-OBJECTNAME.
*   MOVE MQMD-REPLYTOQMGR   TO MQOD-OBJECTQMGRNAME.
*   MOVE MQMT-REPLY         TO MQMD-MSGTYPE.
*   MOVE SPACES             TO MQMD-REPLYTOQ.
*   MOVE SPACES             TO MQMD-REPLYTOQMGR.
*   MOVE LOW-VALUES         TO MQMD-MSGID.
*   COMPUTE MQPMO-OPTIONS = MQPMO-SYNCPOINT +
*                           MQPMO-PASS-IDENTITY-CONTEXT.
*   MOVE W03-HOBJ-CHECKQ    TO MQPMO-CONTEXT.
*   MOVE LENGTH OF CSQ4BQRM-MSG TO W03-BUFFLEN.
*
*   CALL 'MQPUT1' USING W03-HCONN
*                       MQOD
*                       MQMD
*                       MQPMO
*                       W03-BUFFLEN
*                       W03-PUT-BUFFER
*                       W03-COMPCODE
*                       W03-REASON.
*   IF W03-COMPCODE NOT = MQCC-OK
*       MOVE 'MQPUT1'      TO M02-OPERATION
*       MOVE MQOD-OBJECTNAME TO M02-OBJECTNAME
*       PERFORM RECORD-CALL-ERROR
*       PERFORM FORWARD-MSG-TO-DLQ
*   END-IF.
*

```

COBOL 예시: 메시지 받기

이 예제는 MQGET 호출을 사용하여 큐에서 메시지를 제거하는 방법을 보여줍니다.

이 발췌문은 신용 조회 샘플 신청서에서 가져온 것입니다.  이것은 IBM MQ for z/OS 프로그램 CSQ4CVB1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

-  다중 플랫폼용 샘플 절차형 프로그램 사용
-  다음에 대한 예제 절차형 프로그램을 사용하여 z/OS

```

:
*-----*

```

```

WORKING-STORAGE SECTION.
* -----*
*
*   W03 - MQM API fields
*
01  W03-HCONN          PIC S9(9) BINARY VALUE ZERO.
01  W03-HOBJ-RESPONSE  PIC S9(9) BINARY.
01  W03-OPTIONS        PIC S9(9) BINARY.
01  W03-BUFFLEN        PIC S9(9) BINARY.
01  W03-DATALEN        PIC S9(9) BINARY.
01  W03-COMPCODE       PIC S9(9) BINARY.
01  W03-REASON         PIC S9(9) BINARY.
*
01  W03-GET-BUFFER.
    05 W03-CSQ4BAM.
    COPY CSQ4VB2.
*
*   API control blocks
*
01  MQM-MESSAGE-DESCRIPTOR.
    COPY CMQMDV.
01  MQM-GET-MESSAGE-OPTIONS.
    COPY CMQGMV.
*
*   MQV contains constants (for filling in the
*   control blocks) and return codes (for testing
*   the result of a call).
*
01  MQM-CONSTANTS.
    COPY CMQV SUPPRESS.
* -----*
A-MAIN SECTION.
* -----*
:
*   Open response queue.
:
* -----*
PROCESS-RESPONSE-SCREEN SECTION.
* -----*
*
*   This section gets a message from the response queue.
*
*   When a correct response is received, it is
*   transferred to the map for display; otherwise
*   an error message is built.
*
* -----*

```

```

*
*   Set get-message options
*
    COMPUTE MQGMO-OPTIONS = MQGMO-SYNCPOINT +
                          MQGMO-ACCEPT-TRUNCATED-MSG +
                          MQGMO-NO-WAIT.
*
* Set msgid and correlid in MQMD to nulls so that any
* message will qualify.
* Set length to available buffer length.
*
    MOVE MQMI-NONE TO MQMD-MSGID.
    MOVE MQCI-NONE TO MQMD-CORRELID.
    MOVE LENGTH OF W03-GET-BUFFER TO W03-BUFFLEN.
*
    CALL 'MQGET' USING W03-HCONN
                      W03-HOBJ-RESPONSE
                      MQMD
                      MQGMO
                      W03-BUFFLEN
                      W03-GET-BUFFER
                      W03-DATALEN
                      W03-COMPCODE
                      W03-REASON.
    EVALUATE TRUE
      WHEN W03-COMPCODE NOT = MQCC-FAILED
      :
*       Process the message
      :
      WHEN (W03-COMPCODE = MQCC-FAILED AND
            W03-REASON = MQRC-NO-MSG-AVAILABLE)
        MOVE M01-MESSAGE-9 TO M00-MESSAGE

```

```

                PERFORM CLEAR-RESPONSE-SCREEN
*
        WHEN OTHER
            MOVE 'MQGET '      TO M01-MSG4-OPERATION
            MOVE W03-COMPCODE TO M01-MSG4-COMPCODE
            MOVE W03-REASON   TO M01-MSG4-REASON
            MOVE M01-MESSAGE-4 TO M00-MESSAGE
            PERFORM CLEAR-RESPONSE-SCREEN
        END-EVALUATE.

```

COBOL 예시: 대기 옵션을 사용하여 메시지 수신

이 예제는 대기 옵션으로 MQGET 호출을 사용하고 잘린 메시지를 받는 방법을 예시합니다.

이 발췌문은 신용 조회 샘플 신청서에서 가져온 것입니다.  이것은 IBM MQ for z/OS 프로그램 CSQ4CVB5 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

-  다중 플랫폼용 샘플 절차형 프로그램 사용
-  다음에 대한 예제 절차형 프로그램을 사용하여 z/OS

```

:
* -----*
* WORKING-STORAGE SECTION.
* -----*
*
* W00 - General work fields
*
* 01 W00-WAIT-INTERVAL   PIC S9(09) BINARY VALUE 30000.
*
* W03 - MQM API fields
*
* 01 W03-HCONN           PIC S9(9) BINARY VALUE ZERO.
* 01 W03-OPTIONS         PIC S9(9) BINARY.
* 01 W03-HOBJ-CHECKQ     PIC S9(9) BINARY.
* 01 W03-COMPCODE        PIC S9(9) BINARY.
* 01 W03-REASON          PIC S9(9) BINARY.
* 01 W03-DATALEN         PIC S9(9) BINARY.
* 01 W03-BUFFLEN         PIC S9(9) BINARY.
*
* 01 W03-MSG-BUFFER.
*   05 W03-CSQ4BCAQ.
*   COPY CSQ4VB3.
*
* API control blocks
*
* 01 MQM-MESSAGE-DESCRIPTOR.
*   COPY CMQMDV.
* 01 MQM-GET-MESSAGE-OPTIONS.
*   COPY CMQGMV.
*
* CMQV contains constants (for filling in the
* control blocks) and return codes (for testing
* the result of a call).
*
* 01 MQM-MQV.
*   COPY CMQV SUPPRESS.
* -----*
* PROCEDURE DIVISION.
* -----*
:
*   Open input queue.
:

```

```

*
*   Get and process messages.
*
*   COMPUTE MQGMO-OPTIONS = MQGMO-WAIT +
*                           MQGMO-ACCEPT-TRUNCATED-MSG +
*                           MQGMO-SYNCPOINT.
*   MOVE LENGTH OF W03-MSG-BUFFER TO W03-BUFFLEN.
*   MOVE W00-WAIT-INTERVAL TO MQGMO-WAITINTERVAL.
*   MOVE MQMI-NONE TO MQMD-MSGID.
*   MOVE MQCI-NONE TO MQMD-CORRELID.

```

```

*
* Make the first MQGET call outside the loop.
*
CALL 'MQGET' USING W03-HCONN
                  W03-HOBJ-CHECKQ
                  MQMD
                  MQGMO
                  W03-BUFFLEN
                  W03-MSG-BUFFER
                  W03-DATALEN
                  W03-COMPCODE
                  W03-REASON.

*
* Test the output of the MQGET call using the
* PERFORM loop that follows.
*
* Perform whilst no failure occurs
*   - process this message
*   - reset the call parameters
*   - get another message
* End-perform

*
* Test the output of the MQGET call. If the call
* fails, send an error message showing the
* completion code and reason code, unless the
* completion code is NO-MSG-AVAILABLE.
*
IF (W03-COMPCODE NOT = MQCC-FAILED) OR
(W03-REASON NOT = MQRC-NO-MSG-AVAILABLE)
  MOVE 'MQGET ' TO M02-OPERATION
  MOVE MQOD-OBJECTNAME TO M02-OBJECTNAME
  PERFORM RECORD-CALL-ERROR
END-IF.
:

```

COBOL 예시: 신호화를 사용하여 메시지 수신

이 예제는 신호보내기에서 MQGET 호출을 사용하는 방법을 보여줍니다. 신호는 오직, IBM MQ for z/OS와 함께 사용할 수 있습니다.

이 발췌문은 신용 조회 샘플 애플리케이션(프로그램 CSQ4CVB2)에서 가져온 것입니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS.](#)

```

:
* -----*
WORKING-STORAGE SECTION.
* -----*
*
* W00 - General work fields
:
01 W00-WAIT-INTERVAL PIC S9(09) BINARY VALUE 30000.
*
* W03 - MQM API fields
*
01 W03-HCONN PIC S9(9) BINARY VALUE ZERO.
01 W03-HOBJ-REPLYQ PIC S9(9) BINARY.
01 W03-COMPCODE PIC S9(9) BINARY.
01 W03-REASON PIC S9(9) BINARY.
01 W03-DATALEN PIC S9(9) BINARY.
01 W03-BUFFLEN PIC S9(9) BINARY.
:
01 W03-GET-BUFFER.
05 W03-CSQ4BQRM.
COPY CSQ4VB4.
*
05 W03-CSQ4BIIM REDEFINES W03-CSQ4BQRM.
COPY CSQ4VB1.
*
05 W03-CSQ4BPGM REDEFINES W03-CSQ4BIIM.
COPY CSQ4VB5.
:
* API control blocks
*
01 MQM-MESSAGE-DESCRIPTOR.
COPY CMQMDV.

```

```

01 MQM-GET-MESSAGE-OPTIONS.
   COPY CMQGMV.
   :
*   MQV contains constants (for filling in the
*   control blocks) and return codes (for testing
*   the result of a call).
*
01 MQM-MQV.
   COPY CMQV SUPPRESS.
* -----*
LINKAGE SECTION.
* -----*
01 L01-ECB-ADDR-LIST.
   05 L01-ECB-ADDR1      POINTER.
   05 L01-ECB-ADDR2      POINTER.

*
01 L02-ECBS.
   05 L02-INQUIRY-ECB1    PIC S9(09) BINARY.
   05 L02-REPLY-ECB2      PIC S9(09) BINARY.
01 REDEFINES L02-ECBS.
   05                      PIC X(02).
   05 L02-INQUIRY-ECB1-CC PIC S9(04) BINARY.
   05                      PIC X(02).
   05 L02-REPLY-ECB2-CC   PIC S9(04) BINARY.

*
* -----*
PROCEDURE DIVISION.
* -----*
:
* Initialize variables, open queues, set signal on
* inquiry queue.
:
* -----*
PROCESS-SIGNAL-ACCEPTED SECTION.
* -----*
* This section gets a message with signal.  If a
* message is received, process it.  If the signal
* is set or is already set, the program goes into
* an operating system wait.
* Otherwise an error is reported and call error set.
* -----*
*
PERFORM REPLYQ-GETSIGNAL.
*
EVALUATE TRUE
  WHEN (W03-COMPCODE = MQCC-OK AND
        W03-REASON = MQRC-NONE)
    PERFORM PROCESS-REPLYQ-MESSAGE
*
  WHEN (W03-COMPCODE = MQCC-WARNING AND
        W03-REASON = MQRC-SIGNAL-REQUEST-ACCEPTED)
    OR
    (W03-COMPCODE = MQCC-FAILED AND
     W03-REASON = MQRC-SIGNAL-OUTSTANDING)
    PERFORM EXTERNAL-WAIT
*
  WHEN OTHER
    MOVE 'MQGET SIGNAL' TO M02-OPERATION
    MOVE MQ0D-OBJECTNAME TO M02-OBJECTNAME
    PERFORM RECORD-CALL-ERROR
    MOVE W06-CALL-ERROR TO W06-CALL-STATUS
END-EVALUATE.
*
PROCESS-SIGNAL-ACCEPTED-EXIT.
* Return to performing section
EXIT.
EJECT
*

* -----*
EXTERNAL-WAIT SECTION.
* -----*
* This section performs an external CICS wait on two
* ECBs until at least one is posted.  It then calls
* the sections to handle the posted ECB.
* -----*
EXEC CICS WAIT EXTERNAL
      ECBLIST(W04-ECB-ADDR-LIST-PTR)

```

```

        NUMEVENTS(2)
        END-EXEC.
*
* At least one ECB must have been posted to get to this
* point. Test which ECB has been posted and perform
* the appropriate section.
*
        IF L02-INQUIRY-ECB1 NOT = 0
            PERFORM TEST-INQUIRYQ-ECB
        ELSE
            PERFORM TEST-REPLYQ-ECB
        END-IF.
*
        EXTERNAL-WAIT-EXIT.
*
        Return to performing section.
*
        EXIT.
        EJECT
        :
* -----*
        REPLYQ-GETSIGNAL SECTION.
* -----*
* This section performs an MQGET call (in syncpoint with *
* signal) on the reply queue. The signal field in the *
* MQGMO is set to the address of the ECB. *
* Response handling is done by the performing section. *
* -----*
*
        COMPUTE MQGMO-OPTIONS          = MQGMO-SYNCPPOINT +
                                         MQGMO-SET-SIGNAL.
        MOVE W00-WAIT-INTERVAL          TO MQGMO-WAITINTERVAL.
        MOVE LENGTH OF W03-GET-BUFFER TO W03-BUFFLEN.
*
        MOVE ZEROS                      TO L02-REPLY-ECB2.
        SET MQGMO-SIGNAL1 TO ADDRESS OF L02-REPLY-ECB2.

*
* Set msgid and correlid to nulls so that any message
* will qualify.
*
        MOVE MQMI-NONE TO MQMD-MSGID.
        MOVE MQCI-NONE TO MQMD-CORRELID.
*
        CALL 'MQGET' USING W03-HCONN
                           W03-HOBJ-REPLYQ
                           MQMD
                           MQGMO
                           W03-BUFFLEN
                           W03-GET-BUFFER
                           W03-DATALEN
                           W03-COMPCODE
                           W03-REASON.
*
        REPLYQ-GETSIGNAL-EXIT.
*
        Return to performing section.
*
        EXIT.
        EJECT
*
        :

```

COBOL 예시: 객체의 속성에 대해 묻기

이 예제는 MQINQ 호출을 사용하여 큐의 속성에 대해 조회하는 방법을 보여줍니다.

이 발췌문은 대기열 속성 샘플 애플리케이션에서 가져온 것입니다.  이것은 IBM MQ for z/OS 프로그램 CSQ4CVC1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

-  다중 플랫폼용 샘플 절차형 프로그램 사용

- ▶ **z/OS** 다음에 대한 예제 절차형 프로그램을 사용하여 z/OS

```

:
* -----*
* WORKING-STORAGE SECTION.
* -----*
*
* W02 - MQM API fields
*
01 W02-SELECTORCOUNT PIC S9(9) BINARY VALUE 2.
01 W02-INTATTRCOUNT PIC S9(9) BINARY VALUE 2.
01 W02-CHARATTRLENGTH PIC S9(9) BINARY VALUE ZERO.
01 W02-CHARATTRS PIC X VALUE LOW-VALUES.
01 W02-HCONN PIC S9(9) BINARY VALUE ZERO.
01 W02-HOBJ PIC S9(9) BINARY.
01 W02-COMPCODE PIC S9(9) BINARY.
01 W02-REASON PIC S9(9) BINARY.
01 W02-SELECTORS-TABLE.
05 W02-SELECTORS PIC S9(9) BINARY OCCURS 2 TIMES
01 W02-INTATTRS-TABLE.
05 W02-INTATTRS PIC S9(9) BINARY OCCURS 2 TIMES
*
* CMQODV defines the object descriptor (MQOD).
*
01 MQM-OBJECT-DESCRIPTOR.
COPY CMQODV.
*
* CMQV contains constants (for setting or testing field
* values) and return codes (for testing the result of a
* call).
*
01 MQM-CONSTANTS.
COPY CMQV SUPPRESS.
* -----*
* PROCEDURE DIVISION.
* -----*
*
* Get the queue name and open the queue.
*
:
*
* Initialize the variables for the inquiry call:
* - Set W02-SELECTORS-TABLE to the attributes whose
* status is required
* - All other variables are already set
*
MOVE MQIA-INHIBIT-GET TO W02-SELECTORS(1).
MOVE MQIA-INHIBIT-PUT TO W02-SELECTORS(2).

*
* Inquire about the attributes.
*
CALL 'MQINQ' USING W02-HCONN,
                  W02-HOBJ,
                  W02-SELECTORCOUNT,
                  W02-SELECTORS-TABLE,
                  W02-INTATTRCOUNT,
                  W02-INTATTRS-TABLE,
                  W02-CHARATTRLENGTH,
                  W02-CHARATTRS,
                  W02-COMPCODE,
                  W02-REASON.
*
* Test the output from the inquiry:
*
* - If the completion code is not OK, display an error
* message showing the completion and reason codes
*
* - Otherwise, move the correct attribute status into
* the relevant screen map fields
*
IF W02-COMPCODE NOT = MQCC-OK
MOVE 'MQINQ' TO M01-MSG4-OPERATION
MOVE W02-COMPCODE TO M01-MSG4-COMPCODE
MOVE W02-REASON TO M01-MSG4-REASON
MOVE M01-MESSAGE-4 TO M00-MESSAGE
*
ELSE
*
Process the changes.

```

```

:
END-IF.
:

```

COBOL 예시: 큐의 속성 설정

이 예제는 MQSET 호출을 사용하여 큐의 속성을 변경하는 방법을 보여줍니다.

이 발췌문은 대기열 속성 샘플 애플리케이션에서 가져온 것입니다. **z/OS** 이것은 IBM MQ for z/OS 프로그램 CSQ4CVC1 입니다.

예제 애플리케이션의 이름과 위치는 다음 항목을 참조하십시오:

- ▶ **Multi** 다중 플랫폼용 샘플 절차형 프로그램 사용
- ▶ **z/OS** 다음에 대한 예제 절차형 프로그램을 사용하여 z/OS

```

:
* -----*
WORKING-STORAGE SECTION.
* -----*
*
*   W02 - MQM API fields
*
01  W02-SELECTORCOUNT    PIC S9(9) BINARY VALUE 2.
01  W02-INTATTRCOUNT    PIC S9(9) BINARY VALUE 2.
01  W02-CHARATTRLENGTH   PIC S9(9) BINARY VALUE ZERO.
01  W02-CHARATTRS        PIC X      VALUE LOW-VALUES.
01  W02-HCONN            PIC S9(9) BINARY VALUE ZERO.
01  W02-HOBJ             PIC S9(9) BINARY.
01  W02-COMPCODE         PIC S9(9) BINARY.
01  W02-REASON           PIC S9(9) BINARY.
01  W02-SELECTORS-TABLE.
    05  W02-SELECTORS     PIC S9(9) BINARY OCCURS 2 TIMES.
01  W02-INTATTRS-TABLE.
    05  W02-INTATTRS     PIC S9(9) BINARY OCCURS 2 TIMES.
*
*   CMQODV defines the object descriptor (MQOD).
*
01  MQM-OBJECT-DESCRIPTOR.
    COPY CMQODV.
*
*   CMQV contains constants (for setting or testing
*   field values) and return codes (for testing the
*   result of a call).
*
01  MQM-CONSTANTS.
    COPY CMQV SUPPRESS.
* -----*
PROCEDURE DIVISION.
* -----*

```

```

*
*   Get the queue name and open the queue.
*
:
*
*
* Initialize the variables required for the set call:
* - Set W02-SELECTORS-TABLE to the attributes to be set
* - Set W02-INTATTRS-TABLE to the required status
* - All other variables are already set
*
    MOVE MQIA-INHIBIT-GET    TO W02-SELECTORS(1).
    MOVE MQIA-INHIBIT-PUT    TO W02-SELECTORS(2).
    MOVE MQQA-GET-INHIBITED TO W02-INTATTRS(1).
    MOVE MQQA-PUT-INHIBITED TO W02-INTATTRS(2).
*
*   Set the attributes.
*
    CALL 'MQSET' USING W02-HCONN,
                      W02-HOBJ,
                      W02-SELECTORCOUNT,
                      W02-SELECTORS-TABLE,
                      W02-INTATTRCOUNT,
                      W02-INTATTRS-TABLE,

```

```

W02-CHARATTRLENGTH,
W02-CHARATTRS,
W02-COMPCODE,
W02-REASON.
*
* Test the output from the call:
*
* - If the completion code is not OK, display an error
*   message showing the completion and reason codes
*
* - Otherwise, move 'INHIBITED' into the relevant
*   screen map fields
*
IF W02-COMPCODE NOT = MQCC-OK
  MOVE 'MQSET' TO M01-MSG4-OPERATION
  MOVE W02-COMPCODE TO M01-MSG4-COMPCODE
  MOVE W02-REASON TO M01-MSG4-REASON
  MOVE M01-MESSAGE-4 TO M00-MESSAGE
ELSE
*
* Process the changes.
*
*
END-IF.

```

어셈블리어로 ‘MQI’ 앱 개발하기

System/390 어셈블리 언어로 작성된 IBM MQ for z/OS MQI 프로그램에 대한 정보.

- [1177 페이지의 『이름』](#)
- [1177 페이지의 『MQI 호출 사용』](#)
- [1177 페이지의 『상수 선언』](#)
- [1178 페이지의 『구조 이름 지정』](#)
- [1178 페이지의 『구조 양식 지정』](#)
- [1178 페이지의 『목록 제어』](#)
- [1179 페이지의 『필드의 초기값 지정』](#)
- [1179 페이지의 『재입 가능한 프로그램 작성』](#)
- [1179 페이지의 『CEDF 사용』](#)

이름

호출 설명에서 매개변수의 이름, 구조 설명에서 필드의 이름은 대소문자 혼합으로 표시됩니다. IBM MQ로 제공된 어셈블리 언어 매크로에서 모든 이름은 대문자입니다.

MQI 호출 사용

MQI는 호출 인터페이스이므로 어셈블리 언어 프로그램은 OS 연계 변환을 따라야 합니다.

특히 MQI 호출을 발행하기 전에 어셈블리 언어 프로그램은 18개 이상의 완전한 단어로 이루어진 저장 영역의 레지스터 R13을 가리켜야 합니다. 이 저장 영역은 호출된 프로그램에 대한 스토리지를 제공합니다. 해당 콘텐츠가 삭제되기 전에 호출자의 레지스터를 저장하고 리턴 시 호출자 레지스터의 콘텐츠를 복원합니다.

참고: 이는 동적 스토리지를 설정하기 위해 DFHEIENT 매크로를 사용하지만, 기본 DATAREG를 R13에서 다른 레지스터로 대체하도록 선택하는 CICS 어셈블리 언어 프로그램의 경우 중요합니다. CICS 자원 관리자 인터페이스가 스텝에서 제어를 수신하는 경우 R13이 가리키는 주소에서 레지스터의 현재 콘텐츠를 저장합니다. 이러한 목적을 위해 저장 영역을 예약하는 데 실패하면 예측할 수 없는 결과를 제공하며 CICS에서 이상 종료될 수 있습니다.

상수 선언

대부분의 상수는 매크로 CMQA에서와 같이 선언됩니다.

그러나 다음 상수는 동일하게 정의할 수 없으며 기본 옵션을 사용하여 매크로를 호출하는 경우 포함되지 않습니다.

- MQACT_NONE
- MQCI_NONE
- MQFMT_NONE
- MQFMT_ADMIN
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- MQFMT_EVENT
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_PCF
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_XMIT_Q_HEADER
- MQMI_NONE

상수를 포함시키려면, 매크로를 호출할 때 키워드 EQUONLY=NO를 추가하십시오.

CMQA는 여러 선언에 대해 보호되므로 여러 번 포함할 수 있습니다. 키워드 EQUONLY는 처음 매크로가 포함될 때에만 적용됩니다.

구조 이름 지정

하나 이상의 인스턴스가 선언되도록 허용하려면, 구조를 생성하는 매크로에서 각 필드의 이름에 사용자 지정 가능한 문자열과 밑줄 문자(_)로 접두부를 지정합니다.

매크로를 호출하는 경우 문자열을 지정하십시오. 문자열을 지정하지 않는 경우 매크로는 접두부를 구성하기 위해 구조의 이름을 사용합니다.

```
* Declare two object descriptors
CMQODA          Prefix used="MQOD_" (the default)
MY_MQOD CMQODA  Prefix used="MY_MQOD_"
```

[호출 설명](#)에서 구조 선언은 기본 접두부를 표시합니다.

구조 양식 지정

매크로는 DSECT 매개변수에 의해 제어되는 두 가지 중 하나의 양식에서 구조 선언을 생성할 수 있습니다.

DSECT=YES

어셈블러 언어 DSECT 명령어는 새 데이터 섹션을 시작하는 데 사용됩니다. 구조 정의가 DSECT 명령문 바로 뒤에 옵니다. 스토리지가 할당되지 않으므로 초기화는 불가능합니다. 매크로 호출의 레이블은 데이터 섹션의 이름으로 사용됩니다. 레이블이 지정되지 않은 경우 구조의 이름이 사용됩니다.

DSECT=NO

어셈블러 언어 DC 명령어는 루틴의 현재 위치에서 구조를 정의하는 데 사용됩니다. 필드는 매크로 호출에 관련 매개변수를 코딩하여 지정할 수 있는 값으로 초기화됩니다. 매크로 호출에서 값이 지정되지 않은 필드는 기본값을 사용하여 초기화됩니다.

DSECT=NO는 DSECT 매개변수가 지정되지 않은 것으로 가정합니다.

목록 제어

LIST 매개변수를 사용하는 어셈블러 언어 목록에서 구조 선언의 표시를 제어할 수 있습니다.

LIST=YES

구조 선언이 어셈블러 언어 목록에 표시됩니다.

LIST=NO

구조 선언이 어셈블러 언어 목록에 표시되지 않습니다. LIST 매개변수가 지정되지 않은 것으로 가정합니다.

필드의 초기값 지정

필요한 값을 동반하는 매크로 호출의 매개변수로 (접두부 없이) 해당 필드의 이름을 코딩하여 구조에서 필드를 초기화하는 데 사용될 값을 지정할 수 있습니다.

예를 들어, MQMT_REQUEST로 초기화된 *MsgType* 필드 및 문자열 MY_REPLY_TO_QUEUE로 초기화된 *ReplyToQ* 필드를 가진 메시지 디스크립터 구조를 선언하려면, 다음 코드를 사용하십시오.

```
MY_MQMD      CMQMDA      MSGTYPE=MQMT_REQUEST,      X
REPLYTOQ=MY_REPLY_TO_QUEUE
```

이름 지정된 상수를 매크로 호출의 값으로 지정하는 경우 CMQA 매크로를 사용하여 이름 지정된 상수를 정의하십시오. 문자열인 값을 작은따옴표(' ')로 묶지 않아야 합니다.

재입 가능한 프로그램 작성

IBM MQ는 입력 및 출력 모두에 대해 해당 구조를 사용합니다. 프로그램을 재입 가능하도록 유지하려면 다음을 수행하십시오.

1. 구조의 작업 스토리지 버전을 DSECT로 정의하거나 이미 정의된 DSECT 내에 구조 인라인을 정의하십시오. 그리고 다음을 사용하여 확보된 스토리지로 DSECT를 복사하십시오.

- 배치 및 TSO 프로그램의 경우 STORAGE 또는 GETMAIN z/OS 어셈블러 매크로
- CICS, 작업 스토리지 DSECT (DFHEISTG) 또는 EXEC CICS GETMAIN 명령

이러한 작업 스토리지 구조를 올바르게 초기화하려면 해당하는 구조의 상수 버전을 작업 스토리지 버전에 복사하십시오.

참고: MQMD 및 MQXQH 구조의 길이는 각각 256바이트 이상입니다. 이러한 구조를 스토리지에 복사하려면, MVCL 어셈블러 명령을 사용하십시오.

2. CALL 매크로의 LIST 양식(MF=L)을 사용하여 스토리지에서 공간을 예약하십시오. 1179 페이지의 『CEDF 사용』 CALL 매크로를 사용하여 MQI 호출을 수행할 때는, 아래 예시와 같이 앞서 할당해 둔 저장 공간을 사용하여 매크로의 EXECUTE 형식(MF=E)을 사용하십시오. 이를 수행하기 위한 추가적인 예제는 IBM MQ와 함께 제공되는 어셈블러 언어 샘플 프로그램을 참조하십시오.

어셈블러 언어 RENT 옵션을 사용하면 프로그램이 재입 가능한지 판별하는 데 도움이 됩니다.

재입력 가능한 프로그램 작성에 관한 정보는 [응용 프로그램 개발 가이드: z/OSMVS 어셈블리어 언어 프로그램을 참조하십시오.](#)

CEDF 사용

프로그램을 디버그하는 데 도움을 주기 위해 CICS 제공 트랜잭션, CEDF(CICS Execution Diagnostic Facility)를 사용하려는 경우, VL 키워드를 각 CALL 명령문에 추가하십시오. 예를 들면, 다음과 같습니다.

```
CALL MQCONN, (NAME, HCONN, COMPCODE, REASON), MF=(E, PARMAREA), VL
```

이전 예는 PARMAREA가 사용자가 지정한 작업 스토리지 영역인 재입 가능한 어셈블러 언어 코드입니다.

관련 참조

1080 페이지의 『MQI 샘플에 표시된 기능은 다음과 같습니다. z/OS』

각 샘플 애플리케이션에서 시연된 ‘MQI’ 기능에 대한 요약으로, 각 샘플이 작성된 프로그래밍 언어와 실행 환경이 포함됩니다.

MQI 호출 사용

MQI는 호출 인터페이스이므로 어셈블러 언어 프로그램은 OS 연계 변환을 따라야 합니다. 특히 MQI 호출을 발행하기 전에 어셈블러 언어 프로그램은 18개 이상의 완전한 단어로 이루어진 저장 영역의 레지스터 R13을 가리켜

참고: 이는 동적 스토리지를 설정하기 위해 DFHEIENT 매크로를 사용하지만, 기본 DATAREG를 R13에서 다른 레지스터로 대체하도록 선택하는 CICS 어셈블러 언어 프로그램의 경우 중요합니다. CICS 자원 관리자 인터페이스가 스텝에서 제어를 수신하는 경우 R13이 가리키는 주소에서 레지스터의 현재 콘텐츠를 저장합니다. 이러한 목적을 위해 적절한 저장 영역을 예약하는 데 실패하면 예측할 수 없는 결과를 제공하며 CICS에서 이상 종료될 수 있습니다.

이 예에서는 MQCONN 호출을 사용하여 프로그램을 z/OS 일괄처리의 큐 관리자에 연결하는 방법을 보여줍니다. 이 발췌문은 Browse 샘플 애플리케이션(프로그램 CSQ4BAA1)에서 가져온 것입니다.

```

: WORKAREA DSECT
*
PARMLIST CALL ,(0,0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
COMPCODE DS      F      Completion code
REASON   DS      F      Reason code
HCONN    DS      F      Connection handle
          ORG
PARMADDR DS      F      Address of parm field
PARMLEN  DS      H      Length of parm field
*
MQMNAME  DS      CL48    Queue manager name
*
*
*****
* SECTION NAME : MAINPARM
*****
MAINPARM DS      0H
          MVI     MQMNAME,X'40'
          MVC     MQMNAME+1(L'MQMNAME-1),MQMNAME
*
* Space out first byte and initialize
*
*
* Code to address and verify parameters passed omitted
*
*
PARM1MVE DS      0H
          SR      R1,R3      Length of data
          LA      R4,MQMNAME Address for target
          BCTR    R1,R0      Reduce for execute
          EX      R1,MOVEPARM Move the data
*
*****
* EXECUTES
*****
MOVEPARM MVC      0(*-*,R4),0(R3)
*
EJECT

```

1180 IBM MO 용 애플리케이션 개발


```

*      MVC      INF4_TYP,=CL10'CONNECT      '
BAL      R7,ERRCODE                          Translate error
LA       R0,8                                Set exit code
ST       R0,EXITCODE                          to 8
B        ENDPROG                             End the program
*

```

▶ z/OS 어셈블러 예제: 큐 매니저에서 분리하기

이 예에서는 MQDISC 호출을 사용하여 z/OS 일괄처리의 큐 관리자에서 프로그램의 연결을 끊는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 다음에 대한 예제 절차형 프로그램을 사용하여 z/OS.

```

*
*      ISSUE MQI DISC REQUEST USING REENTRANT FORM
*      OF CALL MACRO
*
*      HCONN WAS SET BY A PREVIOUS MQCONN REQUEST
*      R5 = WORK REGISTER
*
DISC      DS      0H
          CALL    MQDISC,                      X
          (HCONN,                      X
          COMPCODE,                      X
          REASON),                      X
          VL,MF=(E,CALLLST)
*
          LA      R5,MQCC_OK
          C       R5,COMPCODE
          BNE     BADCALL
          :

```

```

BADCALL    DS      0H
:
*
*
*
*
*          CMQA
*
*          WORKING STORAGE (RE-ENTRANT)
*
*          DSECT
*
CALLLST    CALL    , (0,0,0,0,0,0,0,0,0,0,0,0) ,VL,MF=L
*
HCONN      DS      F
COMPCODE   DS      F
REASON     DS      F
*
*
*          EQU      *-WKEG3
LEG3
END

```

z/OS 어셈블러 예제: 동적 큐 생성

이 예제는 MQOPEN 호출을 사용하여 동적 큐를 작성하는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 다음에 대한 예제 절차형 프로그램을 사용하여 z/OS.

```

:
*
*      R5 = WORK REGISTER.
*
OPEN      DS      0H
*
*      MVC  WOD_AREA,MQOD_AREA INITIALIZE WORKING VERSION OF
*              MQOD WITH DEFAULTS
*      MVC  WOD_OBJECTNAME,MOD_Q      COPY IN THE MODEL Q NAME

```

[illegible]

이 예제는 MQOPEN 호출을 사용하여 이미 정의된 큐를 여는 방법을 설명합니다. 이는 두 옵션을 지정하는 방법을 표시합니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 다음에 대한 예제 절차형 프로그램을 사용하여 z/OS.

1182 IBM MQ 용 애플리케이션 개발


```

BADCALL DS 0H
:
*
*          CONSTANTS
*
*          CMQA
*
*          WORKING STORAGE (REENTRANT)
*
WEG4 DSECT
*
CALLLST CALL , (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
HCONN DS F
HOBJ DS F
OPTIONS DS F
COMPCODE DS F
REASON DS F
*
*
LEG4 EQU *-WKEG4
END

```

어셈블러 예제: MQPUT을 사용하여 메시지 전송

이 예제는 MQPUT 호출을 사용하여 큐에 메시지를 넣는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS.](#)

```

:
*      CONNECT TO QUEUE MANAGER
*
CONN  DS 0H
:
*
*      OPEN A QUEUE
*
OPEN  DS 0H
:
*
*      R4,R5,R6,R7 = WORK REGISTER.
*
PUT   DS 0H
      LA R4,MQMD          SET UP ADDRESSES AND
      LA R5,MQMD_LENGTH   LENGTH FOR USE BY MVCL
      LA R6,WMD           INSTRUCTION, AS MQMD IS
      LA R7,WMD_LENGTH    OVER 256 BYES LONG.
      MVCL R6,R4          INITIALIZE WORKING VERSION
*                          OF MESSAGE DESCRIPTOR
*
*      MVC WPMO_AREA,MQPMO_AREA  INITIALIZE WORKING MQPMO
*
      LA R5,BUFFER_LEN     RETRIEVE THE BUFFER LENGTH
      ST R5,BUFFLEN        AND SAVE IT FOR MQM USE
*
      MVC BUFFER,TEST_MSG  SET THE MESSAGE TO BE PUT
*
*      ISSUE MQI PUT REQUEST USING REENTRANT FORM
*      OF CALL MACRO
*
*      HCONN WAS SET BY PREVIOUS MQCONN REQUEST
*      HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST
*
      CALL MQPUT,          X
          (HCONN,          X
           HOBJ,           X
           WMD,            X
           WPMO,           X
           BUFFLEN,        X
           BUFFER,         X
           COMPCODE,       X
           REASON),VL,MF=(E,CALLLST)
*
      LA R5,MQCC_OK
      C  R5,COMPCODE
      BNE BADCALL

```

```

*
*
BADCALL DS 0H
*
*
*
CONSTANTS
*
CMQMDA DSECT=NO,LIST=YES,PERSISTENCE=MQPER_PERSISTENT
CMQPMOA DSECT=NO,LIST=YES
CMQA
TEST_MSG DC CL80'THIS IS A TEST MESSAGE'
*
*
WORKING STORAGE DSECT
*
WORKSTG DSECT
*
COMPCODE DS F
REASON DS F
BUFFLEN DS F
OPTIONS DS F
HCONN DS F
HOBJ DS F
*
BUFFER DS CL80
BUFFER_LEN EQU *-BUFFER
*
WMD CMQMDA DSECT=NO,LIST=NO
WPMO CMQPMOA DSECT=NO,LIST=NO
*
CALLLST CALL , (0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
*
END

```

어셈블러 예제: `MQPUT1`를 사용하여 메시지 출력하기

이 예제는 MQPUT1 호출을 사용하여 큐를 열고 단일 메시지를 큐에 넣은 후에 큐를 닫는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#).

```

*
*
CONNECT TO QUEUE MANAGER
*
CONN DS 0H
*
*
R4,R5,R6,R7 = WORK REGISTER.
*
PUT DS 0H
*
MVC WOD_AREA,MQOD_AREA INITIALIZE WORKING VERSION OF
* MQOD WITH DEFAULTS
MVC WOD_OBJECTNAME,Q_NAME SPECIFY Q NAME FOR PUT1
*
LA R4,MQMD SET UP ADDRESSES AND
LA R5,MQMD_LENGTH LENGTH FOR USE BY MVCL
LA R6,WMD INSTRUCTION, AS MQMD IS
LA R7,WMD_LENGTH OVER 256 BYES LONG.
MVCL R6,R4 INITIALIZE WORKING VERSION
* OF MESSAGE DESCRIPTOR

*
MVC WPMO_AREA,MQPMO_AREA INITIALIZE WORKING MQPMO
*
LA R5,BUFFER_LEN RETRIEVE THE BUFFER LENGTH
ST R5,BUFFLEN AND SAVE IT FOR MQM USE
*
MVC BUFFER,TEST_MSG SET THE MESSAGE TO BE PUT
*
*
ISSUE MQI PUT REQUEST USING REENTRANT FORM OF CALL MACRO
*

```

```

*      HCONN WAS SET BY PREVIOUS MQCONN REQUEST
*      HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST
*
*      CALL MQPUT1,
*              (HCONN,
*              LMQOD,
*              LMQMD,
*              LMQPMO,
*              BUFFERLENGTH,
*              BUFFER,
*              COMPCODE,
*              REASON),VL,MF=(E,CALLST)
*
*      LA R5,MQCC_OK
*      C R5,COMPCODE
*      BNE BADCALL
*
*      :
BADCALL DS 0H
*      :
*

```

```

*      CONSTANTS
*
*      CMQMDA DSECT=NO,LIST=YES,PERSISTENCE=MQPER_PERSISTENT
*      CMQPMOA DSECT=NO,LIST=YES
*      CMQODA DSECT=NO,LIST=YES
*      CMQA
*
*      TEST_MSG DC CL80'THIS IS ANOTHER TEST MESSAGE'
*      Q_NAME DC CL48'TEST.QUEUE.NAME'
*
*      WORKING STORAGE DSECT
*
*      WORKSTG DSECT
*
*      COMPCODE DS F
*      REASON DS F
*      BUFFLEN DS F
*      OPTIONS DS F
*      HCONN DS F
*      HOBJ DS F
*
*      BUFFER DS CL80
*      BUFFER_LEN EQU *-BUFFER
*
*      WOD CMQODA DSECT=NO,LIST=YES WORKING VERSION OF MQOD
*      WMD CMQMDA DSECT=NO,LIST=NO
*      WPMO CMQPMOA DSECT=NO,LIST=NO
*
*      CALLST CALL ,(0,0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
*      :
*      END

```

▶ z/OS 어셈블러 예제: 메시지 받기

이 예제는 MQGET 호출을 사용하여 큐에서 메시지를 제거하는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#).

```

:
*
*      CONNECT TO QUEUE MANAGER
*
*      CONN DS 0H
*
*      :
*
*      OPEN A QUEUE FOR GET
*
*      OPEN DS 0H
*
*      :
*
*      R4,R5,R6,R7 = WORK REGISTER.
*

```



```

*
*      CONSTANTS
*
*      CMQMDA DSECT=NO,LIST=YES
*      CMQGMOA DSECT=NO,LIST=YES
*      CMQA
*
*      WORKING STORAGE DSECT
*
WORKSTG   DSECT
*
COMPCODE  DS F
REASON    DS F
BUFFLEN   DS F
DATALEN   DS F
OPTIONS    DS F
HCONN     DS F
HOBJ      DS F
*
BUFFER     DS CL80
BUFFER_LEN EQU *-BUFFER
*
WMD        CMQMDA DSECT=NO,LIST=NO
WGMO       CMQGMOA DSECT=NO,LIST=NO
*
CALLLST    CALL , (0,0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
:
END

```

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS.](#)

```

:
*      CONNECT TO QUEUE MANAGER
CONN   DS   0H
:
*      OPEN A QUEUE FOR GET
OPEN   DS   0H
:
*      R4,R5,R6,R7 = WORK REGISTER.
GET    DS   0H
      LA   R4,MQMD                SET UP ADDRESSES AND
      LA   R5,MQMD_LENGTH          LENGTH FOR USE BY MVCL
      LA   R6,WMD                  INSTRUCTION, AS MQMD IS
      LA   R7,WMD_LENGTH           OVER 256 BYES LONG.
      MVCL R6,R4                  INITIALIZE WORKING VERSION
*                                OF MESSAGE DESCRIPTOR

*
*      MVC   WGM0_AREA,MQGM0_AREA  INITIALIZE WORKING MQGM0
L       R5,=AL4(MQGM0_WAIT)
A       R5,=AL4(MQGM0_ACCEPT_TRUNCATED_MSG)
ST      R5,WGM0_OPTIONS
MVC     WGM0_WAITINTERVAL,TWO_MINUTES  WAIT UP TO TWO
                                         MINUTES BEFORE
                                         FAILING THE
                                         CALL

*
      LA   R5,BUFFER_LEN          RETRIEVE THE BUFFER LENGTH
      ST   R5,BUFFLEN             AND SAVE IT FOR MQM USE

*
*      ISSUE MQI GET REQUEST USING REENTRANT FORM OF CALL MACRO
*
*      HCONN WAS SET BY PREVIOUS MQCONN REQUEST
*      HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST
*
      CALL MQGET,                  X
      (HCONN,                      X
      HOBJ,                        X
      WMD,                         X
      WGM0,                        X
      BUFFLEN,                     X
      BUFFER,                      X
      DATALEN,                   X
      COMPCODE,                   X
      REASON),                    X
      VL,MF=(E,CALLST)

*
      LA   R5,MQCC_OK              DID THE MQGET REQUEST
      C    R5,COMPCODE             WORK OK?
      BE   GETOK                  YES, SO GO AND PROCESS.
      LA   R5,MQCC_WARNING         NO, SO CHECK FOR A WARNING.
      C    R5,COMPCODE             IS THIS A WARNING?
      BE   CHECK_W                YES, SO CHECK THE REASON.

*
      LA   R5,MQRC_NO_MSG_AVAILABLE IT MUST BE AN ERROR.
                                         IS IT DUE TO AN EMPTY
      C    R5,REASON               QUEUE?
      BE   NOMSG                  YES, SO HANDLE THE ERROR
      B    BADCALL                NO, SO GO TO ERROR ROUTINE

*
CHECK_W DS   0H
      LA   R5,MQRC_TRUNCATED_MSG_ACCEPTED IS THIS A
                                         TRUNCATED
      C    R5,REASON               MESSAGE?
      BE   GETOK                  YES, SO GO AND PROCESS.
      B    BADCALL                NO, SOME OTHER WARNING

*
NOMSG   DS   0H
:
GETOK   DS   0H
:

BADCALL DS   0H
:
*
*      CONSTANTS
*

```

```

CMQMDA DSECT=NO,LIST=YES
CMQGMOA DSECT=NO,LIST=YES
CMQA
*
TWO_MINUTES DC F'120000'      GET WAIT INTERVAL
*
*      WORKING STORAGE DSECT

```

```

*
WORKSTG  DSECT
*
COMPCODE DS F
REASON   DS F
BUFFLEN  DS F
DATALEN  DS F
OPTIONS  DS F
HCONN    DS F
HOBJ     DS F
*
BUFFER   DS CL80
BUFFER_LEN EQU *-BUFFER
*
WMD      CMQMDA DSECT=NO,LIST=NO
WGMO     CMQGMOA DSECT=NO,LIST=NO
*
CALLLIST CALL , (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
:
END

```

어셈블러 예제: 신호 전송을 이용한 메시지 수신

이 예는 MQGET 호출을 사용하여 적절한 메시지가 큐에 도착할 때 알림을 받도록 신호를 설정하는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#).

```

:
*
*      CONNECT TO QUEUE MANAGER
*
CONN    DS  0H
:
*
*      OPEN A QUEUE FOR GET
*
OPEN    DS  0H
:
*
*      R4,R5,R6,R7 = WORK REGISTER.
*
GET     DS  0H
LA      R4,MQMD                      SET UP ADDRESSES AND
LA      R5,MQMD_LENGTH               LENGTH FOR USE BY MVCL
LA      R6,WMD                      INSTRUCTION, AS MQMD IS
LA      R7,WMD_LENGTH               OVER 256 BYES LONG.
MVCL    R6,R4                      INITIALIZE WORKING VERSION
*                                  OF MESSAGE DESCRIPTOR

```

```

*
MVC     WGMO_AREA,MQGMO_AREA        INITIALIZE WORKING MQGMO
LA      R5,MQGMO_SET_SIGNAL
ST      R5,WGMO_OPTIONS
MVC     WGMO_WAITINTERVAL,FIVE_MINUTES WAIT UP TO FIVE
*                                           MINUTES BEFORE
*                                           FAILING THE CALL
*
XC      SIG_ECB,SIG_ECB             CLEAR THE ECB
LA      R5,SIG_ECB                  GET THE ADDRESS OF THE ECB
ST      R5,WGMO_SIGNAL1             AND PUT IT IN THE WORKING
*                                  MQGMO
*
LA      R5,BUFFER_LEN               RETRIEVE THE BUFFER LENGTH

```

```

      ST   R5,BUFFLEN          AND SAVE IT FOR MQM USE
*
*
*   ISSUE MQI GET REQUEST USING REENTRANT FORM OF CALL MACRO
*
*       HCONN WAS SET BY PREVIOUS MQCONN REQUEST
*       HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST
*
      CALL  MQGET,              X
            (HCONN,             X
             HOBJ,              X
             WMD,               X
             WGM0,              X
             BUFFLEN,           X
             BUFFER,            X
             DATALEN,          X
             COMPCODE,          X
             REASON),           X
            VL,MF=(E,CALLST)
*
      LA   R5,MQCC_OK          DID THE MQGET REQUEST
      C    R5,COMPCODE          WORK OK?
      BE   GETOK               YES, SO GO AND PROCESS.
      LA   R5,MQCC_WARNING     NO, SO CHECK FOR A WARNING.
      C    R5,COMPCODE          IS THIS A WARNING?
      BE   CHECK_W             YES, SO CHECK THE REASON.
      B    BADCALL             NO, SO GO TO ERROR ROUTINE
*

```

```

CHECK_W  DS   0H
      LA   R5,MQRC_SIGNAL_REQUEST_ACCEPTED
      C    R5,REASON          SIGNAL REQUEST SIGNAL SET?
      BNE  BADCALL           NO, SOME ERROR OCCURRED
      B    DOWORK             YES, SO DO SOMETHING
                               ELSE
*
*
CHECKSIG DS   0H
      CLC  SIG_ECB+1(3),=AL3(MQEC_MSG_ARRIVED)
                               IS A MESSAGE AVAILABLE?
      BE   GET                YES, SO GO AND GET IT
*
      CLC  SIG_ECB+1(3),=AL3(MQEC_WAIT_INTERVAL_EXPIRED)
                               HAVE WE WAITED LONG ENOUGH?
      BE   NOMSG              YES, SO SAY NO MSG AVAILABLE
      B    BADCALL           IF IT'S ANYTHING ELSE
                               GO TO ERROR ROUTINE.
*
*
DOWORK   DS   0H
      :
      TM   SIG_ECB,X'40'      HAS THE SIGNAL ECB BEEN POSTED?
      BO   CHECKSIG          YES, SO GO AND CHECK WHY
      B    DOWORK            NO, SO GO AND DO MORE WORK
*
NOMSG    DS   0H
      :
GETOK     DS   0H
      :
BADCALL   DS   0H
      :
*
*   CONSTANTS
*
      CMQMDA DSECT=NO,LIST=YES
      CMQGM0A DSECT=NO,LIST=YES
      CMQA
*
FIVE_MINUTES DC F'300000'    GET SIGNAL INTERVAL
*
      WORKING STORAGE DSECT
*
WORKSTG   DSECT
*
COMPCODE  DS   F
REASON    DS   F
BUFFLEN   DS   F
DATALEN   DS   F
OPTIONS   DS   F
HCONN     DS   F
HOBJ      DS   F
SIG_ECB   DS   F

```

```

*
BUFFER    DS CL80
BUFFER_LEN EQU *-BUFFER
*
WMD       CMQMDA DSECT=NO,LIST=NO
WGMO      CMQGMOA DSECT=NO,LIST=NO
*
CALLLIST  CALL , (0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
:
:
END

```

어셈블러 예제: 큐의 속성 조회 및 설정

이 예제는 MQINQ 호출을 사용하여 큐의 속성을 조회하고 MQSET 호출을 사용하여 큐의 속성을 변경하는 방법을 보여줍니다.

이 발췌문은 대기열 속성 샘플 애플리케이션(프로그램 'CSQ4CAC1')에서 가져온 것입니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#).

```

:
:
DFHEISTG DSECT
:
OBJDESC  CMQODA LIST=YES    Working object descriptor
*
SELECTORCOUNT DS F        Number of selectors
INTATTRCOUNT DS F        Number of integer attributes
CHARATTRLENGTH DS F        char attributes length
CHARATTRS      DS C        Area for char attributes
*
OPTIONS DS F               Command options
HCONN DS F                Handle of connection
HOBJ DS F                 Handle of object
COMPCODE DS F             Completion code
REASON DS F               Reason code
SELECTOR DS 2F            Array of selectors
INTATTRS DS 2F            Array of integer attributes
:
OBJECT DS CL(MQ_Q_NAME_LENGTH) Name of queue
:
CALLLIST CALL , (0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*****
*
:
PROGRAM EXECUTION STARTS HERE
*
:
CSQ4CAC1 DFHEIENT CODEREG=(R3),DATAREG=(R13)
:
*
Initialize the variables for the set call
*
SR R0,R0          Clear register zero
ST R0,CHARATTRLENGTH Set char length to zero
LA R0,2           Load to set
ST R0,SELECTORCOUNT selectors add
ST R0,INTATTRCOUNT integer attributes
*
LA R0,MQIA_INHIBIT_GET Load q attribute selector
ST R0,SELECTOR+0      Place in field
LA R0,MQIA_INHIBIT_PUT Load q attribute selector
ST R0,SELECTOR+4      Place in field
*
UPDTEST DS 0H
CLC ACTION,CINHIB     Are we inhibiting?
BE UPDINHBT          Yes branch to section
*
CLC ACTION,CALLOW     Are we allowing?
BE UPDALLOW          Yes branch to section
*
MVC M00_MSG,M01_MSG1 Invalid request
BR R6                Return to caller
*

```

```

UPDINHBT DS 0H
MVC UPDTYPE,CINHIBIT   Indicate action type
LA R0,MQQA_GET_INHIBITED Load attribute value
ST R0,INTATTRS+0      Place in field
LA R0,MQQA_PUT_INHIBITED Load attribute value

```

```

        ST    R0,INTATTRS+4      Place in field
        B     UPDCALL            Go and do call
*
UPDALLOW DS    0H
        MVC   UPDTYPE,CALLOWED   Indicate action type
        LA    R0,MQQA_GET_ALLOWED Load attribute value
        ST    R0,INTATTRS+0      Place in field
        LA    R0,MQQA_PUT_ALLOWED Load attribute value
        ST    R0,INTATTRS+4      Place in field
        B     UPDCALL            Go and do call
*
UPDCALL  DS    0H
        CALL  MQSET,              C
                (HCONN,           C
                HOBJ,             C
                SELECTORCOUNT,   C
                SELECTOR,         C
                INTATTRCOUNT,    C
                INTATTRS,         C
                CHARATTRLENGTH,   C
                CHARATTRS,        C
                COMPCODE,         C
                REASON),          C
                VL,MF=(E,CALLLIST)
*
        LA    R0,MQCC_OK         Load expected compcode
        C     R0,COMPCODE        Was set successful?
        :
* SECTION NAME : INQUIRE *
* FUNCTION : Inquires on the objects attributes *
* CALLED BY : PROCESS *
* CALLS : OPEN, CLOSE, CODES *
* RETURN : To Register 6 *
INQUIRE DS    0H
        :

* Initialize the variables for the inquire call
*
        SR    R0,R0              Clear register zero
        ST    R0,CHARATTRLENGTH Set char length to zero
        LA    R0,2               Load to set
        ST    R0,SELECTORCOUNT selectors add
        ST    R0,INTATTRCOUNT integer attributes
*
        LA    R0,MQIA_INHIBIT_GET Load attribute value
        ST    R0,SELECTOR+0      Place in field
        LA    R0,MQIA_INHIBIT_PUT Load attribute value
        ST    R0,SELECTOR+4      Place in field
        CALL  MQINQ,              C
                (HCONN,           C
                HOBJ,             C
                SELECTORCOUNT,   C
                SELECTOR,         C
                INTATTRCOUNT,    C
                INTATTRS,         C
                CHARATTRLENGTH,   C
                CHARATTRS,        C
                COMPCODE,         C
                REASON),          C
                VL,MF=(E,CALLLIST)
        LA    R0,MQCC_OK         Load expected compcode
        C     R0,COMPCODE        Was inquire successful?
        :

```

PL/I 에서 MQI 앱 개발하기

PL/I로 IBM MQ를 코딩하는 경우 유용한 정보입니다.

구조

구조는 BASED 속성을 사용하여 선언되므로 프로그램에서 하나 이상의 구조 인스턴스를 선언하지 않는 한 스토리지를 사용하지 마십시오.

like 속성을 사용하여 구조의 인스턴스를 선언할 수 있습니다. 예를 들어,

```
dcl my_mqmd like MQMD; /* one instance */
dcl my_other_mqmd like MQMD; /* another one */
```

구조 필드는 INITIAL 속성을 사용하여 선언됩니다. 구조의 인스턴스를 선언하는 데 like 속성이 사용되는 경우 인스턴스는 해당 구조에 정의된 초기값을 상속합니다. 필수 값이 초기값과 다른 필드만 설정해야 합니다.

PL/I는 대소문자를 구분하지 않으므로 호출 이름, 구조 필드 및 상수를 소문자, 대문자 또는 대소문자 혼합으로 코드화할 수 있습니다.

이름 지정된 상수

이름 지정된 상수는 매크로 변수로 선언됩니다. 결과적으로 프로그램에서 언급되지 않는 이름 지정된 상수는 컴파일 프로시저에서 스토리지를 차지하지 않습니다.

그러나 프로그램이 컴파일될 때 소스를 매크로 프리프로세서에서 처리하게 되는 컴파일러 옵션을 지정해야 합니다.

모든 매크로 변수는 숫자 값을 나타내더라도 문자 변수입니다. 직관에 반대되는 것처럼 보일 수 있어도 매크로 변수가 매크로 프로세서에 의해 대체된 후 데이터 유형 충돌은 발생하지 않습니다. 예를 들어, 다음과 같습니다.

```
%dcl MQMD_STRUC_ID char;
%MQMD_STRUC_ID = 'MQMD';

%dcl MQMD_VERSION_1 char;
%MQMD_VERSION_1 = '1';
```

관련 참조

1080 페이지의 『MQI 샘플에 표시된 기능은 다음과 같습니다. z/OS』

각 샘플 애플리케이션에서 시연된 ‘MQI’ 기능에 대한 요약으로, 각 샘플이 작성된 프로그래밍 언어와 실행 환경이 포함됩니다.

▶ z/OS PL/I 예시: 큐 매니저에 연결하기

이 예에서는 MQCONN 호출을 사용하여 프로그램을 z/OS 일괄처리의 큐 관리자에 연결하는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS.](#)

```
%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*****
/* STRUCTURE BASED ON PARAMETER INPUT AREA (PARAM) */
*****/
DCL 1 INPUT_PARAM      BASED(ADDR(PARAM)),
    2 PARAM_LENGTH     FIXED BIN(15),
    2 PARAM_MQMNAME    CHAR(48);
:

/*****
/* WORKING STORAGE DECLARATIONS
*****/
DCL MQMNAME             CHAR(48);
DCL COMPCODE            BINARY FIXED (31);
DCL REASON              BINARY FIXED (31);
DCL HCONN               BINARY FIXED (31);
:

/*****
/* COPY QUEUE MANAGER NAME PARAMETER
/* TO LOCAL STORAGE
*****/
MQMNAME = ' ';
MQMNAME = SUBSTR(PARAM_MQMNAME,1,PARAM_LENGTH);
:

/*****
/* CONNECT FROM THE QUEUE MANAGER
*****/
CALL MQCONN (MQMNAME, /* MQM SYSTEM NAME
                  HCONN, /* CONNECTION HANDLE
```

```

                COMPCODE,      /* COMPLETION CODE      */
                REASON);      /* REASON CODE        */

/*****
/* TEST THE COMPLETION CODE OF THE CONNECT CALL.      */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE      */
/* SHOWING THE COMPLETION CODE AND THE REASON CODE.    */
*****/
IF COMPCODE = MQCC_OK
    THEN DO;
    :
    CALL ERROR_ROUTINE;
END;

```

▶ z/OS PL/I 예시: 큐 매니저에서 연결 해제

이 예에서는 MQDISC 호출을 사용하여 z/OS 일괄처리의 큐 관리자에서 프로그램의 연결을 끊는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS.](#)

```

%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*****
/* WORKING STORAGE DECLARATIONS      */
*****/
DCL COMPCODE      BINARY FIXED (31);
DCL REASON        BINARY FIXED (31);
DCL HCONN         BINARY FIXED (31);
:
/*****
/* DISCONNECT FROM THE QUEUE MANAGER  */
*****/
CALL MQDISC (HCONN,      /* CONNECTION HANDLE */
             COMPCODE,   /* COMPLETION CODE   */
             REASON);    /* REASON CODE       */

/*****
/* TEST THE COMPLETION CODE OF THE DISCONNECT CALL.      */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE      */
/* SHOWING THE COMPLETION CODE AND THE REASON CODE.    */
*****/
IF COMPCODE = MQCC_OK
    THEN DO;
    :
    CALL ERROR_ROUTINE;
END;

```

▶ z/OS PL/I 예시: 동적 큐 생성

이 예제는 MQOPEN 호출을 사용하여 동적 큐를 작성하는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS.](#)

```

%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*****
/* WORKING STORAGE DECLARATIONS      */
*****/
DCL COMPCODE      BINARY FIXED (31);
DCL REASON        BINARY FIXED (31);
DCL HCONN         BINARY FIXED (31);
DCL HOBJ          BINARY FIXED (31);
DCL OPTIONS       BINARY FIXED (31);
:
DCL MODEL_QUEUE_NAME CHAR(48) INIT('PL1.REPLY.MODEL');
DCL DYNAMIC_NAME_PREFIX CHAR(48) INIT('PL1.TEMPQ.*');
DCL DYNAMIC_QUEUE_NAME CHAR(48) INIT(' ');
:

```

```

/*****
/* LOCAL COPY OF OBJECT DESCRIPTOR
*/
*****/
DCL 1 LMQOD LIKE MQOD;
:
/*****
/* SET UP OBJECT DESCRIPTOR FOR OPEN OF REPLY QUEUE
*/
*****/
LMQOD.OBJECTTYPE =MQOT_Q;
LMQOD.OBJECTNAME = MODEL_QUEUE_NAME;
LMQOD.DYNAMICQNAME = DYNAMIC_NAME_PREFIX;
OPTIONS = MQOO_INPUT_EXCLUSIVE;

CALL MQOPEN (HCONN,
              LMQOD,
              OPTIONS,
              HOBJ,
              COMPCODE,
              REASON);

/*****
/* TEST THE COMPLETION CODE OF THE OPEN CALL.
*/
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE
*/
/* SHOWING THE COMPLETION CODE AND THE REASON CODE.
*/
/* IF THE CALL HAS SUCCEEDED THEN EXTRACT THE NAME OF
*/
/* THE NEWLY CREATED DYNAMIC QUEUE FROM THE OBJECT
*/
/* DESCRIPTOR.
*/
*****/
IF COMPCODE = MQCC_OK
THEN DO;
:
CALL ERROR_ROUTINE;
END;
ELSE
DYNAMIC_QUEUE_NAME = LMQOD_OBJECTNAME;

```

PL/I 예시: 기존 큐 열기

이 예제는 MQOPEN 호출을 사용하여 기존 큐를 여는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#).

```

%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*****
/* WORKING STORAGE DECLARATIONS
*/
*****/
DCL COMPCODE          BINARY FIXED (31);
DCL REASON            BINARY FIXED (31);
DCL HCONN             BINARY FIXED (31);
DCL HOBJ              BINARY FIXED (31);
DCL OPTIONS           BINARY FIXED (31);
:
DCL QUEUE_NAME        CHAR(48) INIT('PL1.LOCAL.QUEUE');
:
/*****
/* LOCAL COPY OF OBJECT DESCRIPTOR
*/
*****/
DCL 1 LMQOD LIKE MQOD;
:
/*****
/* SET UP OBJECT DESCRIPTOR FOR OPEN OF REPLY QUEUE
*/
*****/
LMQOD.OBJECTTYPE = MQOT_Q;
LMQOD.OBJECTNAME = QUEUE_NAME;
OPTIONS = MQOO_INPUT_EXCLUSIVE;

CALL MQOPEN (HCONN,
              LMQOD,
              OPTIONS,
              HOBJ,
              COMPCODE,
              REASON);

/*****

```

```

/* TEST THE COMPLETION CODE OF THE OPEN CALL.          */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE       */
/* SHOWING THE COMPLETION CODE AND THE REASON CODE.    */
/*****
  IF COMPCODE = MQCC_OK
  THEN DO;
  :
  CALL ERROR_ROUTINE;
END;

```

▶ z/OS PL/I 예시: 큐 닫기

이 예제는 MQCLOSE 호출을 사용하는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS.](#)

```

%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*****
/* WORKING STORAGE DECLARATIONS
*/
*****/
DCL COMPCODE          BINARY FIXED (31);
DCL REASON            BINARY FIXED (31);
DCL HCONN             BINARY FIXED (31);
DCL HOBJ              BINARY FIXED (31);
DCL OPTIONS           BINARY FIXED (31);
:
/*****
/* SET CLOSE OPTIONS
*/
*****/
OPTIONS=MQCO_NONE;

/*****
/* CLOSE QUEUE
*/
*****/
CALL MQCLOSE (HCONN, /* CONNECTION HANDLE
                  HOBJ, /* OBJECT HANDLE
                  OPTIONS, /* CLOSE OPTIONS
                  COMPCODE, /* COMPLETION CODE
                  REASON); /* REASON CODE

/*****
/* TEST THE COMPLETION CODE OF THE CLOSE CALL.
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE
/* SHOWING THE COMPLETION CODE AND THE REASON CODE.
*****/
  IF COMPCODE = MQCC_OK
  THEN DO;
  :
  CALL ERROR_ROUTINE;
END;

```

▶ z/OS PL/I 예시: MQPUT을 사용하여 메시지 전송

이 예제는 컨텍스트를 사용하여 MQPUT 호출을 사용하는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS.](#)

```

%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*****
/* WORKING STORAGE DECLARATIONS
*/
*****/
DCL COMPCODE          BINARY FIXED (31);
DCL REASON            BINARY FIXED (31);
DCL HCONN             BINARY FIXED (31);
DCL HOBJ              BINARY FIXED (31);
DCL OPTIONS           BINARY FIXED (31);

```

```

DCL BUFFLEN          BINARY FIXED (31);
DCL BUFFER           CHAR(80);
:
DCL PL1_TEST_MESSAGE CHAR(80)
INIT('***** THIS IS A TEST MESSAGE *****');
:
*****/
/* LOCAL COPY OF MESSAGE DESCRIPTOR          */
/* AND PUT MESSAGE OPTIONS                    */
*****/
DCL 1 LMQMD LIKE MQMD;
DCL 1 LMQPMO LIKE MQPMO;
:
*****/
/* SET UP MESSAGE DESCRIPTOR                  */
*****/
LMQMD.MSGTYPE = MQMT_DATAGRAM;
LMQMD.PRIORITY = 1;
LMQMD.PERSISTENCE = MQPER_PERSISTENT;
LMQMD.REPLYTOQ = ' ';
LMQMD.REPLYTOQMGR = ' ';
LMQMD.MSGID = MQMI_NONE;
LMQMD.CORRELID = MQCI_NONE;

/* SET UP PUT MESSAGE OPTIONS                  */
*****/
LMQPMO.OPTIONS = MQPMO_NO_SYNCPOINT;

*****/
/* SET UP LENGTH OF MESSAGE BUFFER AND THE MESSAGE */
*****/
BUFFLEN = LENGTH(BUFFER);
BUFFER = PL1_TEST_MESSAGE;
*****/
/*
/* HCONN WAS SET BY PREVIOUS MQCONN REQUEST.
/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST.
/*
*****/
CALL MQPUT (HCONN,
            HOBJ,
            LMQMD,
            LMQPMO,
            BUFFLEN,
            BUFFER,
            COMPCODE,
            REASON);

*****/
/* TEST THE COMPLETION CODE OF THE PUT CALL.          */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE      */
/* SHOWING THE COMPLETION CODE AND THE REASON CODE.   */
*****/
IF COMPCODE /= MQCC_OK
THEN DO;
:
CALL ERROR_ROUTINE;
END;

```

PL/I 예시: MQPUT1 를 사용하여 메시지 게시하기

이 예제는 MQPUT1 호출을 사용하는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#).

```

%INCLUDE SYSLIB(CMQEPP);
%INCLUDE SYSLIB(CMQP);
:
*****/
/* WORKING STORAGE DECLARATIONS          */
*****/
DCL COMPCODE          BINARY FIXED (31);
DCL REASON             BINARY FIXED (31);
DCL HCONN              BINARY FIXED (31);

```

```

DCL OPTIONS          BINARY FIXED (31);
DCL BUFFLEN          BINARY FIXED (31);
DCL BUFFER           CHAR(80);
:
DCL REPLY_TO_QUEUE   CHAR(48) INIT('PL1.REPLY.QUEUE');
DCL QUEUE_NAME       CHAR(48) INIT('PL1.LOCAL.QUEUE');
DCL PL1_TEST_MESSAGE CHAR(80)
INIT('***** THIS IS ANOTHER TEST MESSAGE *****');
:
/*****
/* LOCAL COPY OF OBJECT DESCRIPTOR, MESSAGE DESCRIPTOR */
/* AND PUT MESSAGE OPTIONS */
*****/
DCL 1 LMQOD LIKE MQOD;
DCL 1 LMQMD LIKE MQMD;
DCL 1 LMQPMO LIKE MQPMO;
:
/*****
/* SET UP OBJECT DESCRIPTOR AS REQUIRED. */
*****/
LMQOD.OBJECTTYPE = MQOT_Q;
LMQOD.OBJECTNAME = QUEUE_NAME;

/*****
/* SET UP MESSAGE DESCRIPTOR AS REQUIRED. */
*****/
LMQMD.MSGTYPE = MQMT_REQUEST;
LMQMD.PRIORITY = 5;
LMQMD.PERSISTENCE = MQPER_PERSISTENT;
LMQMD.REPLYTOQ = REPLY_TO_QUEUE;
LMQMD.REPLYTOQMGR = 'I';
LMQMD.MSGID = MQMI_NONE;
LMQMD.CORRELID = MQCI_NONE;

/*****
/* SET UP PUT MESSAGE OPTIONS AS REQUIRED */
*****/
LMQPMO.OPTIONS = MQPMO_NO_SYNCPOINT;

/*****
/* SET UP LENGTH OF MESSAGE BUFFER AND THE MESSAGE */
*****/
BUFFLEN = LENGTH(BUFFER);
BUFFER = PL1_TEST_MESSAGE;

CALL MQPUT1 (HCONN,
LMQOD,
LMQMD,
LMQPMO,
BUFFLEN,
BUFFER,
COMPCODE,
REASON);

/*****
/* TEST THE COMPLETION CODE OF THE PUT1 CALL. */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE SHOWING */
/* THE COMPLETION CODE AND THE REASON CODE. */
*****/
IF COMPCODE = MQCC_OK
THEN DO;
:
CALL ERROR_ROUTINE;
END;

```

PL/I 예시: 메시지 받기

이 예제는 MQGET 호출을 사용하여 큐에서 메시지를 제거하는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#).

```

%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*****

```



```

/* WORKING STORAGE DECLARATIONS */
/*****
DCL COMPCODE          BINARY FIXED (31);
DCL REASON            BINARY FIXED (31);
DCL HCONN            BINARY FIXED (31);
DCL HOBJ             BINARY FIXED (31);
DCL BUFFLEN          BINARY FIXED (31);
DCL DATALEN         BINARY FIXED (31);
DCL BUFFER           CHAR(80);
:
/*****

/*****
/* LOCAL COPY OF MESSAGE DESCRIPTOR AND */
/* GET MESSAGE OPTIONS */
/*****
DCL 1 LMQMD LIKE MQMD;
DCL 1 LMQGMO LIKE MQGMO;
:
/*****
/* SET UP MESSAGE DESCRIPTOR AS REQUIRED. */
/* MSGID AND CORRELID IN MQMD SET TO NULLS SO FIRST */
/* AVAILABLE MESSAGE WILL BE RETRIEVED. */
/*****
LMQMD.MSGID = MQMI_NONE;
LMQMD.CORRELID = MQCI_NONE;

/*****
/* SET UP GET MESSAGE OPTIONS AS REQUIRED. */
/*****
LMQGMO.OPTIONS = MQGMO_NO_SYNCPOINT;

/*****
/* SET UP LENGTH OF MESSAGE BUFFER. */
/*****
BUFFLEN = LENGTH(BUFFER);
/*****
/*
/* HCONN WAS SET BY PREVIOUS MQCONN REQUEST. */
/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST. */
/*
/*****

CALL MQGET (HCONN,
           HOBJ,
           LMQMD,
           LMQGMO,
           BUFFERLEN,
           BUFFER,
           DATALEN,
           COMPCODE,
           REASON);

/*****
/* TEST THE COMPLETION CODE OF THE GET CALL. */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE */
/* SHOWING THE COMPLETION CODE AND THE REASON CODE. */
/*****
IF COMPCODE = MQCC_OK
THEN DO;
:
CALL ERROR_ROUTINE;
END;

```

PL/I 예시: 대기 옵션을 사용하여 메시지 수신

이 예제는 대기 옵션으로 MQGET 호출을 사용하고 잘린 메시지를 받는 방법을 예시합니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#).

```

%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*****
/* WORKING STORAGE DECLARATIONS */
/*****

```

```

DCL COMPCODE          BINARY FIXED (31);
DCL REASON            BINARY FIXED (31);
DCL HCONN            BINARY FIXED (31);
DCL HOBJ            BINARY FIXED (31);
DCL BUFFLEN          BINARY FIXED (31);
DCL DATALEN         BINARY FIXED (31);
DCL BUFFER           CHAR(80);
:
/*****
/* LOCAL COPY OF MESSAGE DESCRIPTOR AND GET MESSAGE      */
/* OPTIONS                                                  */
*****/
DCL 1 LMQMD LIKE MQMD;
DCL 1 LMQGMO LIKE MQGMO;
:
/*****
/* SET UP MESSAGE DESCRIPTOR AS REQUIRED.                  */
/* MSGID AND CORRELID IN MQMD SET TO NULLS SO FIRST      */
/* AVAILABLE MESSAGE WILL BE RETRIEVED.                  */
*****/
LMQMD.MSGID = MQMI_NONE;
LMQMD.CORRELID = MQCI_NONE;

/*****
/* SET UP GET MESSAGE OPTIONS AS REQUIRED.                  */
/* WAIT INTERVAL SET TO ONE MINUTE.                      */
*****/
LMQGMO.OPTIONS = MQGMO_WAIT +
                MQGMO_ACCEPT_TRUNCATED_MSG +
                MQGMO_NO_SYNCPOINT;
LMQGMO.WAITINTERVAL=60000;

/*****
/* SET UP LENGTH OF MESSAGE BUFFER.                      */
*****/
BUFFLEN = LENGTH(BUFFER);

/*****
/*
/* HCONN WAS SET BY PREVIOUS MQCONN REQUEST.              */
/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST.               */
/*
*****/

CALL MQGET (HCONN,
            HOBJ,
            LMQMD,
            LMQGMO,
            BUFFERLEN,
            BUFFER,
            DATALEN,
            COMPCODE,
            REASON);

/*****
/* TEST THE COMPLETION CODE OF THE GET CALL.              */
/* TAKE APPROPRIATE ACTION BASED ON COMPLETION CODE AND  */
/* REASON CODE.                                           */
*****/

SELECT(COMPCODE);
  WHEN (MQCC_OK) DO;    /* GET WAS SUCCESSFUL */
  :
  END;
  WHEN (MQCC_WARNING) DO;
    IF REASON = MQRC_TRUNCATED_MSG_ACCEPTED
    THEN DO;           /* GET WAS SUCCESSFUL */
    :
    END;
    ELSE DO;
    :
    CALL ERROR_ROUTINE;
    END;
  END;
  WHEN (MQCC_FAILED) DO;
  :
  CALL ERROR_ROUTINE;
  END;
END;

```

```
OTHERWISE;  
END;
```

PL/I 예시: 신호화를 사용하여 메시지 수신

신호보내기에서 MQGET 호출을 사용하는 방법을 보여줍니다. 신호는 오직, IBM MQ for z/OS와 함께 사용할 수 있습니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS.](#)

```
%INCLUDE SYSLIB(CMQP);  
%INCLUDE SYSLIB(CMQEPP);  
:  
/*****  
/* WORKING STORAGE DECLARATIONS  
*****/  
DCL COMPCODE          BINARY FIXED (31);  
DCL REASON            BINARY FIXED (31);  
DCL HCONN             BINARY FIXED (31);  
DCL HOBJ              BINARY FIXED (31);  
DCL DATALEN          BINARY FIXED (31);  
DCL BUFFLEN           BINARY FIXED (31);  
DCL BUFFER            CHAR(80);  
:  
DCL ECB_FIXED          FIXED BIN(31);  
DCL 1 ECB_OVERLAY BASED(ADDR(ECB_FIXED)),  
    3 ECB_WAIT      BIT,  
    3 ECB_POSTED    BIT,  
    3 ECB_FLAG3_8    BIT(6),  
    3 ECB_CODE       PIC'999';  
:  
/*****  
/* LOCAL COPY OF MESSAGE DESCRIPTOR AND GET MESSAGE  
/* OPTIONS  
*****/  
DCL 1 LMQMD LIKE MQMD;  
DCL 1 LMQGMO LIKE MQGMO;  
:  
/*****  
/* CLEAR ECB FIELD.  
*****/  
ECB_FIXED = 0;  
:  
/*****  
/* SET UP MESSAGE DESCRIPTOR AS REQUIRED.  
/* MSGID AND CORRELID IN MQMD SET TO NULLS SO FIRST  
/* AVAILABLE MESSAGE WILL BE RETRIEVED.  
*****/  
LMQMD.MSGID = MQMI_NONE;  
LMQMD.CORRELID = MQCI_NONE;  
/*****  
/* SET UP GET MESSAGE OPTIONS AS REQUIRED.  
/* WAIT INTERVAL SET TO ONE MINUTE.  
*****/  
LMQGMO.OPTIONS = MQGMO_SET_SIGNAL +  
                MQGMO_NO_SYNCPOINT;  
LMQGMO.WAITINTERVAL=60000;  
LMQGMO.SIGNAL1 = ADDR(ECB_FIXED);  
  
/*****  
/* SET UP LENGTH OF MESSAGE BUFFER.  
/* CALL MESSAGE RETRIEVAL ROUTINE.  
*****/  
BUFFLEN = LENGTH(BUFFER);  
CALL GET_MSG;  
  
/*****  
/* TEST THE COMPLETION CODE OF THE GET CALL.  
/* TAKE APPROPRIATE ACTION BASED ON COMPLETION CODE AND  
/* REASON CODE.  
*****/  
  
SELECT;  
    WHEN ((COMPCODE = MQCC_OK) &
```

```

        (REASON = MQCC_NONE)) DO
        :
        CALL MSG_ROUTINE;
        :
    END;
    WHEN ((COMPCODE = MQCC_WARNING) &
          (REASON = MQRC_SIGNAL_REQUEST_ACCEPTED)) DO;
        :
        CALL DO_WORK;
        :
    END;
    WHEN ((COMPCODE = MQCC_FAILED) &
          (REASON = MQRC_SIGNAL_OUTSTANDING)) DO;
        :
        CALL DO_WORK;
        :
    END;
    OTHERWISE DO;          /* FAILURE CASE */
/*****
/* ISSUE AN ERROR MESSAGE SHOWING THE COMPLETION CODE */
/* AND THE REASON CODE. */
*****/
        :
        CALL ERROR_ROUTINE;
        :
    END;
END;
:

```

```

DO_WORK: PROC;
:
IF ECB_POSTED
THEN DO;
    SELECT(ECB_CODE);
    WHEN(MQEC_MSG_ARRIVED) DO;
        :
        CALL GET_MSG;
        :
    END;
    WHEN(MQEC_WAIT_INTERVAL_EXPIRED) DO;
        :
        CALL NO_MSG;
        :
    END;
    OTHERWISE DO;          /* FAILURE CASE */
/*****
/* ISSUE AN ERROR MESSAGE SHOWING THE COMPLETION CODE */
/* AND THE REASON CODE. */
*****/
        :
        CALL ERROR_ROUTINE;
        :
    END;
END;
:
END DO_WORK;
GET_MSG: PROC;

```

```

/*****
/*
/* HCONN WAS SET BY PREVIOUS MQCONN REQUEST. */
/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST. */
/* MD AND GMO SET UP AS REQUIRED. */
/*
*****/

    CALL MQGET (HCONN,
                HOBJ,
                LMQMD,
                LMQGMO,
                BUFLLEN,
                BUFFER,
                DATALEN,
                COMPCODE,

```

```

        REASON);

END GET_MSG;

NO_MSG: PROC;
:
END NO_MSG;

```

PL/I 예시: 객체의 속성에 대해 묻기

이 예제는 MQINQ 호출을 사용하여 큐의 속성에 대해 조회하는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#).

```

%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*****
/* WORKING STORAGE DECLARATIONS
*****/
DCL COMPCODE          BINARY FIXED (31);
DCL REASON            BINARY FIXED (31);
DCL HCONN             BINARY FIXED (31);
DCL HOBJ              BINARY FIXED (31);
DCL OPTIONS           BINARY FIXED (31);
DCL SELECTORCOUNT   BINARY FIXED (31);
DCL INTATTRCOUNT    BINARY FIXED (31);
DCL 1 SELECTOR_TABLE,
   3 SELECTORS(5)      BINARY FIXED (31);
DCL 1 INTATTR_TABLE,
   3 INTATTRS(5)       BINARY FIXED (31);
DCL CHARATTRLENGTH   BINARY FIXED (31);
DCL CHARATTRS        CHAR(100);
:

/*****
/* SET VARIABLES FOR INQUIRE CALL
/* INQUIRE ON THE CURRENT QUEUE DEPTH
*****/

SELECTORS(01) = MQIA_CURRENT_Q_DEPTH;

SELECTORCOUNT = 1;
INTATTRCOUNT  = 1;

CHARATTRLENGTH = 0;

/*****
/* HCONN WAS SET BY PREVIOUS MQCONN REQUEST.
/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST.
*****/
CALL MQINQ (HCONN,
            HOBJ,
            SELECTORCOUNT,
            SELECTORS,
            INTATTRCOUNT,
            INTATTRS,
            CHARATTRLENGTH,
            CHARATTRS,
            COMPCODE,
            REASON);

/*****
/* TEST THE COMPLETION CODE OF THE INQUIRE CALL.
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE SHOWING
/* THE COMPLETION CODE AND THE REASON CODE.
*****/
IF COMPCODE ->= MQCC_OK
THEN DO;
:
CALL ERROR_ROUTINE;
END;

```

▶ z/OS PL/I 예시: 큐의 속성 설정

이 예제는 MQSET 호출을 사용하여 큐의 속성을 변경하는 방법을 보여줍니다.

이 내용은 IBM MQ와 함께 제공된 샘플 애플리케이션에서 추출되지 않습니다.

예제 애플리케이션의 이름과 위치는 다음을 참조하십시오 [다음에 대한 예제 절차형 프로그램을 사용하여 z/OS](#).

```
%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*****
/* WORKING STORAGE DECLARATIONS
*****/
DCL COMPCODE          BINARY FIXED (31);
DCL REASON            BINARY FIXED (31);
DCL HCONN             BINARY FIXED (31);
DCL HOBJ              BINARY FIXED (31);
DCL OPTIONS           BINARY FIXED (31);
DCL SELECTORCOUNT    BINARY FIXED (31);
DCL INTATTRCOUNT    BINARY FIXED (31);
DCL 1 SELECTOR_TABLE,
    3 SELECTORS(5)      BINARY FIXED (31);
DCL 1 INTATTR_TABLE,
    3 INTATTRS(5)      BINARY FIXED (31);
DCL CHARATTRLENGTH    BINARY FIXED (31);
DCL CHARATTRS         CHAR(100);
:

/*****
/* SET VARIABLES FOR SET CALL
/* SET GET AND PUT INHIBITED
*****/

SELECTORS(01) = MQIA_INHIBIT_GET;
SELECTORS(02) = MQIA_INHIBIT_PUT;

INTATTRS(01) = MQQA_GET_INHIBITED;
INTATTRS(02) = MQQA_PUT_INHIBITED;

SELECTORCOUNT = 2;
INTATTRCOUNT  = 2;

CHARATTRLENGTH = 0;

/*****
/*
/* HCONN WAS SET BY PREVIOUS MQCONN REQUEST.
/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST.
/*
*****/
CALL MQSET (HCONN,
            HOBJ,
            SELECTORCOUNT,
            SELECTORS,
            INTATTRCOUNT,
            INTATTRS,
            CHARATTRLENGTH,
            CHARATTRS,
            COMPCODE,
            REASON);

/*****
/* TEST THE COMPLETION CODE OF THE SET CALL.
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE SHOWING
/* THE COMPLETION CODE AND THE REASON CODE.
*****/
IF COMPCODE /= MQCC_OK
THEN DO;
:
CALL ERROR_ROUTINE;
END;
```


IBM i RPG로 'MQI' 앱 개발하기

IBM MQ 문서에서, 호출의 매개변수, 데이터 유형 이름, 구조의 필드 및 상수 이름은 모두 긴 이름을 사용하여 설명됩니다. RPG에서 이러한 이름은 6개 이하의 대문자로 단축됩니다.

예를 들어, *MsgType* 필드는 RPG에서 *MDMT*가 됩니다. 자세한 정보는 [IBM i 애플리케이션 프로그래밍 참조서 \(ILE/RPG\)](#)를 참조하십시오.

관련 참조

977 페이지의 『C, COBOL 및 RPG MQI 샘플에 표시된 기능은 다음과 같습니다. IBM i』 샘플 프로그램에서 시연한 기법을 IBM MQIBM i 보여줍니다.

Managed File Transfer용 애플리케이션 개발

Managed File Transfer와 함께 실행할 프로그램을 지정하고, Managed File Transfer와 함께 Apache Ant 를 사용하고, 사용자 엑시트와 함께 Managed File Transfer 를 사용자 정의하고, 에이전트 명령 큐에 메시지를 넣어 Managed File Transfer 를 제어하십시오.

MFT와 함께 실행할 프로그램 지정

Managed File Transfer Agent가 실행 중인 시스템에서 프로그램을 실행할 수 있습니다. 파일 전송 요청의 일부 분으로 전송이 시작되기 전이나 완료된 후에 실행할 프로그램을 지정할 수 있습니다. 또한 관리 호출 요청을 제출하여 파일 전송 요청의 일부분이 아닌 프로그램을 시작할 수 있습니다.

이 태스크 정보

실행할 프로그램을 지정할 수 있는 다섯 가지 시나리오가 있습니다.

- 전송을 시작하기 전에 소스 에이전트에서 전송 요청의 일부로.
- 전송을 시작하기 전에 대상 에이전트에서 전송 요청의 일부로.
- 전송 완료 후 소스 에이전트에서 전송 요청의 일부로.
- 전송이 완료된 후 대상 에이전트에서 전송 요청의 일부로.
- 전송 요청의 일부분이 아닌 경우, 프로그램을 실행하도록 에이전트에 요청을 제출할 수 있습니다. 이 시나리오는 관리 호출이라고도 합니다.

사용자 엑시트 및 프로그램 호출은 다음 순서로 호출됩니다.

```
- SourceTransferStartExit(onSourceTransferStart).  
- PRE_SOURCE Command.  
- DestinationTransferStartExits(onDestinationTransferStart).  
- PRE_DESTINATION Command.  
- The Transfer request is performed.  
- DestinationTransferEndExits(onDestinationTransferEnd).  
- POST_DESTINATION Command.  
- SourceTransferEndExits(onSourceTransferEnd).  
- POST_SOURCE Command.
```

참고:

1. **DestinationTransferEndExits**는 전송이 정상적으로 또는 부분적으로 완료될 때만 실행됩니다.
2. **postDestinationCall**는 전송이 정상적으로 또는 부분적으로 완료될 때만 실행됩니다.
3. **SourceTransferEndExits**는 성공, 부분 성공 또는 전송 실패 시 실행됩니다.
4. **postSourceCall**은 다음 경우에만 호출됩니다.
 - 전송이 취소되지 않았습니다.
 - 성공 또는 부분 성공 결과가 존재합니다.
 - 모든 사후 대상 전송 프로그램이 실행되었습니다.

프로시저

- 다음 옵션 중 하나를 사용하여 실행할 프로그램을 지정하십시오.

Apache Ant 태스크 사용

`fte:filecopy`, `fte:filemove` 및 `fte:call` Ant 태스크 중 하나를 사용하여 프로그램을 시작하십시오. Ant 태스크를 통해 `fte:presrc`, `fte:predst`, `fte:postdst`, `fte:postsrc` 및 `fte:command` 중첩 요소를 사용하여 다섯 가지 시나리오 중 하나에서 프로그램을 지정할 수 있습니다. 자세한 정보는 [프로그램 호출 중첩 요소를 참조하십시오](#).

파일 전송 요청 메시지 편집

전송 요청으로 생성된 XML을 편집할 수 있습니다. 이 방법을 사용하면 **preSourceCall**, **postSourceCall**, **preDestinationCall**, **postDestinationCall** 및 **managedCall** 요소를 XML 파일에 추가하여 다섯 가지 시나리오 중 하나에서 프로그램을 실행할 수 있습니다. 그런 다음 수정된 이 XML 파일을 새 파일 전송 요청의 전송 정의로 사용하십시오. 예를 들어, **fteCreateTransfer -td** 매개변수를 사용할 수 있습니다. 자세한 정보는 [MFT 에이전트 호출 요청 메시지 예를 참조하십시오](#).

fteCreateTransfer 명령 사용

fteCreateTransfer 명령을 사용하여 시작할 프로그램을 지정할 수 있습니다. 이 명령을 사용하면 전송 요청의 일부분으로 처음 네 개의 시나리오에서 실행할 프로그램을 지정할 수 있지만 관리 호출은 시작할 수 없습니다. 사용할 매개변수에 대한 정보는 **fteCreateTransfer: 새 파일 전송 시작**을 참조하십시오. 이 명령 사용의 예는 [fteCreateTransfer를 사용한 프로그램 시작 예를 참조하십시오](#).

관련 참조

[commandPath MFT 특성](#)

관리 호출

Managed File Transfer(MFT) 에이전트는 일반적으로 파일이나 메시지를 전송하는 데 사용됩니다. 해당 에이전트는 관리 전송이라고 합니다. 에이전트를 사용하여 파일 또는 메시지를 전송할 필요 없이 명령, 스크립트 또는 JCL을 실행할 수도 있습니다. 이 기능은 관리 호출이라고 합니다.

관리 호출 요청은 다음과 같은 여러 방법으로 에이전트에 제출할 수 있습니다.

- `fte:call` Ant 태스크 사용.
- 명령 또는 스크립트를 실행하는 태스크 XML로 자원 모니터 구성. 자세한 정보는 [명령 및 스크립트를 시작하도록 모니터 태스크 구성을 참조하십시오](#).
- XML 메시지를 에이전트의 명령 대기열에 직접 넣습니다. 관리 호출 XML 스키마에 대한 자세한 정보는 [파일 전송 요청 메시지 형식](#)을 참조하십시오.

관리 호출의 경우, 실행 중인 명령 또는 스크립트를 포함하는 디렉토리를 에이전트 특성 **commandPath**에 지정해야 합니다.

관리되는 호출은 에이전트의 **commandPath**. 따라서 에이전트가 악성 코드를 실행할 수 없습니다.

중요사항: 이를 보장하기 위해 기본적으로 다음을 지정할 때 **commandPath**:

- 기존 에이전트 샌드박스는 에이전트가 시작될 때 구성되므로, 모든 **commandPath** 디렉터리가 전송에 대한 접근이 거부된 디렉터리 목록에 자동으로 추가됩니다.
- 에이전트가 시작될 때 기존 사용자 샌드박스가 업데이트되어 모든 디렉터리(및 하위 **commandPath** 디렉터리)가 요소와 `<write>` `<read>` 요소에 `<exclude>` 요소로 추가됩니다.
- 에이전트가 에이전트 샌드박스나 사용자 샌드박스를 사용하도록 구성되지 않은 경우, 에이전트 시작 시 거부된 디렉터리로 지정된 **commandPath** 디렉터리를 포함하는 새로운 에이전트 샌드박스가 생성됩니다.

또한 권한 부여된 사용자만 관리 호출 요청을 제출할 수 있도록 에이전트에 대한 권한 검사를 사용으로 설정할 수도 있습니다. 자세한 정보는 [MFT 에이전트 조치에 대한 사용자 권한 제한을 참조하십시오](#).

관리 호출의 일부로 호출된 명령, 스크립트 또는 JCL은 에이전트가 모니터링하는 외부 프로세스로 실행됩니다. 프로세스가 종료되면 관리된 호출이 완료되며, 프로세스의 반환 코드는 해당 작업을 **fte:call**Ant 호출한 에이전트 또는 Ant 스크립트에 제공됩니다.

fte:call Ant 태스크를 통해 관리 호출이 시작된 경우, Ant 스크립트가 리턴 코드의 값을 확인하여 관리 호출에 성공했는 판별할 수 있습니다.

기타 모든 유형의 관리 호출에 대해 관리 호출이 성공적으로 완료되었음을 표시하기 위해 사용해야 하는 리턴 코드 값을 지정할 수 있습니다. 에이전트는 외부 프로세스가 완료될 때 프로세스의 리턴 코드를 해당 리턴 코드와 비교합니다.

참고: 관리 호출은 외부 프로세스로 실행되므로 일단 시작되면 취소할 수 없습니다.

관리 호출 및 소스 전송 슬롯

에이전트에는 고급 에이전트 특성: 전송 한계에 설명된 에이전트 특성 **maxSourceTransfers**에 지정된 대로 여러 개의 소스 전송 슬롯이 포함되어 있습니다.

관리 호출 또는 관리 전송을 실행할 때마다 소스 전송 슬롯을 점유합니다. 관리 호출 또는 관리 전송이 완료되면 슬롯이 해제됩니다.

에이전트가 새 관리 호출 또는 관리 전송 요청을 수신할 때 모든 소스 전송 슬롯이 사용 중인 경우, 슬롯이 사용 가능하게 될 때까지 에이전트가 요청을 큐에 넣습니다.

관리 호출이 관리 전송을 시작하면(예: 관리 호출이 Ant 스크립트를 실행하고 해당 Ant 스크립트에서 **fte:filecopy** 또는 **fte:filemove** 태스크를 사용하여 파일 전송) 다음 두 개의 소스 전송 슬롯이 필요합니다.

- 관리 전송용으로 하나
- 관리 호출용으로 하나

이 상황에서 관리되는 전송이 완료되는 데 오랜 시간이 걸리거나 복구 모드로 전환될 경우, 두 개의 소스 전송 슬롯은 관리되는 전송이 완료되거나 취소되거나, **transferRecoveryTimeout**로 인해 시간 초과될 때까지 점유된다는 점을 유의해야 합니다. 자세한 내용은 전송 복구 시간 초과 **transferRecoveryTimeout** 개념을 참조하십시오. 잠재적으로 에이전트가 처리할 수 있는 기타 관리 전송 또는 관리 호출 수를 제한할 수 있습니다.

따라서 긴 시간 동안 소스 전송 슬롯을 점유하지 않도록 관리 호출 디자인을 고려해야 합니다.

관리되는 호출을 REST API 사용하여

HTTP의 **GET** 및 HTTP의 **POST** 메서드는 관리형 호출을 활성화하기 위해 지원되며, REST API NET Framework 버전 3에서만 작동합니다.

HTTP **DELETE** 및 HTTP **UPDATE**와 같은 다른 동사들은 지원되지 않으며, 사용 시 HTTP 405 오류 코드를 반환합니다.



주의: 관리된 통화는 일단 제출되면 을 사용하여 취소할 REST API 수 없습니다.

MFT과(와) 함께 Apache Ant 사용

Managed File Transfer는 파일 전송 기능을 Apache Ant 도구에 통합하는 데 사용할 수 있는 태스크를 제공합니다.

명령어를 **fteAnt** 사용하여 이미 구성한 환경에서 Managed File Transfer 작업을 Ant 실행할 수 있습니다. Ant 스크립트에서 제공하는 파일 전송 Ant 태스크를 사용하여 해석된 스크립팅 언어에서 복합 파일 전송 조작을 조정할 수 있습니다.

Apache Ant에 대한 자세한 정보는 Apache Ant 프로젝트 웹 페이지(<https://ant.apache.org/>)를 참조하십시오.

관련 개념

1208 페이지의 『MFT을(를) 사용하여 Ant 스크립트를 사용하여 시작하기』

스크립트를 Ant 사용하여 해석형 스크립팅 언어로 Managed File Transfer 복잡한 파일 전송 작업을 조정할 수 있습니다.

fteAnt: 작업 Ant 실행 MFT

관련 참조

1209 페이지의 『샘플 Ant 작업 MFT』

설치 시 제공되는 여러 샘플 AntManaged File Transfer 스크립트가 있습니다. 이러한 샘플은 `MQ_INSTALLATION_PATH/mqft/samples/fteant` 디렉토리에 있습니다. 각 샘플 스크립트에는 `init` 대상이 포함되어 있으며, `init` 대상에 설정된 특성을 편집하여 해당 스크립트를 구성과 함께 실행하십시오.

MFT을(를) 사용하여 Ant 스크립트를 사용하여 시작하기

스크립트를 Ant 사용하여 해석형 스크립팅 언어로 Managed File Transfer 복잡한 파일 전송 작업을 조정할 수 있습니다.

Ant 스크립트

Ant 스크립트(또는 빌드 파일)는 하나 이상의 대상을 정의하는 XML 문서입니다. 이 대상에는 실행할 태스크 요소가 포함됩니다. Managed File Transfer는 파일 전송 기능을 Apache Ant에 통합하는 데 사용할 수 있는 태스크를 제공합니다. Ant 스크립트에 대한 정보는 Apache Ant 프로젝트 웹 페이지(<https://ant.apache.org/>)를 참조하십시오.

Managed File Transfer 태스크를 사용하는 Ant 스크립트의 예는 `MQ_INSTALLATION_PATH/mqft/samples/fteant` 디렉토리에 제품 설치와 함께 제공됨

프로토콜 브릿지 에이전트에서는 Ant 스크립트가 프로토콜 브릿지 에이전트 시스템에서 실행됩니다. 이러한 Ant 스크립트에는 FTP 또는 SFTP 서버의 파일에 대한 직접 액세스 권한은 없습니다.

네임스페이스

네임스페이스는 동일한 이름을 공유할 수 있는 다른 Ant 태스크와 파일 전송 Ant 태스크를 구별하기 위해 사용됩니다. Ant 스크립트의 `project` 태그에서 네임스페이스를 정의합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs" default="do_ping">

  <target name="do_ping">
    <fte:ping cmdqm="qm@localhost@1414@SYSTEM.DEF.SVRCONN" agent="agent1@qm1"
      rcproperty="ping.rc" timeout="15"/>
  </target>

</project>
```

`xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs"` 속성은 Ant에 `com.ibm.wmqfte.ant.taskdefs` 라이브러리에서 `fte`가 접두부로 사용된 태스크의 정의를 찾으도록 지시합니다.

네임스페이스 접두부로 `fte`를 사용할 필요는 없으며 어떤 값이든 사용할 수 있습니다. 네임스페이스 접두부 `fte`는 모든 예제 및 샘플 Ant 스크립트에서 사용됩니다.

Ant 스크립트 실행

파일 전송 Ant 태스크를 포함하는 Ant 스크립트를 실행하려면 **fteAnt** 명령을 사용하십시오. 예를 들면, 다음과 같습니다.

```
fteAnt -file ant_script_location/ant_script_name
```

자세한 내용은 다음을 참조하십시오 **fteAnt:** 에서 작업 AntMFT 실행.

리턴 코드

파일 전송 Ant 태스크는 Managed File Transfer 명령과 동일한 리턴 코드를 리턴합니다. 자세한 정보는 [MFT의 리턴 코드](#)를 참조하십시오.

관련 참조

fteAnt: 작업 Ant 실행 MFT

1209 페이지의 『샘플 Ant 작업 MFT』

설치 시 제공되는 여러 샘플 AntManaged File Transfer 스크립트가 있습니다. 이러한 샘플은 `MQ_INSTALLATION_PATH/mqft/samples/fteant` 디렉토리에 있습니다. 각 샘플 스크립트에는 `init` 대상이 포함되어 있으며, `init` 대상에 설정된 특성을 편집하여 해당 스크립트를 구성과 함께 실행하십시오.

샘플 Ant 작업 MFT

설치 시 제공되는 여러 샘플 AntManaged File Transfer 스크립트가 있습니다. 이러한 샘플은 `MQ_INSTALLATION_PATH/mqft/samples/fteant` 디렉토리에 있습니다. 각 샘플 스크립트에는 `init` 대상이 포함되어 있으며, `init` 대상에 설정된 특성을 편집하여 해당 스크립트를 구성과 함께 실행하십시오.

이메일

email 샘플에서는 Ant 태스크를 사용하여 파일을 전송하고 전송이 실패하는 경우 지정된 주소로 이메일을 송신하는 방법을 보여줍니다. 스크립트는 Managed File Transfer ping 태스크를 사용하여 소스 및 목적지 에이전트가 활성 상태이고 전송을 처리할 수 있는지 검사합니다. 두 에이전트 모두 활성 상태이면 스크립트는 Managed File Transfer `fte:filecopy` 태스크를 사용하여 원래 파일을 삭제하지 않고 소스와 목적지 에이전트 간에 파일을 전송합니다. 전송이 실패하면 스크립트는 표준 Ant email 태스크를 사용하여 실패에 대한 정보가 포함된 이메일을 송신합니다.

허브(hub)

hub 샘플은 두 개의 스크립트 `hubcopy.xml` 및 `hubprocess.xml` (으)로 이루어져 있습니다. `hubcopy.xml` 스크립트는 Ant 스크립팅을 사용하여 'hub 및 spoke' 스타일 토폴로지를 빌드할 수 있는 방법을 보여줍니다. 이 샘플에서, 두 파일은 spoke 시스템에서 실행 중인 에이전트에서 hub 시스템에서 실행 중인 에이전트로 전송됩니다. 두 파일이 모두 동시에 전송되며 전송이 완료되면 `hubprocess.xml` Ant 스크립트가 hub 시스템에서 실행되어 파일을 처리합니다. 두 파일 모두 올바르게 전송되면 Ant 스크립트가 파일 콘텐츠를 병합합니다. 파일이 올바르게 전송되지 않는 경우 Ant 스크립트는 전송된 모든 파일 데이터를 삭제하여 정리합니다. 이 예가 제대로 작동하게 하려면 hub 에이전트의 명령 경로에 `hubprocess.xml` 스크립트를 배치해야 합니다. 에이전트의 명령 경로 설정에 대한 자세한 정보는 [commandPath MFT 특성](#)을 참조하십시오.

librarytransfer(IBM i 플랫폼 전용)

IBM i

 librarytransfer 샘플은 Ant 태스크를 사용하여 한 IBM i 시스템의 IBM i 라이브러리를 두 번째 IBM i 시스템에 전송하는 방법을 보여줍니다.

 라이브러리 전송 샘플은 두 IBM i 시스템 간에 네이티브 라이브러리 객체를 전송하기 위해 Managed File Transfer, 사전 정의된 Ant 작업이 포함된 IBM i 네이티브 저장 파일 지원 기능을 사용합니다. 이 샘플은 filecopy Managed File Transfer 작업 내부에 `<presrc>` 중첩 요소를 사용하여 실행 스크립트를 호출합니다. `librarysave.sh`. 이 스크립트는 소스 에이전트 시스템에서 요청된 라이브러리를 임시 저장 파일로 저장합니다. 저장 파일은 filecopy ant 작업에 의해 대상 에이전트 시스템으로 이동되며, 여기서 `<postdst>` 중첩 요소를 사용하여 실행 스크립트를 호출하여 저장 파일에 저장된 `libraryrestore.sh` 라이브러리를 대상 시스템으로 복원합니다.

 이 샘플을 실행하기 전에 `librarytransfer.xml` 파일에 설명된 대로 일부 구성을 완료해야 합니다. 또한 두 대의 IBM i 컴퓨터에 작동 가능한 Managed File Transfer 환경이 구축되어 있어야 합니다. 설정은 첫 번째 IBM i 시스템에서 실행 중인 소스 에이전트와 두 번째 IBM i 시스템에서 실행 중인 목적지 에이전트로 구성되어야 합니다. 두 에이전트가 서로 통신할 수 있도록 해야 합니다.

 librarytransfer 샘플은 다음과 같은 3개 파일로 구성됩니다.

- `librarytransfer.xml`
- `librarysave.sh`(`<presrc>` 실행 스크립트)
- `libraryrestore.sh`(`<postdst>` 실행 스크립트)

샘플 파일은 /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer 디렉토리에 있습니다.

▶ **IBM i** 이 샘플을 실행하려면 사용자가 다음 단계를 완료해야 합니다.

1. Qshell 세션을 시작하십시오. IBM i 명령 창 유형: STRQSH
2. 다음과 같이 디렉토리를 bin 디렉토리로 변경하십시오.

```
cd /QIBM/ProdData/WMQFTE/V7/bin
```

3. 필수 구성을 완료한 후 다음 명령을 사용하여 샘플을 실행하십시오.

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer/librarytransfer.xml
```

physicalfiletransfer(IBM i 플랫폼 전용)

▶ **IBM i** physicalfiletransfer 샘플은 Ant 태스크를 사용하여 한 IBM i 시스템의 라이브러리에서 두 번째 IBM i 시스템으로 소스 실제 파일 또는 데이터베이스 파일을 전송하는 방법을 보여줍니다.

▶ **IBM i** 물리적 파일 전송 샘플은 두 IBM i 시스템 간에 완전한 소스 물리적 파일 및 데이터베이스 파일을 전송하기 위해 Managed File Transfer, IBM i에서 사전 정의된 Ant 작업이 사용 가능한 의 기본 저장 파일 지원을 활용합니다. 이 샘플은 <Managed File Transfer filecopy> 작업 내부에 <presrc> 중첩 요소를 사용하여 실행 가능한 스크립트를 호출합니다 physicalfilesave.sh. 이 스크립트는 소스 에이전트 시스템의 라이브러리에서 요청된 소스 물리 파일 또는 데이터베이스 파일을 임시 저장 파일로 저장합니다. 파일 복사 앤트 태스크가 저장 파일을 대상 에이전트 시스템으로 이동합니다. 여기서 <postdst> 중첩 요소를 사용하여 실행 스크립트를 physicalfilerestore.sh 호출한 후, 저장 파일 내의 파일 객체를 대상 시스템의 지정된 라이브러리로 복원합니다.

▶ **IBM i** 이 샘플을 실행하기 전에 physicalfiletransfer.xml 파일에 설명된 대로 일부 구성을 완료해야 합니다. 또한 두 IBM i 시스템에 실행 가능한 Managed File Transfer 환경이 있어야 합니다. 설정은 첫 번째 IBM i 시스템에서 실행 중인 소스 에이전트와 두 번째 IBM i 시스템에서 실행 중인 목적지 에이전트로 구성되어야 합니다. 두 에이전트가 서로 통신할 수 있도록 해야 합니다.

▶ **IBM i** physicalfiletransfer 샘플은 다음과 같은 3개 파일로 구성됩니다.

- physicalfiletransfer.xml
- physicalfilesave.sh(<presrc> 실행 스크립트)
- physicalfilerestore.sh(<postdst> 실행 스크립트)

샘플 파일은 /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer 디렉토리에 있습니다.

▶ **IBM i** 이 샘플을 실행하려면 사용자가 다음 단계를 완료해야 합니다.

1. Qshell 세션을 시작하십시오. IBM i 명령 창 유형: STRQSH
2. 다음과 같이 디렉토리를 bin 디렉토리로 변경하십시오.

```
cd /QIBM/ProdData/WMQFTE/V7/bin
```

3. 필수 구성을 완료한 후 다음 명령을 사용하여 샘플을 실행하십시오.

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer/physicalfiletransfer.xml
```

제한시간

timeout 샘플에서는 Ant 태스크를 사용하여 파일 전송을 시도하고 지정된 제한시간 값보다 오랜 시간이 소요되는 경우 전송을 취소하는 방법을 보여 줍니다. 스크립트는 Managed File Transfer [fte:filecopy](#) 태스크를 사용하

여 파일 전송을 시작합니다. 이 전송의 결과는 지연됩니다. 스크립트는 Managed File Transfer `fte:awaitoutcome` Ant 태스크를 사용하여 지정된 시간 동안 전송이 완료되기를 대기합니다. 지정된 시간 내에 전송이 완료되지 않으면 Managed File Transfer `fte:cancel` Ant 태스크를 사용하여 파일 전송을 취소합니다.

vsamtransfer

➤ `z/OS`

➤ `z/OS` `vsamtransfer` 샘플에서는 Ant 태스크를 통해 Managed File Transfer을(를) 사용하여 VSAM 데이터 세트에서 다른 VSAM 데이터 세트로 전환하는 방법을 설명합니다. Managed File Transfer는 현재 VSAM 데이터 세트 전송을 지원하지 않습니다. 샘플 스크립트는 실행 파일 `datasetcopy.sh`을(를) 호출하기 위해 `presrc` 프로그램 호출 중첩 요소를 사용하여 순차 데이터 세트에 VSAM 데이터 레코드를 로드 해제합니다. 스크립트는 Managed File Transfer `fte:filemove` 태스크를 사용하여 순차 데이터 세트를 소스 에이전트에서 목적지 에이전트로 전송합니다. 그런 다음, 스크립트는 `postdst` 프로그램 호출 중첩 요소를 사용하여 `loadvsam.jcl` 스크립트를 호출합니다. 이 JCL 스크립트는 전송된 데이터 세트 레코드를 목적지 VSAM 데이터 세트에 로드합니다. 이 샘플에서는 목적지 호출에 JCL을 사용하여 해당 언어 옵션을 예시합니다. 두 번째 셀 스크립트를 대신 사용해도 동일한 결과를 얻을 수 있습니다.

➤ `z/OS` 이 샘플에서는 소스 및 목적지 데이터 세트가 VSAM이 아니어도 됩니다. 샘플은 소스 데이터 세트와 목적지 데이터 세트의 유형이 동일하면 모든 데이터 세트에 대해 작동합니다.

➤ `z/OS` 이 샘플이 제대로 작동하게 하려면 소스 에이전트의 명령 경로에 `datasetcopy.sh` 스크립트를 배치하고 목적지 에이전트의 명령 경로에 `loadvsam.jcl` 스크립트를 배치해야 합니다. 에이전트의 명령 경로 설정에 대한 자세한 정보는 [commandPath MFT](#) 특성을 참조하십시오.

zip

zip 샘플은 두 개의 스크립트(`zip.xml` 및 `zipfiles.xml`)로 구성됩니다. 샘플에서는 파일 전송 이동 조작을 수행하기 전에 Managed File Transfer `fte:filemove` 태스크에서 `presrc` 중첩 요소를 사용하여 Ant 스크립트를 실행하는 방법을 보여줍니다. `zip.xml` 스크립트에서 `presrc` 중첩 요소가 호출하는 `zipfiles.xml` 스크립트는 디렉토리의 콘텐츠를 압축합니다. `zip.xml` 스크립트는 압축 파일을 전송합니다. 이 샘플의 경우 소스 에이전트의 명령 경로에 `zipfiles.xml` Ant 스크립트가 있어야 합니다. 이는 `zipfiles.xml` Ant 스크립트에 소스 에이전트의 디렉토리 콘텐츠를 압축하는 데 사용되는 대상이 포함되어 있기 때문입니다. 에이전트의 명령 경로 설정에 대한 자세한 정보는 [commandPath MFT](#) 특성을 참조하십시오.

관련 개념

[1208 페이지의 『MFT을\(를\) 사용하여 Ant 스크립트를 사용하여 시작하기』](#)

스크립트를 Ant 사용하여 해석형 스크립팅 언어로 Managed File Transfer 복잡한 파일 전송 작업을 조정할 수 있습니다.

관련 참조

[fteAnt: 작업 Ant 실행 MFT](#)

사용자 엑시트를 사용하여 MFT 사용자 정의

사용자 엑시트 루틴으로 알려진 고유 프로그램을 사용하여 Managed File Transfer의 기능을 사용자 정의할 수 있습니다.

중요사항: 사용자 종료 지점 내의 모든 코드는 지원되지 IBM 않으며, 해당 코드와 관련된 모든 문제는 귀사의 기업 또는 해당 종료 지점을 제공한 공급업체가 우선적으로 조사해야 합니다.

Managed File Transfer에서는 Managed File Transfer가 사용자가 작성한 프로그램(사용자 엑시트 루틴)에 대한 제어를 전달할 수 있는 코드의 지점을 제공합니다. 이러한 지점은 사용자 엑시트 지점으로 알려져 있습니다. 그런 다음, Managed File Transfer는 프로그램이 작업을 완료했을 때 제어를 계속할 수 있습니다. 사용자 엑시트를 사용하지 않아도 되지만, Managed File Transfer 시스템의 기능을 확장하고 사용자 정의하여 사용자의 특정 요구 사항에 맞추려는 경우 이 사용자 엑시트가 유용합니다.

파일 전송 처리 중에 소스 시스템에서 사용자 엑시트를 호출할 수 있는 두 개의 지점과 파일 전송 처리 중에 목적지 시스템에서 사용자 엑시트를 호출할 수 있는 두 개의 지점이 있습니다. 다음 표에서는 이러한 사용자 엑시트 지점 각각과 엑시트 지점을 사용하기 위해 구현해야 하는 Java 인터페이스에 대한 요약を提供합니다.

표 196. 소스 측 종료점 및 목적지 측 종료점과 Java 인터페이스에 대한 요약	
종료점	구현할 Java 인터페이스
소스 측 종료점	
전체 파일 전송 시작 전	SourceTransferStartExit.java 인터페이스
전체 파일 전송 완료 후	SourceTransferEndExit.java 인터페이스
목적지 측 종료점	
전체 파일 전송 시작 전	DestinationTransferStartExit.java 인터페이스
전체 파일 전송 완료 후	DestinationTransferEndExit.java 인터페이스

다음 순서로 사용자 엑시트가 호출됩니다.

1. SourceTransferStartExit
2. DestinationTransferStartExit
3. DestinationTransferEndExit
4. SourceTransferEndExit

SourceTransferStartExit 및 DestinationTransferStartExit의 변경사항은 후속 엑시트에 입력으로 전파됩니다. 예를 들어, SourceTransferStartExit 엑시트가 전송 메타데이터를 수정하는 경우, 변경사항이 기타 엑시트에 대한 입력 전송 메타데이터에 반영됩니다.

사용자 엑시트 및 프로그램 호출은 다음 순서로 호출됩니다.

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferEnd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd).
- POST_SOURCE Command.
```

참고:

1. **DestinationTransferEndExits**는 전송이 정상적으로 또는 부분적으로 완료될 때만 실행됩니다.
2. **postDestinationCall**는 전송이 정상적으로 또는 부분적으로 완료될 때만 실행됩니다.
3. **SourceTransferEndExits**는 성공, 부분 성공 또는 전송 실패 시 실행됩니다.
4. **postSourceCall**은 다음 경우에만 호출됩니다.
 - 전송이 취소되지 않았습니다.
 - 성공 또는 부분 성공 결과가 존재합니다.
 - 모든 사후 대상 전송 프로그램이 실행되었습니다.

사용자 엑시트 빌드

사용자 엑시트를 빌드하기 위한 인터페이스는 `MQ_INSTALL_DIRECTORY/mqft/lib/com.ibm.wmqfte.exitroutines.api.jar`에 포함되어 있습니다. 엑시트를 빌드할 때 클래스 경로에 이 .jar 파일을 포함시켜야 합니다. 엑시트를 실행하려면 엑시트를 .jar 파일로 추출하고 이 .jar 파일을 다음 절에 설명된 대로 디렉토리에 저장하십시오.

사용자 엑시트 위치

사용자 엑시트 루틴은 다음 두 위치에 저장할 수 있습니다.

- exits 디렉토리. 각 에이전트 디렉토리에 exits 디렉토리가 있습니다(예: `var\mqm\mqft\config\QM_JUPITER\agents\AGENT1\exits`)

- `exitClassPath` 특성을 설정하여 대체 위치를 지정할 수 있습니다. `exits` 디렉토리 및 `exitClassPath`를 통해 설정된 클래스 경로 모두에 엑시트 클래스가 있는 경우에는 `exits` 디렉토리의 클래스가 우선합니다. 이는 두 위치에 동일한 이름의 클래스가 있으면 `exits` 디렉토리의 클래스가 우선함을 의미합니다.

사용자 엑시트를 사용하도록 에이전트 구성

4개의 에이전트 특성을 설정하여 에이전트가 호출하는 사용자 엑시트를 지정할 수 있습니다. 해당 에이전트 특성은 `sourceTransferStartExitClasses`, `sourceTransferEndExitClasses`, `destinationTransferStartExitClasses` 및 `destinationTransferEndExitClasses`입니다. 이러한 특성 사용 방법에 대한 정보는 [사용자 엑시트를 위한 MFT 에이전트 특성](#)을 참조하십시오.

프로토콜 브릿지 에이전트에서 사용자 엑시트 실행

소스 에이전트가 엑시트를 호출할 때, 전송에 대한 소스 항목의 목록을 엑시트에 전달합니다. 정상 에이전트의 경우 이것은 완전한 파일 이름의 목록입니다. 파일이 로컬(또는 마운트를 통해 액세스 가능)이어야 하기 때문에 엑시트는 파일에 액세스하고 암호화할 수 있습니다.

그러나 프로토콜 브릿지 에이전트의 경우에는 목록의 항목이 다음 형식입니다.

```
"<file server identifier>:<fully-qualified file name of the file on the remote file server>"
```

목록의 각 항목마다 엑시트는 먼저 파일 서버에 연결하고(FTP, FTPS 또는 SFTP 프로토콜을 사용하여), 파일을 다운로드하며, 로컬에서 파일을 암호화한 후 암호화한 파일을 파일 서버로 다시 업로드해야 합니다.

Connect:Direct 브릿지 에이전트에서 사용자 엑시트 실행

Connect:Direct® 브릿지 에이전트에서 사용자 엑시트를 실행할 수 없습니다.

관련 개념

[1213 페이지의 『MFT 소스 및 목적지 사용자 엑시트』](#)

[MFT 사용자 엑시트의 메타데이터](#)

[Java 사용자 종료 MFT 인터페이스](#)

관련 참조

[1218 페이지의 『MFT 사용자 엑시트를 위한 리모트 디버깅 사용』](#)

사용자 엑시트 개발 시 디버거를 사용하여 코드의 문제점을 찾을 수 있습니다.

[1218 페이지의 『샘플 MFT 소스 전송 사용자 엑시트』](#)

[1219 페이지의 『샘플 프로토콜 브릿지 신임 사용자 엑시트』](#)

[MFT 자원 모니터 사용자 엑시트](#)

[사용자 엑시트에 대한 MFT 에이전트 특성](#)

MFT 소스 및 목적지 사용자 엑시트

디렉토리 구분 기호

소스 파일 스펙의 디렉토리 구분 기호는 **fteCreateTransfer** 명령 또는 IBM MQ Explorer에서 디렉토리 구분 기호를 지정한 방법에 관계 없이 항상 슬래시(/) 문자를 사용하여 표시됩니다. 엑시트를 기록할 때 이 사항을 고려해야 합니다. 예를 들어, 다음 소스 파일이 존재하는지 확인하려는 경우: `c:\a\b.txt` 그리고 이 소스 파일을 명령어나 **fteCreateTransfer**를 사용하여 IBM MQ Explorer 지정했다면, 파일 이름이 실제로 다음과 같이 저장된다는 점에 유의하십시오: `c:/a/b.txt` 따라서 원래 문자열인 `c:\a\b.txt`을 검색하면 일치하는 항목을 찾을 수 없습니다.

소스 측 종료점

전체 파일 전송 시작 전

전송 요청이 보류 중인 전송 목록 다음에 오고 전송이 시작되려 할 때 이 엑시트가 소스 에이전트에 의해 호출됩니다.

이 종료점의 사용 예는 에이전트가 외부 명령 사용에 대한 읽기/쓰기 액세스 권한을 갖는 디렉토리로 단계별로 파일을 송신하거나 목적지 시스템에서 파일의 이름을 바꾸는 것입니다.

다음 인수를 이 엑시트에 전달하십시오.

- 소스 에이전트 이름
- 목적지 에이전트 이름
- 환경 메타데이터
- 전송 메타데이터
- 파일 스펙(파일 메타데이터 포함)

이 엑시트에서 리턴된 데이터는 다음과 같습니다.

- 업데이트된 전송 메타데이터. 입력 항목을 추가, 수정 및 삭제할 수 있습니다.
- 파일 스펙의 업데이트된 목록. 소스 파일 이름 및 목적지 파일 이름 쌍으로 구성되어 있습니다. 입력 항목을 추가, 수정 및 삭제할 수 있습니다.
- 전송을 계속할지 지정하는 표시기
- 전송 로그에 삽입할 문자열

[SourceTransferStartExit.java 인터페이스](#)를 구현하여 이 종료점에서 사용자 엑시트 코드를 호출하십시오.

전체 파일 전송 완료 후

전체 파일 전송이 완료된 후 이 엑시트가 소스 에이전트에 의해 호출됩니다.

이 종료점의 사용 예는 이메일 또는 IBM MQ 메시지를, 전송이 완료된 플래그로 보내는 등의 일부 완료 태스크를 수행하는 것입니다.

다음 인수를 이 엑시트에 전달하십시오.

- 전송 엑시트 결과
- 소스 에이전트 이름
- 목적지 에이전트 이름
- 환경 메타데이터
- 전송 메타데이터
- 파일 결과

이 엑시트에서 리턴된 데이터는 다음과 같습니다.

- 전송 로그에 삽입할 업데이트된 문자열

[SourceTransferEndExit.java 인터페이스](#)를 구현하여 이 종료점에서 사용자 엑시트 코드를 호출하십시오.

목적지 측 종료점

전체 파일 전송 시작 전

이 종료점의 사용 예는 목적지에서 권한을 유효화하는 것입니다.

다음 인수를 이 엑시트에 전달하십시오.

- 소스 에이전트 이름
- 목적지 에이전트 이름
- 환경 메타데이터
- 전송 메타데이터
- 파일 스펙

이 엑시트에서 리턴된 데이터는 다음과 같습니다.

- 목적지 파일 이름의 업데이트된 세트. 입력 항목을 수정할 수 있지만 추가 또는 삭제할 수는 없습니다.
- 전송을 계속할지 지정하는 표시기
- 전송 로그에 삽입할 문자열

[DestinationTransferStartExit.java 인터페이스](#)를 구현하여 이 종료점에서 사용자 엑시트 코드를 호출하십시오.

전체 파일 전송 완료 후

이 사용자 엑시트의 사용 예는 전송된 파일을 사용하는 배치 프로세스를 시작하거나 전송이 실패한 경우 이 메일을 보내는 것입니다.

다음 인수를 이 엑시트에 전달하십시오.

- 전송 엑시트 결과
- 소스 에이전트 이름
- 목적지 에이전트 이름
- 환경 메타데이터
- 전송 메타데이터
- 파일 결과

이 엑시트에서 리턴된 데이터는 다음과 같습니다.

- 전송 로그에 삽입할 업데이트된 문자열

[DestinationTransferEndExit.java 인터페이스](#)를 구현하여 이 종료점에서 사용자 엑시트 코드를 호출하십시오.

관련 개념

[Java 사용자 종료 MFT 인터페이스](#)

관련 참조

[1218 페이지의 『MFT 사용자 엑시트를 위한 리모트 디버깅 사용』](#)
사용자 엑시트 개발 시 디버거를 사용하여 코드의 문제점을 찾을 수 있습니다.

[1218 페이지의 『샘플 MFT 소스 전송 사용자 엑시트』](#)

[MFT 자원 모니터 사용자 엑시트](#)

MFT I/O 사용자 엑시트 사용

Managed File Transfer 전송 I/O 사용자 엑시트를 사용하면 Managed File Transfer 전송을 위한 근본적인 파일 시스템 I/O 작업을 수행하도록 사용자 정의 코드를 구성할 수 있습니다.

일반적으로 MFT 전송의 경우 에이전트는 내장 I/O 제공자 중 하나를 선택하여 전송에 적절한 파일 시스템과 상호작용합니다. 내장 I/O 제공자는 다음과 같은 유형의 파일 시스템을 지원합니다.

- 일반 UNIX 유형 및 Windows 유형 파일 시스템
-  z/OS 순차 및 파티션된 데이터 세트(z/OS에서만)
-  IBM i 고유 저장 파일(IBM i에서만)
- IBM MQ 큐
- 원격 FTP 및 SFTP 프로토콜 서버(프로토콜 브릿지 에이전트의 경우만)
- 원격 Connect:Direct 노드(Connect:Direct 브릿지 에이전트에서만)

지원되지 않거나 사용자 정의 I/O 작동이 필요한 파일 시스템의 경우 전송 I/O 사용자 엑시트를 기록할 수 있습니다.

전송 I/O 사용자 엑시트는 사용자 엑시트에 대한 기존 인프라를 사용합니다. 그러나 이러한 전송 I/O 사용자 엑시트는 다른 사용자 엑시트와 달리 각 파일의 전송 전체에서 해당 기능에 여러 번 액세스합니다.

로드할 I/O 엑시트 클래스를 지정하려면 에이전트 특성 IOExitClasses(agent.properties 파일에 있음)를 사용하십시오. 각 엑시트 클래스를 심볼로 분리하십시오. 예를 들어, 다음과 같습니다.

```
IOExitClasses=testExits.TestExit1,testExits.testExit2
```

전송 I/O 사용자 엑시트의 Java 인터페이스는 다음과 같습니다.

IOExit

I/O 엑시트 사용 여부를 판별하는 데 사용되는 기본 시작점입니다. 이 인스턴스는 IOExitPath 인스턴스의 작성을 담당합니다.

에이전트 특성 IOExitClasses의 IOExit I/O 엑시트 인터페이스만 지정해야 합니다.

IOExitPath

요약 인터페이스를 표시합니다. 예를 들어, 데이터 컨테이너를 표시하는 데이터 컨테이너 또는 와일드카드가 있습니다. 이 인터페이스를 구현하는 클래스 인스턴스를 작성할 수 없습니다. 이 인터페이스를 통해 경로를 조사하고 도출된 경로를 나열할 수 있습니다. IOExitResourcePath 및 IOExitWildcardPath 인터페이스는 IOExitPath를 확장합니다.

IOExitChannel

IOExitPath 자원에서 데이터를 읽거나 이 자원으로 데이터를 쓸 수 있습니다.

IOExitRecordChannel

레코드 중심 IOExitPath의 IOExitChannel 인터페이스를 확장하므로 여러 레코드에 있는 IOExitPath 자원에서 데이터를 읽거나 이 자원으로 데이터를 쓸 수 있습니다.

IOExitLock

공유 또는 독점 액세스를 위해 IOExitPath 자원에서 잠금을 표시합니다.

IOExitRecordResourcePath

IOExitResourcePath 인터페이스를 확장하여 레코드 지향 파일의 데이터 컨테이너를 나타냅니다(예: z/OS 데이터 세트). 인터페이스를 사용하여 데이터를 찾고 읽기 또는 쓰기 조작의 IOExitRecordChannel 인스턴스를 작성할 수 있습니다.

IOExitResourcePath

IOExitPath 인터페이스를 확장하여 데이터 컨테이너(예: 파일 또는 디렉토리)를 표시합니다. 인터페이스를 사용하여 데이터를 찾을 수 있습니다. 인터페이스가 디렉토리를 표시하는 경우 listPaths 메소드를 사용하여 경로 목록을 리턴할 수 있습니다.

IOExitWildcardPath

IOExitPath 인터페이스를 확장하여 와일드카드를 나타내는 경로를 표시합니다. 이 인터페이스를 사용하여 여러 IOExitResourcePaths를 일치시킬 수 있습니다.

IOExitProperties

Managed File Transfer이(가) I/O의 특정 측면에 대해 IOExitPath를 처리하는 방식을 판별하는 특성을 지정합니다. 예를 들어, 중간 파일을 사용할지 또는 전송이 다시 시작된 경우 처음부터 자원을 다시 읽을지 여부를 판별합니다.

관련 개념

[1211 페이지의 『사용자 엑시트를 사용하여 MFT 사용자 정의』](#)

사용자 엑시트 루틴으로 알려진 고유 프로그램을 사용하여 Managed File Transfer의 기능을 사용자 정의할 수 있습니다.

관련 참조

[IOExit.java 인터페이스](#)

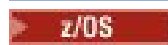
[IOExitChannel.java 인터페이스](#)

[IOExitLock.java 인터페이스](#)

[IOExitPath.java 인터페이스](#)

[IOExitProperties.java 인터페이스](#)

[IOExitRecordChannel.java 인터페이스](#)

 [IOExitRecordResourcePath.java 인터페이스](#)

[IOExitResourcePath.java 인터페이스](#)

[IOExitWildcardPath.java 인터페이스](#)

IBM i IBM i의 샘플 MFT 사용자 엑시트

Managed File Transfer는 설치와 함께 IBM i에 고유한 샘플 사용자 엑시트를 제공합니다. 샘플은 *MQMFT_install_dir/samples/ioexit-IBMi* 및 *MQMFT_install_dir/samples/userexit-IBMi* 디렉토리에 있습니다.

com.ibm.wmqfte.exit.io.ibm.qdls.FTEQDLSExit

com.ibm.wmqfte.exit.io.ibm.qdls.FTEQDLSExit 샘플 사용자 엑시트는 IBM i의 QDLS 파일 시스템에 있는 파일을 전송합니다. 엑시트가 설치되고 나면 /QDLS(으)로 시작하는 파일에 대한 모든 전송이 자동으로 엑시트를 사용합니다.

이 엑시트를 설치하려면 다음과 같은 단계를 완료하십시오.

1. com.ibm.wmqfte.samples.ibm.ioexits.jar 파일을 *WMQFTE_install_dir/samples/ioexit-IBMi* 디렉토리에서 에이전트의 exits 디렉토리로 복사하십시오.
2. com.ibm.wmqfte.exit.io.ibm.qdls.FTEQDLSExit를 IOExitClasses 특성에 추가하십시오.
3. 에이전트를 재시작하십시오.

com.ibm.wmqfte.exit.user.ibm.FileMemberMonitorExit

com.ibm.wmqfte.exit.user.ibm.FileMemberMonitorExit 샘플 사용자 엑시트는 MFT 파일 모니터와 유사하게 작동하며 IBM i 라이브러리로부터 자동으로 실제 파일 멤버를 전송합니다.

이 엑시트를 실행하려면 "library.qsys.monitor" 메타데이터 필드의 값을 지정하십시오(예: **-md** 매개변수 사용). 이 매개변수는 파일 멤버에 대한 IFS 스타일 경로를 사용하며 파일 및 멤버 와일드카드를 포함할 수 있습니다. 예: /QSYS.LIB/FOO.LIB/BAR.FILE/*.MBR, /QSYS.LIB/FOO.LIB/*.FILE/BAR.MBR, /QSYS.LIB/FOO.LIB/*.FILE/*.MBR.

이 샘플 엑시트에는 전송 중에 사용되는 이름 지정 설계를 판별하는 데 사용할 수 있는 선택적 메타데이터 필드 "naming.scheme.qsys.monitor"도 있습니다. 기본적으로 이 필드는 "unix"로 설정되며, 이로 인해 대상 파일이 F00.MBR(이)라 불립니다. IBM i FTP FILE.MEMBER 설계를 사용하기 위해 "ibmi" 값을 지정할 수도 있습니다. 예를 들어, /QSYS.LIB/FOO.LIB/BAR.FILE/BAZ.MBR은 BAR.BAZ로 전송됩니다.

이 엑시트를 설치하려면 다음과 같은 단계를 완료하십시오.

1. com.ibm.wmqfte.samples.ibm.userexits.jar 파일을 *WMQFTE_install_dir/samples/userexit-IBMi* 디렉토리에서 에이전트의 exits 디렉토리로 복사하십시오.
2. com.ibm.wmqfte.exit.user.ibm.FileMemberMonitorExit를 agent.properties 파일의 sourceTransferStartExitClasses 특성에 추가하십시오.
3. 에이전트를 재시작하십시오.

com.ibm.wmqfte.exit.user.ibm.EmptyFileDeleteExit

com.ibm.wmqfte.exit.user.ibm.EmptyFileDeleteExit 샘플 사용자 엑시트는 소스 파일 멤버가 전송의 일부로 삭제될 때 비어 있는 파일 오브젝트를 삭제합니다. IBM i 파일 오브젝트는 잠재적으로 다수의 멤버를 보유할 수 있기 때문에 파일 오브젝트는 MFT에 의해 디렉토리처럼 처리됩니다. 따라서 MFT를 사용하여 파일 오브젝트에 대해 이동 조작을 수행할 수 없으며 이동 조작은 멤버 레벨에서만 지원됩니다. 이에 따라 멤버에 대해 이동 조작을 수행하면 지금 비어 있는 파일이 남게 됩니다. 전송 요청의 일부로 이 비어 있는 파일을 삭제하려면 이 샘플 엑시트를 사용하십시오.

"empty.file.delete" 메타데이터에 대해 "true"를 지정하고 FTEFileMember를 전송하면 샘플 엑시트는 상위 파일이 비어 있는 경우 해당 파일을 삭제합니다.

이 엑시트를 설치하려면 다음과 같은 단계를 완료하십시오.

1. *WMQFTE_install_dir/samples/userexit-IBMi*에서 에이전트의 exits 디렉토리로 com.ibm.wmqfte.samples.ibm.userexits.jar 파일을 복사하십시오.
2. com.ibm.wmqfte.exit.user.ibm.EmptyFileDeleteExit를 agent.properties 파일의 sourceTransferStartExitClasses 특성에 추가하십시오.

3. 에이전트를 재시작하십시오.

관련 참조

[1215 페이지의 『MFT I/O 사용자 엑시트 사용』](#)

Managed File Transfer 전송 I/O 사용자 엑시트를 사용하면 Managed File Transfer 전송을 위한 근본적인 파일 시스템 I/O 작업을 수행하도록 사용자 정의 코드를 구성할 수 있습니다.

[사용자 엑시트에 대한 MFT 에이전트 특성](#)

MFT 사용자 엑시트를 위한 리모트 디버깅 사용

사용자 엑시트 개발 시 디버거를 사용하여 코드의 문제점을 찾을 수 있습니다.

엑시트는 에이전트를 실행하는 JVM(Java Virtual Machine) 내부에서 실행되기 때문에 일반적으로 통합 개발 환경에 포함되는 직접 디버깅 지원은 사용할 수 없습니다. 하지만 JVM의 원격 디버깅을 사용 가능하게 한 후 적합한 원격 디버거를 연결할 수 있습니다.

원격 디버깅을 사용하려면 표준 JVM 매개변수 **-Xdebug** 및 **-Xrunjdwp**을(를) 사용하십시오. 해당 특성은 **BFG_JVM_PROPERTIES** 환경 변수로 에이전트를 실행하는 JVM으로 전달됩니다. 예를 들어, AIX and Linux에서는 다음 명령이 에이전트를 시작하고 JVM이 TCP 포트 8765에서 디버거 연결을 대기하도록 합니다.

```
export BFG_JVM_PROPERTIES="-Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=8765"
fteStartAgent -F TEST_AGENT
```

디버거가 연결될 때까지 에이전트는 시작되지 않습니다. 명령 **export** 대신 Windows 명령을 **set** 사용하십시오.

디버거와 JVM 간에 다른 통신 방법을 사용할 수도 있습니다. 예를 들면, 디버거가 JVM에 대한 연결을 여는 대신 JVM이 디버거에 대한 연결을 열거나 TCP 대신 공유 메모리를 사용할 수 있습니다. 자세한 내용은 [Java Platform Debugger Architecture](#) 문서를 참조하십시오.

원격 디버그 모드에서 에이전트를 시작할 때 **-F**(포그라운드) 매개변수를 사용해야 합니다.

Eclipse 디버거 사용

다음 단계는 Eclipse 개발 환경의 원격 디버깅 기능에 적용됩니다. JPDA와 호환 가능한 다른 원격 디버거도 사용할 수 있습니다.

1. 실행 > 디버그 대화 상자 열기(또는 Eclipse 버전에 따라 실행 > 디버그 구성이나 실행 > 디버그 대화 상자)를 클릭하십시오.
2. 구성 유형 목록에서 원격 **Java 애플리케이션**을 두 번 클릭하여 디버그 구성을 작성하십시오.
3. 구성 필드를 완료하고 디버그 구성을 저장하십시오. 에이전트 JVM을 디버그 모드에서 이미 시작한 경우 즉시 JVM에 연결할 수 있습니다.

샘플 MFT 소스 전송 사용자 엑시트

```
/*
 * A Sample Source Transfer End Exit that prints information about a transfer to standard
 * output.
 * If the agent is run in the background the output will be sent to the agent's event log file.
 * If
 * the agent is started in the foreground by specifying the -F parameter on the fteStartAgent
 * command the output will be sent to the console.
 *
 * To run the exit execute the following steps:
 *
 * Compile and build the exit into a jar file. You need the following in the class path:
 * {MQ_INSTALLATION_PATH}\mqft\lib\com.ibm.wmqfte.exitroutines.api.jar
 *
 * Put the jar in your agent's exits directory:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\exits\
 *
 * Update the agent's properties file:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\agent.properties
```

```

* to include the following property:
* sourceTransferEndExitClasses=[packageName.]SampleEndExit
*
* Restart agent to pick up the exit
*
* Send the agent a transfer request:
* For example: fteCreateTransfer -sa myAgent -da YourAgent -df output.txt input.txt
*/

import java.util.List;
import java.util.Map;
import java.util.Iterator;

import com.ibm.wmqfte.exitroutine.api.SourceTransferEndExit;
import com.ibm.wmqfte.exitroutine.api.TransferExitResult;
import com.ibm.wmqfte.exitroutine.api.FileTransferResult;

public class SampleEndExit implements SourceTransferEndExit {

    public String onSourceTransferEnd(TransferExitResult transferExitResult,
        String sourceAgentName,
        String destinationAgentName,
        Map<String, String>environmentMetaData,
        Map<String, String>transferMetaData,
        List<FileTransferResult>fileResults) {

        System.out.println("Environment Meta Data: " + environmentMetaData);
        System.out.println("Transfer Meta Data: " + transferMetaData);

        System.out.println("Source agent: " +
            sourceAgentName);
        System.out.println("Destination agent: " +
            destinationAgentName);

        if (fileResults.isEmpty()) {
            System.out.println("No files in the list");
            return "No files";
        }
        else {

            System.out.println("File list: ");

            final Iterator<FileTransferResult> iterator = fileResults.iterator();

            while (iterator.hasNext()){
                final FileTransferResult thisFileSpec = iterator.next();
                System.out.println("Source file spec: " +
                    thisFileSpec.getSourceFileSpecification() +
                    ", Destination file spec: " +
                    thisFileSpec.getDestinationFileSpecification());
            }
            return "Done";
        }
    }
}

```

샘플 프로토콜 브릿지 신임 사용자 엑시트

이 샘플 사용자 엑시트의 사용 방법에 대한 정보는 [엑시트 클래스를 사용하여 파일 서버에 대한 신임 정보 매핑을 참조하십시오](#).

```

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import java.util.StringTokenizer;

import com.ibm.wmqfte.exitroutine.api.CredentialExitResult;
import com.ibm.wmqfte.exitroutine.api.CredentialExitResultCode;
import com.ibm.wmqfte.exitroutine.api.CredentialPassword;

```

```

import com.ibm.wmqfte.exitroutine.api.CredentialUserId;
import com.ibm.wmqfte.exitroutine.api.Credentials;
import com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit;

/**
 * A sample protocol bridge credential exit
 *
 * This exit reads a properties file that maps mq user ids to server user ids
 * and server passwords. The format of each entry in the properties file is:
 *
 * mqUserId=serverUserId,serverPassword
 *
 * The location of the properties file is taken from the protocol bridge agent
 * property protocolBridgeCredentialConfiguration.
 *
 * To install the sample exit compile the class and export to a jar file.
 * Place the jar file in the exits subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * In the agent.properties file of the protocol bridge agent set the
 * protocolBridgeCredentialExitClasses to SampleCredentialExit
 * Create a properties file that contains the mqUserId to serverUserId and
 * serverPassword mappings applicable to the agent. In the agent.properties
 * file of the protocol bridge agent set the protocolBridgeCredentialConfiguration
 * property to the absolute path name of this properties file.
 * To activate the changes stop and restart the protocol bridge agent.
 *
 * For further information on protocol bridge credential exits refer to
 * the WebSphere MQ Managed File Transfer documentation online at:
 * https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html
 */
public class SampleCredentialExit implements ProtocolBridgeCredentialExit {

    // The map that holds mq user ID to serverUserId and serverPassword mappings
    final private Map<String,Credentials> credentialsMap = new HashMap<String, Credentials>();

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#initialize(java.util.Map)
     */
    public synchronized boolean initialize(Map<String, String> bridgeProperties) {

        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the mq user ID mapping properties file
        final String propertiesFilePath = bridgeProperties.get("protocolBridgeCredentialConfiguration");

        if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
            // The properties file path has not been specified. Output an error and return false
            System.err.println("Error initializing SampleCredentialExit.");
            System.err.println("The location of the mqUserId mapping properties file has not been
specified in the
protocolBridgeCredentialConfiguration property");
            initialisationResult = false;
        }

        if (initialisationResult) {

            // The Properties object that holds mq user ID to serverUserId and serverPassword
            // mappings from the properties file
            final Properties mappingProperties = new Properties();

            // Open and load the properties from the properties file
            final File propertiesFile = new File (propertiesFilePath);
            FileInputStream inputStream = null;
            try {
                // Create a file input stream to the file
                inputStream = new FileInputStream(propertiesFile);

                // Load the properties from the file
                mappingProperties.load(inputStream);
            }
            catch (FileNotFoundException ex) {
                System.err.println("Error initializing SampleCredentialExit.");
                System.err.println("Unable to find the mqUserId mapping properties file: " +
propertiesFilePath);
                initialisationResult = false;
            }
            catch (IOException ex) {
                System.err.println("Error initializing SampleCredentialExit.");
                System.err.println("Error loading the properties from the mqUserId mapping properties
file: " + propertiesFilePath);
                initialisationResult = false;
            }
        }
    }
}

```

```

    }
    finally {
        // Close the inputStream
        if (inputStream != null) {
            try {
                inputStream.close();
            }
            catch (IOException ex) {
                System.err.println("Error initializing SampleCredentialExit.");
                System.err.println("Error closing the mqUserId mapping properties file: " +
propertiesFilePath);
                initialisationResult = false;
            }
        }
    }

    if (initialisationResult) {
        // Populate the map of mqUserId to server credentials from the properties
        final Enumeration<?> propertyNames = mappingProperties.propertyNames();
        while ( propertyNames.hasMoreElements()) {
            final Object name = propertyNames.nextElement();
            if (name instanceof String ) {
                final String mqUserId = ((String)name).trim();
                // Get the value and split into serverUserId and serverPassword
                final String value = mappingProperties.getProperty(mqUserId);
                final StringTokenizer valueTokenizer = new StringTokenizer(value, ",");
                String serverUserId = "";
                String serverPassword = "";
                if (valueTokenizer.hasMoreTokens()) {
                    serverUserId = valueTokenizer.nextToken().trim();
                }
                if (valueTokenizer.hasMoreTokens()) {
                    serverPassword = valueTokenizer.nextToken().trim();
                }
                // Create a Credential object from the serverUserId and serverPassword
                final Credentials credentials = new Credentials(new CredentialUserId(serverUserId), new
CredentialPassword(serverPassword));
                // Insert the credentials into the map
                credentialsMap.put(mqUserId, credentials);
            }
        }
    }

    return initialisationResult;
}
/* (non-Javadoc)
 * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#mapMQUserId(java.lang.String)
 */
public synchronized CredentialExitResult mapMQUserId(String mqUserId) {
    CredentialExitResult result = null;
    // Attempt to get the server credentials for the given mq user id
    final Credentials credentials = credentialsMap.get(mqUserId.trim());
    if ( credentials == null) {
        // No entry has been found so return no mapping found with no credentials
        result = new CredentialExitResult(CredentialExitResultCode.NO_MAPPING_FOUND, null);
    }
    else {
        // Some credentials have been found so return success to the user along with the credentials
        result = new CredentialExitResult(CredentialExitResultCode.USER_SUCCESSFULLY_MAPPED,
credentials);
    }
    return result;
}
/* (non-Javadoc)
 * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#shutdown(java.util.Map)
 */
public void shutdown(Map<String, String> bridgeProperties) {
    // Nothing to do in this method because there are no resources that need to be released
}
}

```

샘플 프로토콜 브릿지 특성 사용자 엑시트

이 샘플 사용자 엑시트의 사용 방법에 대한 정보는 [ProtocolBridgePropertiesExit2: 프로토콜 파일 서버 특성 검색을 참조하십시오.](#)

SamplePropertiesExit2.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Properties;

import com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2;
import com.ibm.wmqfte.exitroutine.api.ProtocolServerPropertyConstants;

/**
 * A sample protocol bridge properties exit. This exit reads a properties file
 * that contains properties for protocol servers.
 * <p>
 * The format of each entry in the properties file is:
 * {@literal serverName=type://host:port}
 * Ensure there is a default entry such as
 * {@literal default=type://host:port}
 * otherwise the agent will fail to start with a BFGBR0168 as it must have a
 * default server.
 * <p>
 * The location of the properties file is taken from the protocol bridge agent
 * property {@code protocolBridgePropertiesConfiguration}.
 * <p>
 * The methods {@code getCredentialLocation} returns the location of the associated
 * ProtocolBridgeCredentials.xml, this sample it is defined to be stored in a directory
 * defined by the environment variable CREDENTIALSHOME
 * <p>
 * To install the sample exit:
 * <ol>
 * <li>Compile the class and export to a jar file.
 * <li>Place the jar file in the {@code exits} subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the {@code protocolBridgePropertiesExitClasses} to
 * {@code SamplePropertiesExit2}.
 * <li>Create a properties file that contains the appropriate properties to specify the
 * required servers.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the <code>protocolBridgePropertiesConfiguration</code> property to the
 * absolute path name of this properties file.
 * <li>To activate the changes stop and restart the protocol bridge agent.
 * </ol>
 * <p>
 * For further information on protocol bridge properties exits refer to the
 * WebSphere MQ Managed File Transfer documentation online at:
 * <p>
 * {@link https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html}
 */
public class SamplePropertiesExit2 implements ProtocolBridgePropertiesExit2 {

    /**
     * Helper class to encapsulate protocol server information.
     */
    private static class ServerInformation {
        private final String type;
        private final String host;
        private final int port;

        public ServerInformation(String url) {
            int index = url.indexOf("://");
            if (index == -1) throw new IllegalArgumentException("Invalid server URL: "+url);
            type = url.substring(0, index);

            int portIndex = url.indexOf(":", index+3);
            if (portIndex == -1) {
                host = url.substring(index+3);
                port = -1;
            } else {
                host = url.substring(index+3, portIndex);
                port = Integer.parseInt(url.substring(portIndex+1));
            }
        }

        public String getType() {
            return type;
        }
    }
}
```



```

    }

    public String getHost() {
        return host;
    }

    public int getPort() {
        return port;
    }
}

/** A {@code Map} that holds information for each configured protocol server */
final private Map<String, ServerInformation> servers = new HashMap<String, ServerInformation>();

/* (non-Javadoc)
 * @see
com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#getProtocolServerProperties(java.lang.String)
 */
public Properties getProtocolServerProperties(String protocolServerName) {
    // Attempt to get the protocol server information for the given protocol server name
    // If no name has been supplied then this implies the default.
    final ServerInformation info;
    if (protocolServerName == null || protocolServerName.length() == 0) {
        protocolServerName = "default";
    }
    info = servers.get(protocolServerName);

    // Build the return set of properties from the collected protocol server information, when
    // available.
    // The properties set here is the minimal set of properties to be a valid set.
    final Properties result;
    if (info != null) {
        result = new Properties();
        result.setProperty(ProtocolServerPropertyConstants.SERVER_NAME, protocolServerName);
        result.setProperty(ProtocolServerPropertyConstants.SERVER_TYPE, info.getType());
        result.setProperty(ProtocolServerPropertyConstants.SERVER_HOST_NAME, info.getHost());
        if (info.getPort() != -1)
            result.setProperty(ProtocolServerPropertyConstants.SERVER_PORT_VALUE, ""+info.getPort());
        result.setProperty(ProtocolServerPropertyConstants.SERVER_PLATFORM, "UNIX");
        if (info.getType().toUpperCase().startsWith("FTP")) { // FTP & FTPS
            result.setProperty(ProtocolServerPropertyConstants.SERVER_TIMEZONE, "Europe/London");
            result.setProperty(ProtocolServerPropertyConstants.SERVER_LOCALE, "en-GB");
        }
        result.setProperty(ProtocolServerPropertyConstants.SERVER_FILE_ENCODING, "UTF-8");
    } else {
        System.err.println("Error no default protocol file server entry has been supplied");
        result = null;
    }

    return result;
}

/* (non-Javadoc)
 * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#initialize(java.util.Map)
 */
public boolean initialize(Map<String, String> bridgeProperties) {
    // Flag to indicate whether the exit has been successfully initialized or not
    boolean initialisationResult = true;

    // Get the path of the properties file
    final String propertiesFilePath = bridgeProperties.get("protocolBridgePropertiesConfiguration");
    if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
        // The protocol server properties file path has not been specified. Output an error and
        return false;
        System.err.println("Error initializing SamplePropertiesExit.");
        System.err.println("The location of the protocol server properties file has not been
        specified in the
        protocolBridgePropertiesConfiguration property");
        initialisationResult = false;
    }

    if (initialisationResult) {
        // The Properties object that holds protocol server information
        final Properties mappingProperties = new Properties();

        // Open and load the properties from the properties file
        final File propertiesFile = new File (propertiesFilePath);
        FileInputStream inputStream = null;
        try {
            // Create a file input stream to the file
            inputStream = new FileInputStream(propertiesFile);

```

```

        // Load the properties from the file
        mappingProperties.load(inputStream);
    } catch (final FileNotFoundException ex) {
        System.err.println("Error initializing SamplePropertiesExit.");
        System.err.println("Unable to find the protocol server properties file: " +
propertiesFilePath);
        initialisationResult = false;
    } catch (final IOException ex) {
        System.err.println("Error initializing SamplePropertiesExit.");
        System.err.println("Error loading the properties from the protocol server properties
file: " + propertiesFilePath);
        initialisationResult = false;
    } finally {
        // Close the inputStream
        if (inputStream != null) {
            try {
                inputStream.close();
            } catch (final IOException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Error closing the protocol server properties file: " +
propertiesFilePath);
                initialisationResult = false;
            }
        }
    }

    if (initialisationResult) {
        // Populate the map of protocol servers from the properties
        for (Entry<Object, Object> entry : mappingProperties.entrySet()) {
            final String serverName = (String)entry.getKey();
            final ServerInformation info = new ServerInformation((String)entry.getValue());
            servers.put(serverName, info);
        }
    }

    return initialisationResult;
}

/* (non-Javadoc)
 * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#shutdown(java.util.Map)
 */
public void shutdown(Map<String, String> bridgeProperties) {
    // Nothing to do in this method because there are no resources that need to be released
}

/* (non-Javadoc)
 * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2#getCredentialLocation()
 */
public String getCredentialLocation() {
    String envLocationPath;
    if (System.getProperty("os.name").toLowerCase().contains("win")) {
        // Windows style
        envLocationPath = "%CREDENTIALSHOME%\\ProtocolBridgeCredentials.xml";
    }
    else {
        // Unix style
        envLocationPath = "$CREDENTIALSHOME/ProtocolBridgeCredentials.xml";
    }
    return envLocationPath;
}
}

```

에이전트 명령 큐에 메시지를 추가하여 MFT 제어

에이전트 명령 큐에 메시지를 추가하여 Managed File Transfer를 제어하는 애플리케이션을 작성할 수 있습니다. 에이전트가 다음 조치 중 하나를 수행하도록 요청하기 위해 에이전트의 명령 큐에 메시지를 추가할 수 있습니다.

- 파일 전송 작성
- 스케줄된 파일 전송 작성
- 파일 전송 취소
- 스케줄된 파일 전송 취소
- 명령 호출

- 모니터 작성
- 모니터 삭제
- 에이전트가 활성 상태임을 표시하기 위해 ping 리턴

에이전트가 다음 조치 중 하나를 수행하도록 요청하려면 메시지의 형식은 다음 스키마 중 하나를 따르는 XML 형식이어야 합니다.

FileTransfer.xsd

이 형식의 메시지는 파일 전송 또는 스케줄된 파일 전송을 작성하거나, 명령을 호출하거나, 파일 전송 또는 스케줄된 파일 전송을 취소하는 데 사용할 수 있습니다. 자세한 정보는 [파일 전송 요청 메시지 형식](#)을 참조하십시오.

Monitor.xsd

이 형식의 메시지는 자원 모니터를 작성하거나 삭제하는 데 사용할 수 있습니다. 자세한 정보는 [MFT 모니터 요청 메시지 형식](#)을 참조하십시오.

PingAgent.xsd

이 형식의 메시지는 에이전트가 활성 상태인지 확인하기 위해 에이전트를 ping하는 데 사용할 수 있습니다. 자세한 정보는 [Ping MFT 에이전트 요청 메시지 형식](#)을 참조하십시오.

에이전트는 요청 메시지에 응답을 리턴합니다. 응답 메시지는 요청 메시지에 정의된 응답 큐에 추가됩니다. 응답 메시지의 형식은 다음 스키마에 의해 정의된 XML 형식입니다.

Reply.xsd

자세한 정보는 [MFT 에이전트 응답 메시지 형식](#)을 참조하십시오.

IBM MQ Telemetry용 애플리케이션 개발

IBM MQ Telemetry 응용 프로그램은 센서 및 제어 장치를 인터넷과 기업 내에서 이용 가능한 다른 정보 소스와 통합합니다.

디자인 패턴, 작업 예제, 샘플 프로그램, 프로그래밍 개념 및 참조 정보를 사용하여 IBM MQ Telemetry용 애플리케이션을 개발합니다.

관련 개념

[IBM MQ Telemetry](#)

[IBM MQ Telemetry용 사례](#)

관련 태스크

[설치 IBM MQ Telemetry](#)

[IBM MQ Telemetry 관리](#)

[IBM MQ Telemetry 문제점 해결](#)

관련 참조

[IBM MQ Telemetry 참조](#)

MQ Telemetry Transport 샘플 프로그램

샘플 MQ Telemetry Transport v3 클라이언트 애플리케이션(mqttv3app.jar)에 대해 작업하는 샘플 스크립트가 제공됩니다. IBM MQ 8.0.0 이상에서는, 샘플 클라이언트 애플리케이션이 더 이상 IBM MQ Telemetry에 포함되지 않습니다. 더 이상 사용할 수 없는 IBM Messaging Telemetry Clients SupportPac의 일부였습니다. 유사한 샘플 애플리케이션은 Eclipse Paho 및 MQTT.org에서 계속 무료로 제공됩니다.

최신 정보와 다운로드 항목은 다음 자원을 참조하십시오.

- Eclipse 파호 프로젝트 및 MQTT.org에서는 다양한 프로그래밍 언어에 대한 샘플 및 최신 텔레메트리 클라이언트를 무료로 다운로드할 수 있습니다. 이러한 사이트를 참조하여 MQ Telemetry Transport 발행 및 구독과 보안 기능 추가를 위한 샘플 프로그램을 개발할 수 있습니다.
- IBM Messaging Telemetry Clients SupportPac은 더 이상 다운로드할 수 없습니다. 이전에 다운로드 한 사본이 있는 경우 다음 콘텐츠가 있습니다.
 - IBM Messaging Telemetry Clients SupportPac의 MA9B 버전에는 컴파일된 샘플 애플리케이션(mqttv3app.jar) 및 연관된 클라이언트 라이브러리(mqttv3.jar)가 포함되어 있습니다.

- ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar
- ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar
- 이 SupportPac의 MA9C 버전에서는 /SDK/ 디렉토리 및 콘텐츠가 제거되었습니다.
- 샘플 애플리케이션(mqttv3app.jar)의 소스만 제공됩니다. 다음 디렉토리에 있습니다.

```
ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
```

- 컴파일된 클라이언트 라이브러리는 계속 제공됩니다. 다음 디렉토리에 있습니다.

```
ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
```

(더 이상 사용할 수 없는) IBM Messaging Telemetry Clients SupportPac의 사본이 있는 경우, 샘플 애플리케이션 설치 및 실행에 대한 정보는 [명령행을 사용한 IBM MQ Telemetry 설치](#)에 제공되어 있습니다.

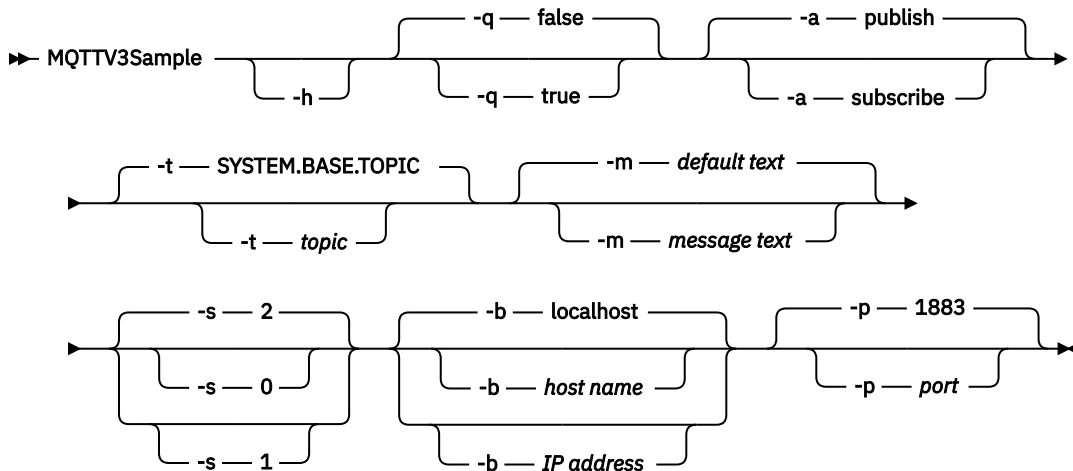
MQTTV3Sample 프로그램

MQTTV3Sample 프로그램의 샘플 구문 및 매개변수에 대한 참조 정보입니다.

목적

MQTTV3Sample 프로그램은 메시지를 발행하고 토픽을 구독하는 데 사용될 수 있습니다. 이 샘플 프로그램을 가져오는 방법에 대한 정보는 [1225 페이지의 『MQ Telemetry Transport 샘플 프로그램』](#)의 내용을 참조하십시오.

MQTTV3Sample syntax



매개변수

- h**
이 도움말 텍스트를 인쇄하고 종료합니다.
- q**
기본 모드인 false를 사용하는 대신 자동 모드를 설정합니다.
- a**
기본 조치인 발행으로 가정하는 대신 발행 또는 구독을 설정합니다.
- t**
기본 토픽을 발행하거나 구독하는 대신 토픽을 발행하거나 구독합니다.
- m**
기본 발행 텍스트인 "MQTT v3 애플리케이션 시작"을 보내는 대신 메시지 텍스트를 발행합니다.
- s**
기본 QoS인 2를 사용하는 대신 QoS를 설정합니다.

-b

기본 호스트 이름인 localhost에 연결하는 대신 이 호스트 이름 또는 IP 주소에 연결합니다.

-p

기본값 1883을 사용하는 대신 이 포트를 사용합니다.

MQTTV3Sample 프로그램 실행

Windows에서 토픽을 구독하려면 다음 명령을 사용하십시오.

```
run MQTTV3Sample -a subscribe
```

Windows에서 토픽을 발행하려면 다음 명령을 사용하십시오.

```
run MQTTV3Sample
```

MQTT 클라이언트 프로그래밍 개념

이 절에는 MQTT protocol용 클라이언트 라이브러리를 이해하는 데 도움이 되는 개념이 설명되어 있습니다. 이 개념은 클라이언트 라이브러리와 함께 제공되는 API 문서를 보완합니다.

최신 정보와 다운로드 항목은 다음 자원을 참조하십시오.

- Eclipse 파호 프로젝트 및 MQTT.org에서는 다양한 프로그래밍 언어에 대한 샘플 및 최신 텔레메트리 클라이언트를 무료로 다운로드할 수 있습니다. 이러한 사이트를 참조하여 MQ Telemetry Transport 발행 및 구독과 보안 기능 추가를 위한 샘플 프로그램을 개발할 수 있습니다.
- IBM Messaging Telemetry Clients SupportPac은 더 이상 다운로드할 수 없습니다. 이전에 다운로드 한 사본이 있는 경우 다음 콘텐츠가 있습니다.
 - IBM Messaging Telemetry Clients SupportPac의 MA9B 버전에는 컴파일된 샘플 애플리케이션(mqttv3app.jar) 및 연관된 클라이언트 라이브러리(mqttv3.jar)가 포함되어 있습니다.
 - ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar
 - ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar
 - 이 SupportPac의 MA9C 버전에서는 /SDK/ 디렉토리 및 콘텐츠가 제거되었습니다.
 - 샘플 애플리케이션(mqttv3app.jar)의 소스만 제공됩니다. 다음 디렉토리에 있습니다.

```
ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
```

- 컴파일된 클라이언트 라이브러리는 계속 제공됩니다. 다음 디렉토리에 있습니다.

```
ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
```

MQTT 클라이언트를 개발 및 실행하려면 클라이언트 디바이스에 이러한 자원을 복사하거나 설치해야 합니다. 별도의 클라이언트 런타임을 설치할 필요는 없습니다.

클라이언트용 라이선스 부여 조건은 클라이언트를 연결하고 있는 서버와 연관됩니다.

MQTT 클라이언트 라이브러리는 MQTT protocol의 참조 구현입니다. 디바이스 플랫폼에 따라 알맞은 언어로 사용자 고유의 클라이언트를 구현할 수 있습니다. [MQ Telemetry Transport 형식 및 프로토콜](#)을 참조하십시오.

API 문서에서는 클라이언트가 연결된 MQTT 서버를 가정하지 않습니다. 다른 서버에 연결하면 클라이언트 동작이 약간 다를 수 있습니다. 다음 설명은 서비스에 IBM MQ Telemetry 연결된 클라이언트의 동작을 기술합니다.

MQTT 클라이언트 애플리케이션에서 콜백 및 동기화

MQTT 클라이언트 프로그래밍 모델은 스레드를 광범위하게 사용합니다. 스레드는 서버와 주고 받는 전송 메시지의 지연에서 MQTT 클라이언트 애플리케이션을 가능한 한 많이 분리시킵니다. 출판물, 전달 토큰 및 연결 손실 이벤트는 콜백 클래스의 메서드에 전달되며, 해당 클래스는 ``를 구현합니다 MqttCallback.

콜백

참고: Eclipse Paho 웹사이트에서 최신 변경 사항을 확인하세요 `MqttCallback`. 예를 들어, `MqttCallback` 은 클라이언트의 Paho 버전에 인터페이스로 정의되고 비동기 메소드는 Paho `MqttAsyncClient` 클래스를 통해 제공됩니다.

`MqttCallback` 인터페이스에는 세 개의 콜백 메소드가 있습니다.

`connectionLost(java.lang.Throwable cause)`

`connectionLost`는 통신 오류로 연결이 중단된 경우에 호출됩니다. 또한 연결이 설정된 후에 서버에서 오류가 발생하여 서버가 연결을 중단한 경우에도 호출됩니다. 서버 오류는 큐 관리자 오류 로그에 로그됩니다. 서버는 클라이언트에 대한 연결을 중단하고 클라이언트는 `MqttCallback.connectionLost`를 호출합니다.

동일한 스레드에서 클라이언트 애플리케이션과 동일한 예외로 처리되는 유일한 리모트 오류는 `MqttClient.connect`의 예외입니다. 연결이 설정된 후 서버가 감지한 오류는 다시 `MqttCallback.connectionLost` 콜백 메소드에 `throwables`로 보고됩니다.

`connectionLost`가 발생하는 일반적인 서버 오류는 권한 부여 오류입니다. 예를 들어, 텔레메트리 서버가 토픽에 발행할 권한이 없는 클라이언트의 동작에서 토픽에 발행하려고 시도하는 경우입니다. 텔레메트리 서버로 `MQCC_FAIL` 조건 코드가 리턴되면 연결이 중단될 수 있습니다.

`deliveryComplete(IMqttDeliveryToken token)`

`deliveryComplete`는 클라이언트 애플리케이션으로 다시 전달 토큰을 전달하기 위해 MQTT 클라이언트에서 호출합니다(1233 페이지의 『전달 토큰』 참조). 콜백은 전달 토큰을 사용하여 `token.getMessage` 메소드로 발행한 메시지에 액세스할 수 있습니다.

메서드에 `deliveryComplete` 의해 호출된 애플리케이션 콜백이 클라이언트에 MQTT 제어권을 반환하면 전송이 완료됩니다. 전달이 완료될 때까지 QoS 1 또는 2의 메시지는 지속 클래스에 보유됩니다.

`deliveryComplete` 호출은 애플리케이션과 지속 클래스 사이의 동기화 지점입니다.

`deliveryComplete` 메소드는 동일한 메시지에 대해 절대 두 번 호출되지 않습니다.

애플리케이션 콜백이 클라이언트로 `deliveryCompleteMQTT` 반환되면, 클라이언트는 또는 를 QoS1 가진 메시지에 2대해 를 `MqttClientPersistence.remove` 호출합니다.

`MqttClientPersistence.remove`는 발행된 메시지의 로컬에 저장된 사본을 삭제합니다.

트랜잭션 처리 퍼스펙티브에서 `deliveryComplete` 호출은 전달을 커밋하는 단일 단계 트랜잭션입니다. 콜백 중에 처리가 실패하는 경우, 클라이언트를 다시 시작할 때

`MqttClientPersistence.remove`가 다시 호출되어 발행된 메시지의 로컬 저장 사본을 삭제합니다. 콜백은 다시 호출되지 않습니다. 전달된 메시지의 로그를 저장하기 위해 콜백을 사용하는 경우 MQTT 클라이언트와 로그를 동기화할 수 없습니다. 로그를 신뢰할 수 있도록 저장하려면 `MqttClientPersistence` 클래스의 로그를 업데이트하십시오.

전달 토큰 및 메시지는 기본 애플리케이션 스레드 및 MQTT 클라이언트에서 참조됩니다. MQTT 클라이언트는 전달이 완료되면 `MqttMessage` 오브젝트의 참조를 해제하며 클라이언트가 연결을 끊으면 전달 토큰 오브젝트의 참조를 해제합니다. 클라이언트 애플리케이션이 참조를 해제하는 경우, `MqttMessage` 오브젝트는 전달이 완료된 후 가비지 콜렉션이 수행될 수 있습니다. 전달 토큰은 세션 연결이 끊어진 후 가비지 콜렉션이 수행될 수 있습니다.

메시지가 발행된 후에 `IMqttDeliveryToken` 및 `MqttMessage` 속성을 가져올 수 있습니다. 메시지가 발행된 후 `MqttMessage` 속성을 설정하려고 시도하면 결과는 정의되지 않습니다.

클라이언트가 동일한 세션에 재연결하는 경우, 클라이언트는 MQTT 계속해서 전송

`ClientIdentifier` 확인을 처리합니다. 자세한 내용은 을 참조하십시오 1231 페이지의 『정리 세션』. MQTT 클라이언트 애플리케이션은 이전 세션에 대해 `MqttClient.CleanSession`을 `false`로 설정해야 하며 새 세션에서는 `false`로 설정해야 합니다. MQTT 클라이언트는 보류 중인 전달에 대한 새 세션에 새 전달 토큰 및 메시지 오브젝트를 작성합니다. 그러면 `MqttClientPersistence` 클래스를 사용하여 오브젝트가 복구됩니다. 애플리케이션 클라이언트에 여전히 이전 전달 토큰 및 메시지에 대한 참조가 있는 경우에는 해당 참조를 해제하십시오. 애플리케이션 콜백은 이전 세션에서 시작되고 이 세션에서 완료된 모든 전달에 대해 새 세션에서 호출됩니다.

애플리케이션 콜백은 애플리케이션 클라이언트가 연결된 후 보류 중인 전달이 완료될 때 호출됩니다. 애플리케이션 클라이언트는 연결하기 전에 `MqttClient.getPendingDeliveryTokens` 메소드를 사용하여 보류 중인 전달을 검색할 수 있습니다.

클라이언트 애플리케이션은 원래 발행된 메시지 오브젝트 및 페이로드(payload) 바이트 배열을 작성했습니다. MQTT 클라이언트는 이러한 오브젝트를 참조합니다. `token.getMessage` 메소드에서 전달 토큰이 리턴한 메시지 오브젝트는 클라이언트가 작성한 동일한 메시지 오브젝트에서 필수가 아닙니다. 새 MQTT 클라이언트 인스턴스가 전달 토큰을 작성할 경우 `MqttClientPersistence` 클래스는 `MqttMessage` 오브젝트를 다시 작성합니다. `token.isCompleted`가 `true`인 경우, 일관성 `token.getMessage`는 메시지 오브젝트가 애플리케이션 클라이언트에서 작성되었는지 또는 `MqttClientPersistence` 클래스에서 작성되었는지에 관계없이 `null`을 리턴합니다.

messageArrived(String topic, MqttMessage message)

`messageArrived` 구독 주제에 일치하는 클라이언트용 게시물이 도착할 때 호출됩니다. `topic`은 구독 필터가 아니라 발행물 토픽입니다. 필터에 와일드카드가 포함된 경우 이 둘은 다를 수 있습니다. 토픽이 클라이언트가 작성한 여러 구독에 일치하는 경우, 클라이언트는 발행물의 여러 사본을 수신합니다. 클라이언트가 구독하기도 하는 토픽에 발행하는 경우에는 자신의 발행물 사본을 수신합니다. 메시지가 또는 QoS 1로 2전송된 경우, 해당 메시지는 MQTT 클라이언트가 호출하기 전에 `MqttClientPersistence` 클래스에 `messageArrived`의해 저장됩니다. `messageArrived` 다음과 같이 동작합니다 `deliveryComplete`: 게시물당 한 번만 호출되며, 클라이언트로 MQTT 반환될 `messageArrived` 때 게시물의 로컬 `MqttClientPersistence.remove` 사본이 제거됩니다. MQTT 클라이언트는 `messageArrived`가 MQTT 클라이언트로 리턴되면 토픽 및 메시지에 대한 해당 참조를 삭제합니다. 애플리케이션 클라이언트가 오브젝트에 대한 참조를 보유하지 않는 경우, 토픽 및 메시지 오브젝트는 가비지 콜렉션됩니다.

콜백, 스레딩 및 클라이언트 애플리케이션 동기화

MQTT 클라이언트는 별도의 스레드에서 콜백 메소드를 기본 애플리케이션 스레드로 호출합니다. 클라이언트 애플리케이션은 콜백용 스레드를 작성하지 않으며 이는 MQTT 클라이언트에서 작성됩니다.

MQTT 클라이언트는 콜백 메소드를 동기화합니다. 한 번에 하나의 콜백 메소드만 실행됩니다. 동기화를 사용하면 전달된 발행물을 합산하는 오브젝트를 쉽게 업데이트할 수 있습니다. 한 번에 하나의 `MqttCallback.deliveryComplete` 인스턴스가 실행되므로 추가적인 동기화 없이 합산을 안전하게 업데이트할 수 있습니다. 한 번에 하나의 발행물만 도착하는 경우에도 해당합니다. `messageArrived` 메소드의 사용자 코드는 동기화하지 않고 오브젝트를 업데이트할 수 있습니다. 다른 스레드에서 합산 또는 업데이트 중인 오브젝트를 참조하는 경우에는 합산 또는 오브젝트를 동기화하십시오.

전달 토큰은 기본 애플리케이션 스레드와 발행물 전달 사이의 동기화 메커니즘을 제공합니다.

`token.waitForCompletion` 메소드는 특정 발행물이 전달될 때까지 또는 선택적 제한시간이 만기될 때까지 대기합니다. 다음과 같은 방법으로 `token.waitForCompletion`을 사용하여 한 번에 하나의 발행물을 처리할 수 있습니다.

`MqttCallback.deliveryComplete` 메소드와 동기화하려면 다음을 수행하십시오.

`MqttCallback.deliveryComplete`가 MQTT에 리턴될 경우에만 클라이언트가

`token.waitForCompletion`을 계속합니다. 기본 애플리케이션 스레드에서 코드를 실행하기 전에 이 메커니즘을 사용하여 `MqttCallback.deliveryComplete`의 실행 코드를 동기화할 수 있습니다.

각 발행물이 전달되기를 기다리지 않고 발행하기를 원하지만 모든 발행물이 전달되면 확인을 원하십니까? 단일 스레드에서 발행하는 경우 송신되는 마지막 발행물은 전달되는 마지막 발행물입니다.

서버로 송신된 요청 동기화

[1230 페이지](#)의 표 [197](#)에서는 서버에 요청을 송신하는 MQTT Java 클라이언트의 메소드를 설명합니다. 애플리케이션 클라이언트가 제한시간을 무제한으로 설정하지 않는 한 클라이언트는 절대 서버를 무제한으로 기다리지 않습니다. 클라이언트가 정지하면 이는 애플리케이션 프로그래밍 문제이거나 MQTT 클라이언트의 결함입니다.

표 197. 서버에 요청하는 메소드 작동 동기화		
메소드	동기화	제한시간 간격
MqttClient.Connect	서버에 대해 연결이 설정되기를 기다립니다.	기본값은 30초이며 매개변수로 설정된 경우에는 예외가 처리됩니다.
MqttClient.Disconnect	MQTT 클라이언트가 수행해야 하는 모든 작업을 마치고 TCP/IP 세션의 연결을 끊을 때까지 기다리십시오.	
MqttClient.Subscribe MqttClient.UnSubscribe	구독 또는 구독 취소 메소드가 완료되기를 기다립니다.	
MqttClient.Publish	MQTT 클라이언트에 요청을 전달한 후에 애플리케이션 스레드에 즉시 리턴됩니다.	없음
IMqttDeliveryToken.waitForCompletion	전달 토큰이 리턴될 때까지 기다립니다.	무한 또는 매개변수로 설정됩니다.

관련 개념

정리 세션

MQTT 클라이언트 및 텔레메트리(MQXR) 서비스가 세션 상태 정보를 유지보수합니다. 상태 정보는 "최소 한 번" 및 "정확히 한 번" 전달과 발행물의 "정확히 한 번" 수신을 확인하는 데 사용됩니다. 세션 상태에는 MQTT 클라이언트가 작성한 구독도 포함됩니다. 세션 간 상태 정보를 유지보수하거나 하지 않고 MQTT 클라이언트를 실행하도록 선택할 수 있습니다. 연결하기 전에 MqttConnectOptions.cleanSession 를 설정하여 세션 정리 모드를 변경하십시오.

클라이언트 ID

클라이언트 ID는 MQTT 클라이언트를 식별하는 23바이트 문자열입니다. 각 ID는 한 번에 하나의 연결 클라이언트에만 해당되어야 합니다. ID에는 쿼리 관리자 이름의 올바른 문자만 포함되어야 합니다. 이러한 제한조건 내에서 모든 ID 문자열을 사용할 수 있습니다. 클라이언트 ID를 할당하는 프로시저와 선택한 ID로 클라이언트를 구성할 수단이 있어야 합니다.

전달 토큰

이상 종료 시 메시지 발행

MQTT 클라이언트 연결이 예기치 않게 종료되면 "이상 종료 시 메시지" 발행을 송신하도록 IBM MQ Telemetry 을(를) 구성할 수 있습니다. 발행물의 컨텐츠 및 송신할 토픽을 사전정의하십시오. "이상 종료 시 메시지"는 연결 특성입니다. 클라이언트를 연결하기 전에 작성하십시오.

MQTT 클라이언트의 메시지 지속성

발행 메시지는 "최소 한 번" 또는 "정확히 한 번"의 서비스 품질(QoS)로 송신되는 경우에 지속됩니다. 클라이언트에 사용자 고유의 지속 메커니즘을 구현할 수도 있고 클라이언트가 제공하는 기본 지속 메커니즘을 사용할 수도 있습니다. 지속성은 클라이언트로 송신되거나 클라이언트에서 수신된 발행에 대해 양방향으로 작용합니다.

발행물

발행물은 토픽 문자열과 연관된 MqttMessage 의 인스턴스입니다. MQTT 클라이언트는 IBM MQ에 송신할 발행물을 작성하고 IBM MQ에 대한 토픽을 구독하여 발행물을 수신할 수 있습니다.

MQTT 클라이언트에서 제공하는 서비스 품질(QoS)

MQTT 클라이언트는 IBM MQ 및 MQTT 클라이언트에 발행물을 전달하는 것에 대해 세 개의 서비스 품질(QoS)을 제공합니다("최대 한 번", "최소 한 번" 및 "정확히 한 번"). MQTT 클라이언트가 IBM MQ에 요청을 송신하여 구독을 작성하는 경우 "최소 한 번" 서비스 품질(QoS)을 사용하여 요청이 송신됩니다.

보유된 발행 및 MQTT 클라이언트

한 토픽에는 보유된 발행물이 하나만 있을 수 있습니다. 보유된 발행물이 있는 토픽에 대한 구독을 작성하는 경우 발행물이 즉시 사용자에게 전달됩니다.

구독

토픽 필터를 사용하여 관심있는 발행 토픽을 등록하려면 구독을 작성하십시오. 클라이언트는 다중 구독 또는 와일드카드를 사용하는 토픽 필터를 포함하는 구독을 작성하여 관심있는 다중 토픽을 등록할 수 있습니다. 필터와 일치하는 토픽의 발행물이 클라이언트로 송신됩니다. 구독은 클라이언트 연결이 끊긴 중에도 활성 상태로 남아 있을 수 있습니다. 발행물은 클라이언트가 다시 연결할 때 여기에 송신됩니다.

MQTT 클라이언트의 토픽 문자열 및 토픽 필터

토픽 문자열 및 토픽 필터는 발행과 구독에 사용됩니다. MQTT 클라이언트의 토픽 문자열 및 필터 구문은 IBM MQ의 토픽 문자열과 거의 동일합니다.

정리 세션

MQTT 클라이언트 및 텔레메트리(MQXR) 서비스가 세션 상태 정보를 유지보수합니다. 상태 정보는 "최소 한 번" 및 "정확히 한 번" 전달과 발행물의 "정확히 한 번" 수신을 확인하는 데 사용됩니다. 세션 상태에는 MQTT 클라이언트가 작성한 구독도 포함됩니다. 세션 간 상태 정보를 유지보수하거나 하지 않고 MQTT 클라이언트를 실행하도록 선택할 수 있습니다. 연결하기 전에 `MqttConnectOptions.cleanSession` 를 설정하여 세션 정리 모드를 변경하십시오.

`MqttClient.connect` 메소드를 사용하여 MQTT 클라이언트 애플리케이션을 연결할 때 클라이언트는 서버의 주소 및 클라이언트 ID를 사용하여 연결을 식별합니다. 서버는 이전에 서버에 연결된 세션의 정보가 저장되었는지 확인합니다. 이전 세션이 여전히 있으며 `cleanSession=true`인 경우, 클라이언트와 서버에서 이전 세션 정보가 삭제됩니다. `cleanSession=false`인 경우에는 이전 세션이 재개됩니다. 이전 세션이 없는 경우에는 새 세션이 시작됩니다.

참고: IBM MQ 관리자가 열린 세션을 강제로 닫고 모든 세션 정보를 삭제할 수 있습니다. 클라이언트가 세션을 `cleanSession=false`로 다시 여는 경우에는 새 세션이 시작됩니다.

발행물

기본 `MqttConnectOptions`를 사용하거나 클라이언트에 연결하기 전에

`MqttConnectOptions.cleanSession`을 `true`로 설정한 경우에는 클라이언트가 연결할 때 해당 클라이언트에 대해 보류 중인 모든 발행물 전달이 제거됩니다.

정리 세션 설정은 `QoS=0`으로 송신된 발행물에는 아무런 영향을 미치지 않습니다. `QoS=1` 및 `QoS=2`의 경우 `cleanSession=true`를 사용하면 발행물이 손실될 수 있습니다.

구독

기본값을 `MqttConnectOptions` 사용하거나 클라이언트 연결 전에

`MqttConnectOptions.cleanSession`로 `true` 설정하면, 클라이언트가 연결될 때 해당 클라이언트의 기존 구독이 모두 삭제됩니다. 세션 중에 클라이언트가 만드는 모든 새 구독은 연결을 해제할 때 삭제됩니다.

연결하기 전에 `cleanSession`로 `false` 설정하면 `MqttConnectOptions.cleanSession`, 클라이언트가 생성하는 모든 구독은 연결 전 해당 클라이언트에 존재하던 모든 구독에 추가됩니다. 클라이언트가 연결이 끊길 때 모든 구독은 활성인 상태로 남아 있습니다.

`cleanSession` 속성이 구독에 영향을 미치는 방식을 이해하는 다른 방법은 모달 속성으로 간주하는 것입니다. 기본 모드 `cleanSession=true`에서 클라이언트를 구독을 작성하고 세션의 범위 내에서만 발행물을 수신합니다. 대체 모드 `cleanSession=false`에서 구독은 지속됩니다. 클라이언트는 연결하거나 연결을 끊을 수 있고 해당 구독은 활성인 상태로 남아 있습니다. 클라이언트가 다시 연결하면 전달되지 않은 모든 발행물을 수신합니다. 연결된 동안 대신 활성인 구독 세트를 수정할 수 있습니다.

연결하기 전에 모드를 `cleanSession` 설정해야 하며, 이 모드는 세션이 끝날 때까지 유지됩니다. 설정을 변경하려면 클라이언트의 연결을 끊은 후 다시 연결해야 합니다. `cleanSession=true` 모드를 `cleanSession=false`로 변경하면, 해당 클라이언트의 기존 모든 구독과 아직 수신되지 않은 모든 간행물이 삭제됩니다.

관련 개념

MQTT 클라이언트 애플리케이션에서 콜백 및 동기화

MQTT 클라이언트 프로그래밍 모델은 스레드를 광범위하게 사용합니다. 스레드는 서버와 주고 받는 전송 메시지의 지연에서 MQTT 클라이언트 애플리케이션을 가능한 한 많이 분리시킵니다. 출판물, 전달 토큰 및 연결 손실 이벤트는 콜백 클래스의 메서드에 전달되며, 해당 클래스는 ``를 구현합니다 MqttCallback.

클라이언트 ID

클라이언트 ID는 MQTT 클라이언트를 식별하는 23바이트 문자열입니다. 각 ID는 한 번에 하나의 연결 클라이언트에만 해당되어야 합니다. ID에는 큐 관리자 이름의 올바른 문자만 포함되어야 합니다. 이러한 제한조건 내에서 모든 ID 문자열을 사용할 수 있습니다. 클라이언트 ID를 할당하는 프로시저와 선택한 ID로 클라이언트를 구성할 수단이 있어야 합니다.

전달 토큰

이상 종료 시 메시지 발행

MQTT 클라이언트 연결이 예기치 않게 종료되면 "이상 종료 시 메시지" 발행을 송신하도록 IBM MQ Telemetry 을(를) 구성할 수 있습니다. 발행물의 콘텐츠 및 송신할 토픽을 사전정의하십시오. "이상 종료 시 메시지"는 연결 특성입니다. 클라이언트를 연결하기 전에 작성하십시오.

MQTT 클라이언트의 메시지 지속성

발행 메시지는 "최소 한 번" 또는 "정확히 한 번"의 서비스 품질(QoS)로 송신되는 경우에 지속됩니다. 클라이언트에 사용자 고유의 지속 메커니즘을 구현할 수도 있고 클라이언트가 제공하는 기본 지속 메커니즘을 사용할 수도 있습니다. 지속성은 클라이언트로 송신되거나 클라이언트에서 수신된 발행에 대해 양방향으로 작용합니다.

발행물

발행물은 토픽 문자열과 연관된 MqttMessage 의 인스턴스입니다. MQTT 클라이언트는 IBM MQ에 송신할 발행물을 작성하고 IBM MQ에 대한 토픽을 구독하여 발행물을 수신할 수 있습니다.

MQTT 클라이언트에서 제공하는 서비스 품질(QoS)

MQTT 클라이언트는 IBM MQ 및 MQTT 클라이언트에 발행물을 전달하는 것에 대해 세 개의 서비스 품질(QoS)을 제공합니다("최대 한 번", "최소 한 번" 및 "정확히 한 번"). MQTT 클라이언트가 IBM MQ에 요청을 송신하여 구독을 작성하는 경우 "최소 한 번" 서비스 품질(QoS)을 사용하여 요청이 송신됩니다.

보유된 발행 및 MQTT 클라이언트

한 토픽에는 보유된 발행물이 하나만 있을 수 있습니다. 보유된 발행물이 있는 토픽에 대한 구독을 작성하는 경우 발행물이 즉시 사용자에게 전달됩니다.

구독

토픽 필터를 사용하여 관심있는 발행 토픽을 등록하려면 구독을 작성하십시오. 클라이언트는 다중 구독 또는 와일드카드를 사용하는 토픽 필터를 포함하는 구독을 작성하여 관심있는 다중 토픽을 등록할 수 있습니다. 필터와 일치하는 토픽의 발행물이 클라이언트로 송신됩니다. 구독은 클라이언트 연결이 끊긴 중에도 활성 상태로 남아 있을 수 있습니다. 발행물은 클라이언트가 다시 연결할 때 여기에 송신됩니다.

MQTT 클라이언트의 토픽 문자열 및 토픽 필터

토픽 문자열 및 토픽 필터는 발행과 구독에 사용됩니다. MQTT 클라이언트의 토픽 문자열 및 필터 구문은 IBM MQ의 토픽 문자열과 거의 동일합니다.

클라이언트 ID

클라이언트 ID는 MQTT 클라이언트를 식별하는 23바이트 문자열입니다. 각 ID는 한 번에 하나의 연결 클라이언트에만 해당되어야 합니다. ID에는 큐 관리자 이름의 올바른 문자만 포함되어야 합니다. 이러한 제한조건 내에서 모든 ID 문자열을 사용할 수 있습니다. 클라이언트 ID를 할당하는 프로시저와 선택한 ID로 클라이언트를 구성할 수단이 있어야 합니다.

클라이언트 ID는 MQTT 시스템 관리에 사용됩니다. 관리해야 하는 클라이언트가 수십만에 이를 수 있는 만큼 특정 클라이언트를 빠르게 식별할 수 있어야 합니다. 예를 들어, 디바이스가 오작동 상태이며 고객이 헬프데스크에 문의해서 사용자가 알림을 받았다고 가정하십시오. 고객은 디바이스를 식별할 수 있어야 하며 사용자는 일반적으로 클라이언트에 연결된 서버와 해당 식별을 상관시킬 수 있어야 합니다.

MQTT 클라이언트 연결을 통해 찾아볼 때 각 연결은 클라이언트 ID로 레이블이 지정됩니다. 이 ID를 디바이스 및 서버로 매핑할 최상의 방법을 결정하는 데 도움이 되도록 다음 질문에 답하십시오.

- 각 디바이스를 클라이언트 ID 및 서버로 매핑하는 데이터베이스를 유지보수하고 사용하는 것이 편리합니까?
- 디바이스의 이름으로 접속된 서버를 식별할 수 있습니까?
- 클라이언트 ID를 실제 디바이스에 매핑하는 검색 테이블이 필요합니까?
- 클라이언트 ID로 특정 디바이스, 사용자 또는 클라이언트에서 실행 중인 애플리케이션을 식별할 수 있습니까?

- 고객이 결합이 있는 디바이스를 새 디바이스로 교체하는 경우 새 디바이스의 ID가 이전 디바이스와 동일하거나 새 ID를 할당합니까? (물리적 디바이스를 변경하고 동일한 ID를 유지하면 미해결 발행 및 활성 구독이 자동으로 새 디바이스에 전송됩니다.)

또한 클라이언트 ID가 고유한지 확인할 시스템이 필요하며 클라이언트에 ID를 설정하기 위한 신뢰할 수 있는 프로세스가 있어야 합니다. 클라이언트 디바이스가 사용자 인터페이스가 없는 "블랙 박스"인 경우에는, 클라이언트 ID를 사용하여 디바이스를 제조하거나 활성화되기 전에 디바이스를 구성하는 소프트웨어 설치 및 구성 프로세스를 사용할 수 있습니다.

ID를 짧고 고유하게 유지하기 위해 48비트 디바이스 MAC 주소로부터 클라이언트 ID를 생성할 수 있습니다. 전송 크기가 중요한 문제가 아니면 나머지 17바이트를 사용하여 주소를 보다 쉽게 관리할 수 있습니다.

관련 개념

MQTT 클라이언트 애플리케이션에서 콜백 및 동기화

MQTT 클라이언트 프로그래밍 모델은 스레드를 광범위하게 사용합니다. 스레드는 서버와 주고 받는 전송 메시지의 지연에서 MQTT 클라이언트 애플리케이션을 가능한 한 많이 분리시킵니다. 출판물, 전달 토큰 및 연결 손실 이벤트는 콜백 클래스의 메서드에 전달되며, 해당 클래스는 ``를 구현합니다 `MqttCallback`.

정리 세션

MQTT 클라이언트 및 텔레메트리(MQXR) 서비스가 세션 상태 정보를 유지보수합니다. 상태 정보는 "최소 한 번" 및 "정확히 한 번" 전달과 발행물의 "정확히 한 번" 수신을 확인하는 데 사용합니다. 세션 상태에는 MQTT 클라이언트가 작성한 구독도 포함됩니다. 세션 간 상태 정보를 유지보수하거나 하지 않고 MQTT 클라이언트를 실행하도록 선택할 수 있습니다. 연결하기 전에 `MqttConnectOptions.cleanSession`를 설정하여 세션 정리 모드를 변경하십시오.

전달 토큰

이상 종료 시 메시지 발행

MQTT 클라이언트 연결이 예기치 않게 종료되면 "이상 종료 시 메시지" 발행을 송신하도록 IBM MQ Telemetry을(를) 구성할 수 있습니다. 발행물의 콘텐츠 및 송신할 토픽을 사전정의하십시오. "이상 종료 시 메시지"는 연결 특성입니다. 클라이언트를 연결하기 전에 작성하십시오.

MQTT 클라이언트의 메시지 지속성

발행 메시지는 "최소 한 번" 또는 "정확히 한 번"의 서비스 품질(QoS)로 송신되는 경우에 지속됩니다. 클라이언트에 사용자 고유의 지속 메커니즘을 구현할 수도 있고 클라이언트가 제공하는 기본 지속 메커니즘을 사용할 수도 있습니다. 지속성은 클라이언트로 송신되거나 클라이언트에서 수신된 발행에 대해 양방향으로 작용합니다.

발행물

발행물은 토픽 문자열과 연관된 `MqttMessage`의 인스턴스입니다. MQTT 클라이언트는 IBM MQ에 송신할 발행물을 작성하고 IBM MQ에 대한 토픽을 구독하여 발행물을 수신할 수 있습니다.

MQTT 클라이언트에서 제공하는 서비스 품질(QoS)

MQTT 클라이언트는 IBM MQ 및 MQTT 클라이언트에 발행물을 전달하는 것에 대해 세 개의 서비스 품질(QoS)을 제공합니다("최대 한 번", "최소 한 번" 및 "정확히 한 번"). MQTT 클라이언트가 IBM MQ에 요청을 송신하여 구독을 작성하는 경우 "최소 한 번" 서비스 품질(QoS)을 사용하여 요청이 송신됩니다.

보유된 발행 및 MQTT 클라이언트

한 토픽에는 보유된 발행물이 하나만 있을 수 있습니다. 보유된 발행물이 있는 토픽에 대한 구독을 작성하는 경우 발행물이 즉시 사용자에게 전달됩니다.

구독

토픽 필터를 사용하여 관심있는 발행 토픽을 등록하려면 구독을 작성하십시오. 클라이언트는 다중 구독 또는 와일드카드를 사용하는 토픽 필터를 포함하는 구독을 작성하여 관심있는 다중 토픽을 등록할 수 있습니다. 필터와 일치하는 토픽의 발행물이 클라이언트로 송신됩니다. 구독은 클라이언트 연결이 끊긴 중에도 활성 상태로 남아 있을 수 있습니다. 발행물은 클라이언트가 다시 연결할 때 여기에 송신됩니다.

MQTT 클라이언트의 토픽 문자열 및 토픽 필터

토픽 문자열 및 토픽 필터는 발행과 구독에 사용됩니다. MQTT 클라이언트의 토픽 문자열 및 필터 구문은 IBM MQ의 토픽 문자열과 거의 동일합니다.

전달 토큰

클라이언트가 토픽에 대해 발행하면 새 전달 토큰이 작성됩니다. 발행의 전달을 모니터링하거나 전달이 완료될 때까지 클라이언트 애플리케이션을 차단하는 데 전달 토큰을 사용하십시오.

토큰은 `MqttDeliveryToken` 오브젝트입니다. `MqttTopic.publish()` 메소드를 호출함으로써 작성되며 클라이언트 세션 연결이 끊기고 전달이 완료될때까지 MQTT 클라이언트가 보유합니다.

토큰은 일반적으로 전달이 완료되었는지 확인하는 데 사용됩니다. 리턴된 토큰을 통해 `token.waitForCompletion`을 호출하여 전달이 완료될 때까지 클라이언트 애플리케이션을 차단하십시오. 또는 `MqttCallback` 핸들러를 제공하십시오. 발행 전달의 한 부분으로서 MQTT 클라이언트가 기대하고 있던 모든 수신확인을 수신하면 전달 토큰을 매개변수로 전달하면서 `MqttCallback.deliveryComplete`를 호출합니다.

전달이 완료될 때까지 `token.getMessage`를 호출하여 리턴된 전달 토큰을 통해 발행물을 조사할 수 있습니다.

완료된 전달

전달 완료는 비동기식이며 발행과 연관된 서비스의 질에 따라 다릅니다.

최대 한 번

`QoS=0`

전달은 `MqttTopic.publish`에서 리턴되는 즉시 완료됩니다.

`MqttCallback.deliveryComplete`가 즉시 호출됩니다.

최소 한 번

`QoS=1`

큐 관리자로부터 발행에 대한 수신확인이 수신되면 전달은 완료됩니다. 수신확인이 수신되면

`MqttCallback.deliveryComplete`가 호출됩니다. 통신이 느리거나 불안한 경우

`MqttCallback.deliveryComplete`가 호출되기 전에 이 메시지가 두 번 이상 전달될 수 있습니다.

정확히 한 번

`QoS=2`

발행물이 구독자에게 발행되었다는 완료 메시지를 클라이언트가 수신하면 전달이 완료됩니다.

`MqttCallback.deliveryComplete`는 발행 메시지가 수신되는 즉시 호출됩니다. 완료 메시지를 기다리지 않습니다.

MQTT 클라이언트가 `MqttCallback.deliveryComplete`에서 정상적으로 리턴되지 않는 경우도 드물게 있습니다. `MqttCallback.deliveryComplete`가 호출되었으므로 전달이 완료된 것을 알 수 있습니다. 클라이언트가 동일한 세션을 다시 시작하는 경우 `MqttCallback.deliveryComplete`는 다시 호출되지 않습니다.

완료되지 않은 전달

클라이언트 세션 연결이 끊긴 후에 전달이 완료되지 않으면 클라이언트를 다시 연결해 전달을 완료할 수 있습니다. `MqttConnectionOptions` 속성이 `false`로 설정된 세션에서 발행된 메시지만 메시지 전달을 완료할 수 있습니다.

동일한 클라이언트 ID와 서버 주소를 사용하여 클라이언트를 작성한 다음 `cleanSession`

`MqttConnectionOptions` 속성을 `false`로 설정하여 다시 연결하십시오. `cleanSession`을 `true`로 설정하면 보류 중인 전달 토큰은 삭제됩니다.

`MqttClient.getPendingDeliveryTokens`를 호출하여 보류 중인 전달이 있는지 검사할 수 있습니다. 클라이언트를 연결하기 전에 `MqttClient.getPendingDeliveryTokens`를 호출할 수 있습니다.

관련 개념

[MQTT 클라이언트 애플리케이션에서 콜백 및 동기화](#)

MQTT 클라이언트 프로그래밍 모델은 스레드를 광범위하게 사용합니다. 스레드는 서버와 주고 받는 전송 메시지의 지연에서 MQTT 클라이언트 애플리케이션을 가능한 한 많이 분리시킵니다. 출판물, 전달 토큰 및 연결 손실 이벤트는 콜백 클래스의 메서드에 전달되며, 해당 클래스는 ``를 구현합니다 `MqttCallback`.

정리 세션

MQTT 클라이언트 및 텔레메트리(MQXR) 서비스가 세션 상태 정보를 유지보수합니다. 상태 정보는 "최소 한 번" 및 "정확히 한 번" 전달과 발행물의 "정확히 한 번" 수신을 확인하는 데 사용됩니다. 세션 상태에는 MQTT 클라이언트가 작성한 구독도 포함됩니다. 세션 간 상태 정보를 유지보수하거나 하지 않고 MQTT 클라이언트를 실행하

도록 선택할 수 있습니다. 연결하기 전에 `MqttConnectOptions.cleanSession` 를 설정하여 세션 정리 모드를 변경하십시오.

클라이언트 ID

클라이언트 ID는 MQTT 클라이언트를 식별하는 23바이트 문자열입니다. 각 ID는 한 번에 하나의 연결 클라이언트에만 해당되어야 합니다. ID에는 쿼리 관리자 이름의 올바른 문자만 포함되어야 합니다. 이러한 제한조건 내에서 모든 ID 문자열을 사용할 수 있습니다. 클라이언트 ID를 할당하는 프로시저와 선택한 ID로 클라이언트를 구성할 수단이 있어야 합니다.

이상 종료 시 메시지 발행

MQTT 클라이언트 연결이 예기치 않게 종료되면 "이상 종료 시 메시지" 발행을 송신하도록 IBM MQ Telemetry 을(를) 구성할 수 있습니다. 발행물의 콘텐츠 및 송신할 토픽을 사전정의하십시오. "이상 종료 시 메시지"는 연결 특성입니다. 클라이언트를 연결하기 전에 작성하십시오.

MQTT 클라이언트의 메시지 지속성

발행 메시지는 "최소 한 번" 또는 "정확히 한 번"의 서비스 품질(QoS)로 송신되는 경우에 지속됩니다. 클라이언트에 사용자 고유의 지속 메커니즘을 구현할 수도 있고 클라이언트가 제공하는 기본 지속 메커니즘을 사용할 수도 있습니다. 지속성은 클라이언트로 송신되거나 클라이언트에서 수신된 발행에 대해 양방향으로 작용합니다.

발행물

발행물은 토픽 문자열과 연관된 `MqttMessage` 의 인스턴스입니다. MQTT 클라이언트는 IBM MQ에 송신할 발행물을 작성하고 IBM MQ에 대한 토픽을 구독하여 발행물을 수신할 수 있습니다.

MQTT 클라이언트에서 제공하는 서비스 품질(QoS)

MQTT 클라이언트는 IBM MQ 및 MQTT 클라이언트에 발행물을 전달하는 것에 대해 세 개의 서비스 품질(QoS)을 제공합니다("최대 한 번", "최소 한 번" 및 "정확히 한 번"). MQTT 클라이언트가 IBM MQ에 요청을 송신하여 구독을 작성하는 경우 "최소 한 번" 서비스 품질(QoS)을 사용하여 요청이 송신됩니다.

보유된 발행 및 MQTT 클라이언트

한 토픽에는 보유된 발행물이 하나만 있을 수 있습니다. 보유된 발행물이 있는 토픽에 대한 구독을 작성하는 경우 발행물이 즉시 사용자에게 전달됩니다.

구독

토픽 필터를 사용하여 관심있는 발행 토픽을 등록하려면 구독을 작성하십시오. 클라이언트는 다중 구독 또는 와일드카드를 사용하는 토픽 필터를 포함하는 구독을 작성하여 관심있는 다중 토픽을 등록할 수 있습니다. 필터와 일치하는 토픽의 발행물이 클라이언트로 송신됩니다. 구독은 클라이언트 연결이 끊긴 중에도 활성 상태로 남아 있을 수 있습니다. 발행물은 클라이언트가 다시 연결할 때 여기에 송신됩니다.

MQTT 클라이언트의 토픽 문자열 및 토픽 필터

토픽 문자열 및 토픽 필터는 발행과 구독에 사용됩니다. MQTT 클라이언트의 토픽 문자열 및 필터 구문은 IBM MQ의 토픽 문자열과 거의 동일합니다.

이상 종료 시 메시지 발행

MQTT 클라이언트 연결이 예기치 않게 종료되면 "이상 종료 시 메시지" 발행을 송신하도록 IBM MQ Telemetry 을(를) 구성할 수 있습니다. 발행물의 콘텐츠 및 송신할 토픽을 사전정의하십시오. "이상 종료 시 메시지"는 연결 특성입니다. 클라이언트를 연결하기 전에 작성하십시오.

이상 종료 시 메시지의 토픽을 작성하십시오. `MQTTManagement/Connections/server URI/client identifier/Lost`와 같은 토픽을 작성할 수 있습니다.

`MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)` 메소드를 사용해 "이상 종료 시 메시지"를 설정하십시오.

`lastWillPayload` 메시지에 시간소인을 작성하는 것을 고려하십시오. 클라이언트 및 연결 상황을 식별하는데 도움이 되는 기타 클라이언트 정보를 포함시키십시오. `MqttConnectionOptions` 오브젝트를 `MqttClient` 구성자로 전달하십시오.

`lastWillQos`를 1 또는 2로 설정하여 IBM MQ에서 메시지를 지속적으로 만들고 전달을 보장하십시오. 마지막으로 손실된 연결 정보를 보유하려면 `lastWillRetained`를 `true`로 설정하십시오.

예상치 못하게 연결이 종료될 경우 "이상 종료 시 메시지" 발행이 구독자에게 전송됩니다. 이 발행물은 클라이언트가 `MqttClient.disconnect` 메소드를 호출하지 않은 채로 연결이 종료되는 경우에 송신됩니다.

연결을 모니터하려면 연결과 계획된 연결 종료를 기록하기 위한 기타 발행으로 "이상 종료 시 메시지" 발행을 보완하십시오.

관련 개념

MQTT 클라이언트 애플리케이션에서 콜백 및 동기화

MQTT 클라이언트 프로그래밍 모델은 스레드를 광범위하게 사용합니다. 스레드는 서버와 주고 받는 전송 메시지의 지연에서 MQTT 클라이언트 애플리케이션을 가능한 한 많이 분리시킵니다. 출판물, 전달 토큰 및 연결 손실 이벤트는 콜백 클래스의 메서드에 전달되며, 해당 클래스는 ``를 구현합니다 `MqttCallback`.

정리 세션

MQTT 클라이언트 및 텔레메트리(MQXR) 서비스가 세션 상태 정보를 유지보수합니다. 상태 정보는 "최소 한 번" 및 "정확히 한 번" 전달과 발행물의 "정확히 한 번" 수신을 확인하는 데 사용됩니다. 세션 상태에는 MQTT 클라이언트가 작성한 구독도 포함됩니다. 세션 간 상태 정보를 유지보수하거나 하지 않고 MQTT 클라이언트를 실행하도록 선택할 수 있습니다. 연결하기 전에 `MqttConnectOptions.cleanSession` 를 설정하여 세션 정리 모드를 변경하십시오.

클라이언트 ID

클라이언트 ID는 MQTT 클라이언트를 식별하는 23바이트 문자열입니다. 각 ID는 한 번에 하나의 연결 클라이언트에만 해당되어야 합니다. ID에는 쿼리 관리자 이름의 올바른 문자만 포함되어야 합니다. 이러한 제한조건 내에서 모든 ID 문자열을 사용할 수 있습니다. 클라이언트 ID를 할당하는 프로시저와 선택한 ID로 클라이언트를 구성할 수단이 있어야 합니다.

전달 토큰

MQTT 클라이언트의 메시지 지속성

발행 메시지는 "최소 한 번" 또는 "정확히 한 번"의 서비스 품질(QoS)로 송신되는 경우에 지속됩니다. 클라이언트에 사용자 고유의 지속 메커니즘을 구현할 수도 있고 클라이언트가 제공하는 기본 지속 메커니즘을 사용할 수도 있습니다. 지속성은 클라이언트로 송신되거나 클라이언트에서 수신된 발행에 대해 양방향으로 작용합니다.

발행물

발행물은 토픽 문자열과 연관된 `MqttMessage` 의 인스턴스입니다. MQTT 클라이언트는 IBM MQ에 송신할 발행물을 작성하고 IBM MQ에 대한 토픽을 구독하여 발행물을 수신할 수 있습니다.

MQTT 클라이언트에서 제공하는 서비스 품질(QoS)

MQTT 클라이언트는 IBM MQ 및 MQTT 클라이언트에 발행물을 전달하는 것에 대해 세 개의 서비스 품질(QoS)을 제공합니다("최대 한 번", "최소 한 번" 및 "정확히 한 번"). MQTT 클라이언트가 IBM MQ에 요청을 송신하여 구독을 작성하는 경우 "최소 한 번" 서비스 품질(QoS)을 사용하여 요청이 송신됩니다.

보유된 발행 및 MQTT 클라이언트

한 토픽에는 보유된 발행물이 하나만 있을 수 있습니다. 보유된 발행물이 있는 토픽에 대한 구독을 작성하는 경우 발행물이 즉시 사용자에게 전달됩니다.

구독

토픽 필터를 사용하여 관심있는 발행 토픽을 등록하려면 구독을 작성하십시오. 클라이언트는 다중 구독 또는 와일드카드를 사용하는 토픽 필터를 포함하는 구독을 작성하여 관심있는 다중 토픽을 등록할 수 있습니다. 필터와 일치하는 토픽의 발행물이 클라이언트로 송신됩니다. 구독은 클라이언트 연결이 끊긴 중에도 활성 상태로 남아 있을 수 있습니다. 발행물은 클라이언트가 다시 연결할 때 여기에 송신됩니다.

MQTT 클라이언트의 토픽 문자열 및 토픽 필터

토픽 문자열 및 토픽 필터는 발행과 구독에 사용됩니다. MQTT 클라이언트의 토픽 문자열 및 필터 구문은 IBM MQ의 토픽 문자열과 거의 동일합니다.

MQTT 클라이언트의 메시지 지속성

발행 메시지는 "최소 한 번" 또는 "정확히 한 번"의 서비스 품질(QoS)로 송신되는 경우에 지속됩니다. 클라이언트에 사용자 고유의 지속 메커니즘을 구현할 수도 있고 클라이언트가 제공하는 기본 지속 메커니즘을 사용할 수도 있습니다. 지속성은 클라이언트로 송신되거나 클라이언트에서 수신된 발행에 대해 양방향으로 작용합니다.

MQTT에서 메시지 지속성은 메시지가 전송되는 방법과 메시지가 IBM MQ에서 지속 메시지로 큐에 대기되는지 여부라는 두 가지 측면을 가지고 있습니다.

1. MQTT 클라이언트는 메시지 지속성과 서비스 품질을 결합시킵니다. 메시지에 대해 선택하는 서비스 품질(QoS)에 따라 메시지가 지속됩니다. 메시지 지속성은 요구되는 서비스 품질을 구현하기 위해서 필요합니다.

"최대 한 번"(QoS=0)을 지정하면 클라이언트는 메시지가 발행되는 즉시 이를 제거합니다. 메시지의 업스트림 처리에 장애가 있는 경우 메시지는 다시 송신되지 않습니다. 클라이언트가 활성 상태로 남아 있는 경우에도 메시지는 다시 송신되지 않습니다. QoS=0 메시지의 동작은 IBM MQ 빠른 비지속 메시지와 동일합니다.

QoS 1 또는 2로 클라이언트가 메시지를 발행하는 경우에는 지속됩니다. 메시지는 로컬에 저장되며 "최소한 한 번", QoS=1 또는 "정확히 한 번", QoS=2 전달을 위해 더 이상 필요하지 않은 경우에만 클라이언트에서 제거합니다.

2. 메시지가 QoS 1 또는 2로 표시되는 경우 해당 메시지는 IBM MQ에서 지속 메시지로 큐에 대기됩니다. 메시지가 QoS=0으로 표시된 경우 해당 메시지는 IBM MQ에서 비지속 메시지로 큐에 대기됩니다. IBM MQ 비지속성 메시지는 메시지 채널에 NPMSPEED 속성이 설정된 FAST 경우를 제외하고 큐 관리자 간에 "정확히 한 번만" 전송됩니다.

지속 발행물은 클라이언트 애플리케이션에서 수신될 때까지 클라이언트에 저장됩니다. QoS=2의 경우 애플리케이션 콜백이 제어를 리턴하면 발행물이 클라이언트로부터 제거됩니다. QoS=1의 경우 애플리케이션은 장애 발생 시 발행물을 다시 수신할 수 있습니다. QoS=0의 경우 콜백은 발행을 한 번 이상 수신하지 않습니다. 장애가 발생한 경우나 발행될 때 클라이언트의 연결이 끊기는 경우 발행을 수신하지 않을 수 있습니다.

토픽을 구독하는 경우 구독자가 지속 용량에 일치하는 메시지를 수신하도록 QoS를 줄일 수 있습니다. 더 높은 QoS에서 작성된 발행물은 구독자가 요청한 가장 높은 QoS로 송신됩니다.

메시지 저장

작은 디바이스에서 데이터 스토리지 구현은 매우 다양합니다. MQTT 클라이언트가 관리하는 스토리지에 지속 메시지를 임시로 저장하는 모델은 지나치게 느리거나 지나치게 많은 스토리지를 요구할 수 있습니다. 모바일 디바이스에서 모바일 운영 체제는 MQTT 메시지에 이상적인 스토리지 서비스를 제공할 수 있습니다.

소형 디바이스의 제한조건을 만족시키는 유연성을 제공하기 위해 MQTT 클라이언트에는 두 가지 지속 인터페이스가 있습니다. 이러한 인터페이스는 지속 메시지 저장을 포함하는 조작을 정의합니다. 이 인터페이스는 MQTT client for Java에 대한 API 문서에 설명되어 있습니다. MQTT 클라이언트 라이브러리의 클라이언트 API 문서 링크는 [MQTT 클라이언트 프로그래밍 참조](#)를 참조하십시오. 사용자는 디바이스에 맞춰 인터페이스를 구현할 수 있습니다. SE에서 Java 실행되는 클라이언트는 MQTT 파일 시스템에 지속적 메시지를 저장하는 인터페이스의 기본 구현체를 가지고 있습니다. java.io 패키지를 사용합니다.

지속 클래스

MqttClientPersistence

클라이언트에 MQTT 대한 생성자의 MqttClientPersistenceMqttClient 매개변수로 구현체의 인스턴스를 전달하십시오. MqttClient 생성자에서 MqttClientPersistence 매개변수를 생략할 경우 MQTT 클라이언트는 MqttDefaultFilePersistence 클래스를 사용하여 지속 메시지를 저장합니다.

MqttPersistable

MqttClientPersistence는 스토리지 키를 사용하여 MqttPersistable 오브젝트를 가져오고 넣습니다. MqttDefaultFilePersistence를 사용하지 않는 경우에는 MqttClientPersistence 구현과 함께 MqttPersistable 구현도 제공해야 합니다.

MqttDefaultFilePersistence

MQTT 클라이언트는 MqttDefaultFilePersistence 클래스를 제공합니다. 사용자의 클라이언트 애플리케이션에서 MqttDefaultFilePersistence를 인스턴스화하는 경우, 지속 메시지를 저장할 디렉토리를 MqttDefaultFilePersistence 구성자의 매개변수로 제공할 수 있습니다.

또는 MQTT 클라이언트가 MqttDefaultFilePersistence를 인스턴스화하고 다음 기본 디렉토리에 파일을 배치할 수 있습니다.

client identifier -tcp hostname portnumber

다음 문자는 디렉토리 이름 문자열에서 제거됩니다.

"\", "\\\", \"/\", \":\" 및 " "

디렉토리 경로는 시스템 특성 `rcp.data`의 값입니다. `rcp.data`이(가) 설정되지 않은 경우 경로는 시스템 특성 `usr.data`의 값입니다. 여기서,

- `rcp.data`은(는) OSGi 또는 Eclipse RCP(Rich Client Platform)의 설치와 연관된 특성입니다.
- `usr.data`은(는) 애플리케이션을 시작한 Java 명령이 실행된 디렉토리입니다.

관련 개념

MQTT 클라이언트 애플리케이션에서 콜백 및 동기화

MQTT 클라이언트 프로그래밍 모델은 스레드를 광범위하게 사용합니다. 스레드는 서버와 주고 받는 전송 메시지의 지연에서 MQTT 클라이언트 애플리케이션을 가능한 한 많이 분리시킵니다. 출판물, 전달 토큰 및 연결 손실 이벤트는 콜백 클래스의 메서드에 전달되며, 해당 클래스는 ``를 구현합니다 `MqttCallback`.

정리 세션

MQTT 클라이언트 및 텔레메트리(MQXR) 서비스가 세션 상태 정보를 유지보수합니다. 상태 정보는 "최소 한 번" 및 "정확히 한 번" 전달과 발행물의 "정확히 한 번" 수신을 확인하는 데 사용합니다. 세션 상태에는 MQTT 클라이언트가 작성한 구독도 포함됩니다. 세션 간 상태 정보를 유지보수하거나 하지 않고 MQTT 클라이언트를 실행하도록 선택할 수 있습니다. 연결하기 전에 `MqttConnectOptions.cleanSession`를 설정하여 세션 정리 모드를 변경하십시오.

클라이언트 ID

클라이언트 ID는 MQTT 클라이언트를 식별하는 23바이트 문자열입니다. 각 ID는 한 번에 하나의 연결 클라이언트에만 해당되어야 합니다. ID에는 쿼리 관리자 이름의 올바른 문자만 포함되어야 합니다. 이러한 제한조건 내에서 모든 ID 문자열을 사용할 수 있습니다. 클라이언트 ID를 할당하는 프로시저와 선택한 ID로 클라이언트를 구성할 수단이 있어야 합니다.

전달 토큰

이상 종료 시 메시지 발행

MQTT 클라이언트 연결이 예기치 않게 종료되면 "이상 종료 시 메시지" 발행을 송신하도록 IBM MQ Telemetry을(를) 구성할 수 있습니다. 발행물의 콘텐츠 및 송신할 토픽을 사전정의하십시오. "이상 종료 시 메시지"는 연결 특성입니다. 클라이언트를 연결하기 전에 작성하십시오.

발행물

발행물은 토픽 문자열과 연관된 `MqttMessage`의 인스턴스입니다. MQTT 클라이언트는 IBM MQ에 송신할 발행물을 작성하고 IBM MQ에 대한 토픽을 구독하여 발행물을 수신할 수 있습니다.

MQTT 클라이언트에서 제공하는 서비스 품질(QoS)

MQTT 클라이언트는 IBM MQ 및 MQTT 클라이언트에 발행물을 전달하는 것에 대해 세 개의 서비스 품질(QoS)을 제공합니다("최대 한 번", "최소 한 번" 및 "정확히 한 번"). MQTT 클라이언트가 IBM MQ에 요청을 송신하여 구독을 작성하는 경우 "최소 한 번" 서비스 품질(QoS)을 사용하여 요청이 송신됩니다.

보유된 발행 및 MQTT 클라이언트

한 토픽에는 보유된 발행물이 하나만 있을 수 있습니다. 보유된 발행물이 있는 토픽에 대한 구독을 작성하는 경우 발행물이 즉시 사용자에게 전달됩니다.

구독

토픽 필터를 사용하여 관심있는 발행 토픽을 등록하려면 구독을 작성하십시오. 클라이언트는 다중 구독 또는 와일드카드를 사용하는 토픽 필터를 포함하는 구독을 작성하여 관심있는 다중 토픽을 등록할 수 있습니다. 필터와 일치하는 토픽의 발행물이 클라이언트로 송신됩니다. 구독은 클라이언트 연결이 끊긴 중에도 활성 상태로 남아 있을 수 있습니다. 발행물은 클라이언트가 다시 연결할 때 여기에 송신됩니다.

MQTT 클라이언트의 토픽 문자열 및 토픽 필터

토픽 문자열 및 토픽 필터는 발행과 구독에 사용됩니다. MQTT 클라이언트의 토픽 문자열 및 필터 구문은 IBM MQ의 토픽 문자열과 거의 동일합니다.

발행물

발행물은 토픽 문자열과 연관된 `MqttMessage`의 인스턴스입니다. MQTT 클라이언트는 IBM MQ에 송신할 발행물을 작성하고 IBM MQ에 대한 토픽을 구독하여 발행물을 수신할 수 있습니다.

`MqttMessage`에는 페이로드(payload)로 바이트 배열이 있습니다. 메시지를 가능한 한 작게 유지하십시오. MQTT protocol에서 허용하는 메시지의 최대 길이는 250MB입니다.

일반적으로 MQTT 클라이언트 프로그램은 `java.lang.String` 또는 `java.lang.StringBuffer`를 사용하여 메시지 콘텐츠를 조정합니다. 편의를 위해 `MqttMessage` 클래스에는 페이로드를 문자열로 변환하는 `toString` 메소드가 있습니다. `java.lang.String` 또는 `java.lang.StringBuffer`에서 바이트 배열 페이로드를 작성하려면 `getBytes` 메소드를 사용하십시오.

`getBytes` 메소드는 문자열을 해당 플랫폼의 기본 문자 세트로 변환합니다. 기본 문자 세트는 일반적으로 UTF-8입니다. 텍스트만 포함된 MQTT 발행물은 보통 UTF-8로 인코딩됩니다. `getBytes("UTF8")` 메소드를 사용하여 기본 문자 세트를 대체하십시오.

IBM MQ에서는 MQTT 발행물이 `jms-bytes` 메시지로 수신됩니다. 메시지는 `<mqtt>` 및 `<mmps>` 폴더가 포함된 `MQRFH2` 폴더를 포함합니다. `<mqtt>` 폴더에는 `clientId`, `msgId` 및 `qos`가 포함되어 있지만 이 콘텐츠는 나중에 변경될 수 있습니다.

`MqttMessage`에는 QoS(quality of service), 보유 여부 및 중복 여부의 세 가지 추가 속성이 있습니다. 중복 플래그는 서비스 품질(QoS)이 "최소 한 번" 또는 "정확히 한 번"인 경우에만 설정됩니다. 메시지가 이전에 송신되었으나 MQTT 클라이언트에 의해 신속하게 수신확인되지 않은 경우 중복 속성이 `true`로 설정된 채 메시지가 다시 송신됩니다.

발행

MQTT 클라이언트 애플리케이션에서 발행물을 작성하려면 `MqttMessage`를 작성하십시오. 해당 페이로드, 서비스 품질 및 보유 여부를 설정하고 `MqttTopic.publish(MqttMessage message)` 메소드를 호출하십시오. `MqttDeliveryToken`이 리턴되며 발행 완료는 비동기입니다.

또는 MQTT 클라이언트가 구독 작성 시 `MqttTopic.publish(byte [] payload, int qos, boolean retained)` 메소드의 매개변수로부터 사용자를 위한 임시 메시지 오브젝트를 작성할 수 있습니다.

발행의 QoS(Quality of Service)가 "최소 한 번" 또는 "정확히 한 번", QoS=1 또는 QoS=2인 경우, MQTT 클라이언트는 `MqttClientPersistence` 인터페이스를 호출합니다. `MqttClientPersistence`를 호출하면 전달 토큰을 애플리케이션에 전달하기 전에 메시지가 저장됩니다.

애플리케이션은 메시지가 서버에 전달될 때까지 `MqttDeliveryToken.waitForCompletion` 메소드를 사용하여 블록을 선택할 수 있습니다. 또는 애플리케이션이 블로킹하지 않고 계속 진행할 수도 있습니다. 발행물이 차단 없이 전달되는지 확인하려면 MQTT 클라이언트를 사용하여 `MqttCallback`를 구현하는 콜백 클래스의 인스턴스를 등록하십시오. MQTT 클라이언트는 발행이 전달되면 즉시 `MqttCallback.deliveryComplete` 메소드를 호출합니다. 서비스 품질(QoS)에 따라 QoS=0인 경우에는 거의 즉시 전달되며 QoS=2인 경우에는 시간이 걸릴 수 있습니다.

전달이 완료되면 `MqttDeliveryToken.isComplete` 메소드를 사용하여 폴링하십시오.

`MqttDeliveryToken.isComplete`의 값이 `false`이면 `MqttDeliveryToken.getMessage`를 호출하여 메시지 콘텐츠를 가져올 수 있습니다. `MqttDeliveryToken.isComplete`를 호출한 결과가 `true`이면 메시지가 제거되었으며 `MqttDeliveryToken.getMessage`를 호출하면 널 포인터 예외가 전달됩니다.

`MqttDeliveryToken.getMessage`와 `MqttDeliveryToken.isComplete` 사이에는 내장 동기화가 없습니다.

보류 중인 전달 토큰을 모두 수신하기 전에 클라이언트가 연결을 끊는 경우, 연결하기 전에 클라이언트의 새 인스턴스가 보류 중인 전달 토큰을 조회할 수 있습니다. 클라이언트가 연결할 때까지 새 전달은 완료되지 않으므로 `MqttDeliveryToken.getMessage`를 호출하는 것이 안전합니다. `MqttDeliveryToken.getMessage` 메소드를 사용하여 전달되지 않은 발행물을 알아내십시오. `MqttConnectOptions.cleanSession`을 기본 값인 `true`로 설정하여 연결하는 경우에는 보류 중인 전달 토큰이 제거됩니다.

구독

큐 관리자가 MQTT 구독자에게 송신할 발행물을 작성하는 작업을 담당합니다. 큐 관리자는 MQTT 클라이언트에 의해 작성된 구독의 토픽 필터가 발행의 토픽 문자열과 일치하는지 검사합니다. 일치하는 정확한 일치일 수도 있고 일치에 와일드카드가 포함될 수도 있습니다. 큐 관리자는 발행물을 구독자로 전달하기 전에 발행물과 연관된 토픽 속성을 확인합니다. 또한 관리 토픽 오브젝트가 사용자에게 구독 권한을 부여하는지 식별하기 위해 와일드카드 문자가 포함된 토픽 문자열을 사용한 구독에 설명된 검색 프로시저를 수행합니다.

MQTT 클라이언트가 "최소 한 번" 서비스 품질인 발행을 수신하면 `MqttCallback.messageArrived` 메소드를 호출하여 발행을 처리합니다. 발행의 서비스 품질이 "정확히 한 번" QoS=2인 경우 MQTT 클라이언트는 메시지가 수신 시 메시지를 저장하기 위해 `MqttClientPersistence` 인터페이스를 호출합니다. 그런 다음 `MqttCallback.messageArrived`를 호출합니다.

관련 개념

MQTT 클라이언트 애플리케이션에서 콜백 및 동기화

MQTT 클라이언트 프로그래밍 모델은 스레드를 광범위하게 사용합니다. 스레드는 서버와 주고 받는 전송 메시지의 지연에서 MQTT 클라이언트 애플리케이션을 가능한 한 많이 분리시킵니다. 출판물, 전달 토큰 및 연결 손실 이벤트는 콜백 클래스의 메서드에 전달되며, 해당 클래스는 ``를 구현합니다 `MqttCallback`.

정리 세션

MQTT 클라이언트 및 텔레메트리(MQXR) 서비스가 세션 상태 정보를 유지보수합니다. 상태 정보는 "최소 한 번" 및 "정확히 한 번" 전달과 발행물의 "정확히 한 번" 수신을 확인하는 데 사용됩니다. 세션 상태에는 MQTT 클라이언트가 작성한 구독도 포함됩니다. 세션 간 상태 정보를 유지보수하거나 하지 않고 MQTT 클라이언트를 실행하도록 선택할 수 있습니다. 연결하기 전에 `MqttConnectOptions.cleanSession`를 설정하여 세션 정리 모드를 변경하십시오.

클라이언트 ID

클라이언트 ID는 MQTT 클라이언트를 식별하는 23바이트 문자열입니다. 각 ID는 한 번에 하나의 연결 클라이언트에만 해당되어야 합니다. ID에는 쿼리 관리자 이름의 올바른 문자만 포함되어야 합니다. 이러한 제한조건 내에서 모든 ID 문자열을 사용할 수 있습니다. 클라이언트 ID를 할당하는 프로시저와 선택한 ID로 클라이언트를 구성할 수단이 있어야 합니다.

전달 토큰

이상 종료 시 메시지 발행

MQTT 클라이언트 연결이 예기치 않게 종료되면 "이상 종료 시 메시지" 발행을 송신하도록 IBM MQ Telemetry을(를) 구성할 수 있습니다. 발행물의 콘텐츠 및 송신할 토픽을 사전정의하십시오. "이상 종료 시 메시지"는 연결 특성입니다. 클라이언트를 연결하기 전에 작성하십시오.

MQTT 클라이언트의 메시지 지속성

발행 메시지는 "최소 한 번" 또는 "정확히 한 번"의 서비스 품질(QoS)로 송신되는 경우에 지속됩니다. 클라이언트에 사용자 고유의 지속 메커니즘을 구현할 수도 있고 클라이언트가 제공하는 기본 지속 메커니즘을 사용할 수도 있습니다. 지속성은 클라이언트로 송신되거나 클라이언트에서 수신된 발행에 대해 양방향으로 작용합니다.

MQTT 클라이언트에서 제공하는 서비스 품질(QoS)

MQTT 클라이언트는 IBM MQ 및 MQTT 클라이언트에 발행물을 전달하는 것에 대해 세 개의 서비스 품질(QoS)을 제공합니다("최대 한 번", "최소 한 번" 및 "정확히 한 번"). MQTT 클라이언트가 IBM MQ에 요청을 송신하여 구독을 작성하는 경우 "최소 한 번" 서비스 품질(QoS)을 사용하여 요청이 송신됩니다.

보유된 발행 및 MQTT 클라이언트

한 토픽에는 보유된 발행물이 하나만 있을 수 있습니다. 보유된 발행물이 있는 토픽에 대한 구독을 작성하는 경우 발행물이 즉시 사용자에게 전달됩니다.

구독

토픽 필터를 사용하여 관심있는 발행 토픽을 등록하려면 구독을 작성하십시오. 클라이언트는 다중 구독 또는 와일드카드를 사용하는 토픽 필터를 포함하는 구독을 작성하여 관심있는 다중 토픽을 등록할 수 있습니다. 필터와 일치하는 토픽의 발행물이 클라이언트로 송신됩니다. 구독은 클라이언트 연결이 끊긴 중에도 활성 상태로 남아 있을 수 있습니다. 발행물은 클라이언트가 다시 연결할 때 여기에 송신됩니다.

MQTT 클라이언트의 토픽 문자열 및 토픽 필터

토픽 문자열 및 토픽 필터는 발행과 구독에 사용됩니다. MQTT 클라이언트의 토픽 문자열 및 필터 구문은 IBM MQ의 토픽 문자열과 거의 동일합니다.

MQTT 클라이언트에서 제공하는 서비스 품질(QoS)

MQTT 클라이언트는 IBM MQ 및 MQTT 클라이언트에 발행물을 전달하는 것에 대해 세 개의 서비스 품질(QoS)을 제공합니다("최대 한 번", "최소 한 번" 및 "정확히 한 번"). MQTT 클라이언트가 IBM MQ에 요청을 송신하여 구독을 작성하는 경우 "최소 한 번" 서비스 품질(QoS)을 사용하여 요청이 송신됩니다.

발행의 서비스 품질은 `MqttMessage`의 속성입니다. 이 속성은 `MqttMessage.setQos` 메소드로 설정됩니다.

메소드 `MqttClient.subscribe`는 토픽에서 클라이언트에 송신된 발행에 적용된 서비스 품질을 낮출 수 있습니다. 구독자에게 전달된 발행의 서비스 품질은 발행의 서비스 품질에 따라 달라질 수 있습니다. 두 값 중 낮은 값이 발행물을 전달하는 데 사용됩니다.

최대 한 번 **QoS=0**

메시지는 최대 한 번 전달되거나 전혀 전달되지 않습니다. 이 네트워크 간 전달은 수신확인되지 않습니다. 메시지는 저장되지 않습니다. 클라이언트 연결이 끊어지거나 서버가 실패하는 경우 메시지는 손실될 수 있습니다.

QoS=0은 가장 빠른 전송 모드입니다. 이 모드는 "실행 후 삭제"라고도 합니다.

MQTT protocol에서는 서버가 QoS=0의 발행물을 클라이언트에 전달하지 않아도 됩니다. 서버가 발행물을 수신할 때 클라이언트 연결이 끊어지는 경우 서버에 따라 발행물을 제거할 수 있습니다. 텔레메트리 (MQXR) 서비스에서는 QoS=0으로 송신된 메시지를 제거하지 않습니다. 해당 메시지는 비지속 메시지로 저장되고 큐 관리자가 중지되는 경우에만 제거됩니다.

최소 한 번 **QoS=1**

QoS=1은 기본 전송 모드입니다.

메시지는 항상 최소 한 번 전달됩니다. 송신자가 수신확인을 수신하지 않는 경우, 메시지는 수신확인이 수신될 때까지 DUP 플래그가 설정되어 다시 송신됩니다. 따라서 수신자에게 동일한 메시지가 여러 번 전송되고 이를 여러 번 처리할 수도 있습니다.

메시지가 처리될 때까지 송신자와 수신자는 메시지를 로컬에 저장해야 합니다.

메시지는 처리된 후에 수신자로부터 삭제됩니다. 수신자가 브로커인 경우, 메시지는 구독자에게 발행됩니다. 수신자가 클라이언트인 경우 메시지는 구독자 애플리케이션에게로 전달됩니다. 메시지가 삭제된 후 수신자는 송신자에게 수신확인을 송신합니다.

수신자로부터 수신확인을 받고 나면 송신자로부터 메시지가 삭제됩니다.

정확히 한 번 **QoS=2**

메시지는 항상 정확히 한 번만 전송됩니다.

메시지가 처리될 때까지 송신자와 수신자는 메시지를 로컬에 저장해야 합니다.

QoS=2는 가장 안전하지만 가장 느린 전송 모드입니다. 메시지가 송신자에서 삭제되기 전에 송신자와 수신자 사이에 최소 두 쌍의 전송이 발생합니다. 메시지는 첫 번째 전송 후에 수신자 측에서 처리될 수 있습니다.

첫 번째 전송 쌍에서 송신자는 메시지를 전송하고 수신자에게서 메시지를 저장했다는 수신확인을 받습니다. 송신자가 수신확인을 수신하지 않는 경우, 메시지는 수신확인이 수신될 때까지 DUP 플래그가 설정되어 다시 송신됩니다.

두 번째 전송 쌍에서 송신자는 수신자에게 메시지 "PUBREL"의 처리를 완료할 수 있다고 전달합니다. 송신자가 "PUBREL" 메시지의 수신확인을 수신하지 않은 경우 수신확인이 수신될 때까지 "PUBREL" 메시지가 다시 송신됩니다. 송신자는 "PUBREL" 메시지에 대한 수신확인을 수신하면 저장했던 메시지를 삭제합니다.

메시지를 다시 처리하지 않는다는 가정 하에 수신자가 첫 번째 또는 두 번째 단계에서 메시지를 처리할 수 있습니다. 수신자가 브로커인 경우 이는 메시지를 구독자에게 발행합니다. 수신자가 클라이언트인 경우 메시지는 구독자 애플리케이션으로 전달됩니다. 수신자는 송신자에게 메시지 처리가 완료되었다는 완료 메시지를 송신합니다.

관련 개념

MQTT 클라이언트 애플리케이션에서 콜백 및 동기화

MQTT 클라이언트 프로그래밍 모델은 스레드를 광범위하게 사용합니다. 스레드는 서버와 주고 받는 전송 메시지의 지연에서 MQTT 클라이언트 애플리케이션을 가능한 한 많이 분리시킵니다. 출판물, 전달 토큰 및 연결 손실 이벤트는 콜백 클래스의 메서드에 전달되며, 해당 클래스는 ``를 구현합니다 `MqttCallback`.

정리 세션

MQTT 클라이언트 및 텔레메트리(MQXR) 서비스가 세션 상태 정보를 유지보수합니다. 상태 정보는 "최소 한 번" 및 "정확히 한 번" 전달과 발행물의 "정확히 한 번" 수신을 확인하는 데 사용됩니다. 세션 상태에는 MQTT 클라이언트가 작성한 구독도 포함됩니다. 세션 간 상태 정보를 유지보수하거나 하지 않고 MQTT 클라이언트를 실행하

도록 선택할 수 있습니다. 연결하기 전에 `MqttConnectOptions.cleanSession` 를 설정하여 세션 정리 모드를 변경하십시오.

클라이언트 ID

클라이언트 ID는 MQTT 클라이언트를 식별하는 23바이트 문자열입니다. 각 ID는 한 번에 하나의 연결 클라이언트에만 해당되어야 합니다. ID에는 쿼리 관리자 이름의 올바른 문자만 포함되어야 합니다. 이러한 제한조건 내에서 모든 ID 문자열을 사용할 수 있습니다. 클라이언트 ID를 할당하는 프로시저와 선택한 ID로 클라이언트를 구성할 수단이 있어야 합니다.

전달 토큰

이상 종료 시 메시지 발행

MQTT 클라이언트 연결이 예기치 않게 종료되면 "이상 종료 시 메시지" 발행을 송신하도록 IBM MQ Telemetry 을(를) 구성할 수 있습니다. 발행물의 콘텐츠 및 송신할 토픽을 사전정의하십시오. "이상 종료 시 메시지"는 연결 특성입니다. 클라이언트를 연결하기 전에 작성하십시오.

MQTT 클라이언트의 메시지 지속성

발행 메시지는 "최소 한 번" 또는 "정확히 한 번"의 서비스 품질(QoS)로 송신되는 경우에 지속됩니다. 클라이언트에 사용자 고유의 지속 메커니즘을 구현할 수도 있고 클라이언트가 제공하는 기본 지속 메커니즘을 사용할 수도 있습니다. 지속성은 클라이언트로 송신되거나 클라이언트에서 수신된 발행에 대해 양방향으로 작용합니다.

발행물

발행물은 토픽 문자열과 연관된 `MqttMessage` 의 인스턴스입니다. MQTT 클라이언트는 IBM MQ에 송신할 발행물을 작성하고 IBM MQ에 대한 토픽을 구독하여 발행물을 수신할 수 있습니다.

보유된 발행 및 MQTT 클라이언트

한 토픽에는 보유된 발행물이 하나만 있을 수 있습니다. 보유된 발행물이 있는 토픽에 대한 구독을 작성하는 경우 발행물이 즉시 사용자에게 전달됩니다.

구독

토픽 필터를 사용하여 관심있는 발행 토픽을 등록하려면 구독을 작성하십시오. 클라이언트는 다중 구독 또는 와일드카드를 사용하는 토픽 필터를 포함하는 구독을 작성하여 관심있는 다중 토픽을 등록할 수 있습니다. 필터와 일치하는 토픽의 발행물이 클라이언트로 송신됩니다. 구독은 클라이언트 연결이 끊긴 중에도 활성 상태로 남아 있을 수 있습니다. 발행물은 클라이언트가 다시 연결할 때 여기에 송신됩니다.

MQTT 클라이언트의 토픽 문자열 및 토픽 필터

토픽 문자열 및 토픽 필터는 발행과 구독에 사용됩니다. MQTT 클라이언트의 토픽 문자열 및 필터 구문은 IBM MQ의 토픽 문자열과 거의 동일합니다.

보유된 발행 및 MQTT 클라이언트

한 토픽에는 보유된 발행물이 하나만 있을 수 있습니다. 보유된 발행물이 있는 토픽에 대한 구독을 작성하는 경우 발행물이 즉시 사용자에게 전달됩니다.

`MqttMessage.setRetained` 메소드를 사용하여 토픽에 대한 발행물의 보유 여부를 지정하십시오.

보유된 발행물을 작성하거나 업데이트할 때 QoS 1 또는 2로 발행물을 전송하십시오. QoS 0으로 전송하는 경우 IBM MQ은(는) 비지속적 보유된 발행물을 작성합니다. 쿼리 관리자가 중지되는 경우 발행물이 보유되지 않습니다.

보유되지 않은 발행물을 토픽에 발행하는 경우 보유된 발행물은 영향을 받지 않습니다. 현재 구독자는 새로운 발행물을 수신합니다. 새 구독자는 보유된 발행물을 먼저 수신하고 새 발행물을 수신합니다.

보유된 발행물을 사용하여 마지막 측정치 값을 기록할 수 있습니다. 토픽에 대한 새 구독자는 즉시 최신 측정치를 수신합니다. 구독자가 마지막으로 발행 토픽을 구독한 이후로 새 측정치가 측정되지 않은 경우 및 구독자가 다시 구독하는 경우, 구독자가 토픽에 대해 최신으로 보유된 발행물을 다시 수신합니다.

보유된 발행물을 삭제하려면 다음 두 가지 옵션이 있습니다.

- **CLEAR TOPICSTR** MQSC 명령을 실행하십시오.
- 길이가 0인 보유된 발행물을 작성하십시오. MQTT 3.1.1 스펙에 지정된 대로, 길이가 0인 보유된 메시지가 토픽에 발행되면 해당 토픽에 대해 보유된 메시지가 지워집니다.

관련 개념

MQTT 클라이언트 애플리케이션에서 콜백 및 동기화

MQTT 클라이언트 프로그래밍 모델은 스레드를 광범위하게 사용합니다. 스레드는 서버와 주고 받는 전송 메시지의 지연에서 MQTT 클라이언트 애플리케이션을 가능한 한 많이 분리시킵니다. 출판물, 전달 토큰 및 연결 손실 이벤트는 콜백 클래스의 메서드에 전달되며, 해당 클래스는 ``를 구현합니다 `MqttCallback`.

정리 세션

MQTT 클라이언트 및 텔레메트리(MQXR) 서비스가 세션 상태 정보를 유지보수합니다. 상태 정보는 "최소 한 번" 및 "정확히 한 번" 전달과 발행물의 "정확히 한 번" 수신을 확인하는 데 사용합니다. 세션 상태에는 MQTT 클라이언트가 작성한 구독도 포함됩니다. 세션 간 상태 정보를 유지보수하거나 하지 않고 MQTT 클라이언트를 실행하도록 선택할 수 있습니다. 연결하기 전에 `MqttConnectOptions.cleanSession` 를 설정하여 세션 정리 모드를 변경하십시오.

클라이언트 ID

클라이언트 ID는 MQTT 클라이언트를 식별하는 23바이트 문자열입니다. 각 ID는 한 번에 하나의 연결 클라이언트에만 해당되어야 합니다. ID에는 쿼리 관리자 이름의 올바른 문자만 포함되어야 합니다. 이러한 제한조건 내에서 모든 ID 문자열을 사용할 수 있습니다. 클라이언트 ID를 할당하는 프로시저와 선택한 ID로 클라이언트를 구성할 수단이 있어야 합니다.

전달 토큰

이상 종료 시 메시지 발행

MQTT 클라이언트 연결이 예기치 않게 종료되면 "이상 종료 시 메시지" 발행을 송신하도록 IBM MQ Telemetry 을(를) 구성할 수 있습니다. 발행물의 콘텐츠 및 송신할 토픽을 사전정의하십시오. "이상 종료 시 메시지"는 연결 특성입니다. 클라이언트를 연결하기 전에 작성하십시오.

MQTT 클라이언트의 메시지 지속성

발행 메시지는 "최소 한 번" 또는 "정확히 한 번"의 서비스 품질(QoS)로 송신되는 경우에 지속됩니다. 클라이언트에 사용자 고유의 지속 메커니즘을 구현할 수도 있고 클라이언트가 제공하는 기본 지속 메커니즘을 사용할 수도 있습니다. 지속성은 클라이언트로 송신되거나 클라이언트에서 수신된 발행에 대해 양방향으로 작용합니다.

발행물

발행물은 토픽 문자열과 연관된 `MqttMessage` 의 인스턴스입니다. MQTT 클라이언트는 IBM MQ에 송신할 발행물을 작성하고 IBM MQ에 대한 토픽을 구독하여 발행물을 수신할 수 있습니다.

MQTT 클라이언트에서 제공하는 서비스 품질(QoS)

MQTT 클라이언트는 IBM MQ 및 MQTT 클라이언트에 발행물을 전달하는 것에 대해 세 개의 서비스 품질(QoS)을 제공합니다("최대 한 번", "최소 한 번" 및 "정확히 한 번"). MQTT 클라이언트가 IBM MQ에 요청을 송신하여 구독을 작성하는 경우 "최소 한 번" 서비스 품질(QoS)을 사용하여 요청이 송신됩니다.

구독

토픽 필터를 사용하여 관심있는 발행 토픽을 등록하려면 구독을 작성하십시오. 클라이언트는 다중 구독 또는 와일드카드를 사용하는 토픽 필터를 포함하는 구독을 작성하여 관심있는 다중 토픽을 등록할 수 있습니다. 필터와 일치하는 토픽의 발행물이 클라이언트로 송신됩니다. 구독은 클라이언트 연결이 끊긴 중에도 활성 상태로 남아 있을 수 있습니다. 발행물은 클라이언트가 다시 연결할 때 여기에 송신됩니다.

MQTT 클라이언트의 토픽 문자열 및 토픽 필터

토픽 문자열 및 토픽 필터는 발행과 구독에 사용됩니다. MQTT 클라이언트의 토픽 문자열 및 필터 구문은 IBM MQ의 토픽 문자열과 거의 동일합니다.

구독

토픽 필터를 사용하여 관심있는 발행 토픽을 등록하려면 구독을 작성하십시오. 클라이언트는 다중 구독 또는 와일드카드를 사용하는 토픽 필터를 포함하는 구독을 작성하여 관심있는 다중 토픽을 등록할 수 있습니다. 필터와 일치하는 토픽의 발행물이 클라이언트로 송신됩니다. 구독은 클라이언트 연결이 끊긴 중에도 활성 상태로 남아 있을 수 있습니다. 발행물은 클라이언트가 다시 연결할 때 여기에 송신됩니다.

하나 이상의 토픽 필터와 서비스 품질(QoS) 매개변수를 전달하는 `MqttClient.subscribe` 메소드를 사용하여 구독을 작성하십시오. 서비스 품질 매개변수는 메시지를 수신하는 데 구독자가 사용할 준비가 된 최대 서비스 품질을 설정합니다. 이 클라이언트에 송신된 메시지는 이보다 더 높은 서비스 품질로 전달될 수 없습니다. 메시지가 발행되고 구독의 레벨이 지정될 때 서비스 품질은 원래 값이나 이보다 낮은 값으로 설정됩니다. 수신되는 메시지에 대한 기본 서비스 품질은 QoS=1(최소 한 번)입니다.

구독 요청 자체는 QoS=1로 송신됩니다.

구독자는 MQTT 클라이언트가 `MqttCallback.messageArrived` 메소드를 호출할 때 발행물을 수신합니다. `messageArrived` 메소드는 발행된 메시지와 함께 토픽 문자열도 구독자에게 전달합니다.

`MqttClient.unsubscribe` 메소드를 사용하여 구독을 제거하거나 구독을 설정할 수 있습니다.

IBM MQ 명령은 구독을 제거할 수 있습니다. IBM MQ Explorer, 또는 혹은 PCF 명령어를 사용하여 `runmqsc` 구독 목록을 확인합니다. 모든 MQTT 클라이언트 구독이 이름 지정됩니다. 이것들은 다음과 같은 형식의 이름을 부여받습니다: `ClientIdentifier:Topic name`

기본값을 `MqttConnectOptions` 사용하거나 클라이언트 연결 전에 `를`

`MqttConnectOptions.cleanSession` 로 `true` 설정하면, 클라이언트가 연결될 때 해당 클라이언트의 기존 구독이 모두 삭제됩니다. 세션 중에 클라이언트가 만드는 모든 새 구독은 연결을 해제할 때 삭제됩니다.

연결하기 전에 `를` 로 `false` 설정하면 `MqttConnectOptions.cleanSession`, 클라이언트가 생성하는 모든 구독은 연결 전 해당 클라이언트에 존재하던 모든 구독에 추가됩니다. 클라이언트가 연결이 끊길 때 모든 구독은 활성인 상태로 남아 있습니다.

`cleanSession` 속성이 구독에 영향을 미치는 방식을 이해하는 다른 방법은 모달 속성으로 간주하는 것입니다. 기본 모드 `cleanSession=true`에서 클라이언트를 구독을 작성하고 세션의 범위 내에서만 발행물을 수신합니다. 대체 모드 `cleanSession=false`에서 구독은 지속됩니다. 클라이언트는 연결하거나 연결을 끊을 수 있고 해당 구독은 활성인 상태로 남아 있습니다. 클라이언트가 다시 연결하면 전달되지 않은 모든 발행물을 수신합니다. 연결된 동안 대신 활성인 구독 세트를 수정할 수 있습니다.

연결하기 전에 모드를 `cleanSession` 설정해야 하며, 이 모드는 세션이 끝날 때까지 유지됩니다. 설정을 변경하려면 클라이언트의 연결을 끊은 후 다시 연결해야 합니다. `cleanSession=true` 모드를 에서 `cleanSession=false` 로 변경하면, 해당 클라이언트의 기존 모든 구독과 아직 수신되지 않은 모든 간행물이 삭제됩니다.

활성 구독과 일치하는 발행물은 발행되는 즉시 클라이언트로 송신됩니다. 클라이언트의 연결이 끊어지면 `MqttConnectOptions.cleanSession`가 `false`로 설정된 경우 동일한 클라이언트 ID로 동일한 서버에 다시 연결될 때 클라이언트로 송신됩니다.

특정 클라이언트에 대한 구독은 클라이언트 ID로 식별됩니다. 다른 클라이언트 디바이스에서 같은 서버에 클라이언트를 다시 연결해 같은 구독을 계속하여 전달되지 않은 발행을 수신할 수 있습니다.

관련 개념

[MQTT 클라이언트 애플리케이션에서 콜백 및 동기화](#)

MQTT 클라이언트 프로그래밍 모델은 스레드를 광범위하게 사용합니다. 스레드는 서버와 주고 받는 전송 메시지의 지연에서 MQTT 클라이언트 애플리케이션을 가능한 한 많이 분리시킵니다. 출판물, 전달 토큰 및 연결 손실 이벤트는 콜백 클래스의 메서드에 전달되며, 해당 클래스는 ``를 구현합니다 `MqttCallback`.

정리 세션

MQTT 클라이언트 및 텔레메트리(MQXR) 서비스가 세션 상태 정보를 유지보수합니다. 상태 정보는 "최소 한 번" 및 "정확히 한 번" 전달과 발행물의 "정확히 한 번" 수신을 확인하는 데 사용합니다. 세션 상태에는 MQTT 클라이언트가 작성한 구독도 포함됩니다. 세션 간 상태 정보를 유지보수하거나 하지 않고 MQTT 클라이언트를 실행하도록 선택할 수 있습니다. 연결하기 전에 `MqttConnectOptions.cleanSession` 를 설정하여 세션 정리 모드를 변경하십시오.

클라이언트 ID

클라이언트 ID는 MQTT 클라이언트를 식별하는 23바이트 문자열입니다. 각 ID는 한 번에 하나의 연결 클라이언트에만 해당되어야 합니다. ID에는 쿼리 관리자 이름의 올바른 문자만 포함되어야 합니다. 이러한 제한조건 내에서 모든 ID 문자열을 사용할 수 있습니다. 클라이언트 ID를 할당하는 프로시저와 선택한 ID로 클라이언트를 구성할 수단이 있어야 합니다.

전달 토큰

이상 종료 시 메시지 발행

MQTT 클라이언트 연결이 예기치 않게 종료되면 "이상 종료 시 메시지" 발행을 송신하도록 IBM MQ Telemetry 을(를) 구성할 수 있습니다. 발행물의 콘텐츠 및 송신할 토픽을 사전정의하십시오. "이상 종료 시 메시지"는 연결 특성입니다. 클라이언트를 연결하기 전에 작성하십시오.

MQTT 클라이언트의 메시지 지속성

발행 메시지는 "최소 한 번" 또는 "정확히 한 번"의 서비스 품질(QoS)로 송신되는 경우에 지속됩니다. 클라이언트에 사용자 고유의 지속 메커니즘을 구현할 수도 있고 클라이언트가 제공하는 기본 지속 메커니즘을 사용할 수도 있습니다. 지속성은 클라이언트로 송신되거나 클라이언트에서 수신된 발행에 대해 양방향으로 작용합니다.

발행물

발행물은 토픽 문자열과 연관된 `MqttMessage`의 인스턴스입니다. MQTT 클라이언트는 IBM MQ에 송신할 발행물을 작성하고 IBM MQ에 대한 토픽을 구독하여 발행물을 수신할 수 있습니다.

MQTT 클라이언트에서 제공하는 서비스 품질(QoS)

MQTT 클라이언트는 IBM MQ 및 MQTT 클라이언트에 발행물을 전달하는 것에 대해 세 개의 서비스 품질(QoS)을 제공합니다("최대 한 번", "최소 한 번" 및 "정확히 한 번"). MQTT 클라이언트가 IBM MQ에 요청을 송신하여 구독을 작성하는 경우 "최소 한 번" 서비스 품질(QoS)을 사용하여 요청이 송신됩니다.

보유된 발행 및 MQTT 클라이언트

한 토픽에는 보유된 발행물이 하나만 있을 수 있습니다. 보유된 발행물이 있는 토픽에 대한 구독을 작성하는 경우 발행물이 즉시 사용자에게 전달됩니다.

MQTT 클라이언트의 토픽 문자열 및 토픽 필터

토픽 문자열 및 토픽 필터는 발행과 구독에 사용됩니다. MQTT 클라이언트의 토픽 문자열 및 필터 구문은 IBM MQ의 토픽 문자열과 거의 동일합니다.

MQTT 클라이언트의 토픽 문자열 및 토픽 필터

토픽 문자열 및 토픽 필터는 발행과 구독에 사용됩니다. MQTT 클라이언트의 토픽 문자열 및 필터 구문은 IBM MQ의 토픽 문자열과 거의 동일합니다.

토픽 문자열은 발행물을 구독자에게 송신하는 데 사용됩니다.

`MqttClient.getTopic(java.lang.String topicString)` 메소드를 사용하여 토픽 문자열을 작성하십시오.

토픽 필터는 토픽을 구독하고 발행물을 수신하는 데 사용됩니다. 토픽 필터에는 와일드카드가 포함될 수 있습니다. 와일드카드를 사용하여 다중 토픽을 구독할 수 있습니다. 구독 메소드를 사용하여 토픽 필터를 작성하십시오. 예: `MqttClient.subscribe(java.lang.String topicFilter)`.

토픽 문자열

IBM MQ 토픽 문자열의 구문은 토픽 문자열에 설명되어 있습니다. MQTT 토픽 문자열의 구문은 MQTT client for Java에 대한 API 문서의 `MqttClient` 클래스에 설명되어 있습니다. MQTT 클라이언트 라이브러리의 클라이언트 API 문서 링크는 [MQTT 클라이언트 프로그래밍 참조](#)를 참조하십시오.

각 유형의 토픽 문자열 구문은 거의 동일합니다. 네 가지 작은 차이점이 있습니다.

1. MQTT 클라이언트가 IBM MQ에 송신한 토픽 문자열은 큐 관리자 이름에 대한 규칙을 따라야 합니다.
2. 최대 길이는 다양합니다. IBM MQ 토픽 문자열은 10,240자로 제한됩니다. MQTT 클라이언트는 최대 65535 바이트의 토픽 문자열을 작성할 수 있습니다.
3. MQTT 클라이언트에 의해 작성된 토픽 문자열은 널 문자를 포함할 수 없습니다.
4. IBM Integration Bus에서는 널 토픽 레벨, '...//...'이 유효하지 않습니다. 널 토픽 레벨은 IBM MQ에서 지원됩니다.

IBM MQ 발행/구독과 다르게, `mqttv3` 프로토콜에는 관리 토픽 오브젝트란 개념이 없습니다. 토픽 오브젝트와 토픽 문자열에서 토픽 문자열을 구성할 수 없습니다. 그러나 토픽 문자열은 IBM MQ의 관리 토픽에 맵핑됩니다. 관리 토픽과 연관된 액세스 제어는 발행물이 토픽에 발행되는지 또는 제거되는지를 결정합니다. 구독자에게 전달될 때 발행물에 적용되는 속성은 관리 토픽의 속성으로부터 영향을 받습니다.

토픽 필터

IBM MQ 토픽 필터의 구문은 [토픽 기반 와일드카드 스킴](#)에 설명되어 있습니다. MQTT 클라이언트를 사용하여 구성할 수 있는 토픽 필터의 구문은 MQTT client for Java에 대한 API 문서의 `MqttClient` 클래스에 설명되어 있습니다. MQTT 클라이언트 라이브러리의 클라이언트 API 문서 링크는 [MQTT 클라이언트 프로그래밍 참조](#)를 참조하십시오.

관련 개념

MQTT 클라이언트 애플리케이션에서 콜백 및 동기화

MQTT 클라이언트 프로그래밍 모델은 스레드를 광범위하게 사용합니다. 스레드는 서버와 주고 받는 전송 메시지의 지연에서 MQTT 클라이언트 애플리케이션을 가능한 한 많이 분리시킵니다. 출판물, 전달 토큰 및 연결 손실 이벤트는 콜백 클래스의 메서드에 전달되며, 해당 클래스는 ``를 구현합니다 `MqttCallback`.

정리 세션

MQTT 클라이언트 및 텔레메트리(MQXR) 서비스가 세션 상태 정보를 유지보수합니다. 상태 정보는 "최소 한 번" 및 "정확히 한 번" 전달과 발행물의 "정확히 한 번" 수신을 확인하는 데 사용합니다. 세션 상태에는 MQTT 클라이언트가 작성한 구독도 포함됩니다. 세션 간 상태 정보를 유지보수하거나 하지 않고 MQTT 클라이언트를 실행하도록 선택할 수 있습니다. 연결하기 전에 `MqttConnectOptions.cleanSession` 를 설정하여 세션 정리 모드를 변경하십시오.

클라이언트 ID

클라이언트 ID는 MQTT 클라이언트를 식별하는 23바이트 문자열입니다. 각 ID는 한 번에 하나의 연결 클라이언트에만 해당되어야 합니다. ID에는 쿼리 관리자 이름의 올바른 문자만 포함되어야 합니다. 이러한 제한조건 내에서 모든 ID 문자열을 사용할 수 있습니다. 클라이언트 ID를 할당하는 프로시저와 선택한 ID로 클라이언트를 구성할 수단이 있어야 합니다.

전달 토큰

이상 종료 시 메시지 발행

MQTT 클라이언트 연결이 예기치 않게 종료되면 "이상 종료 시 메시지" 발행을 송신하도록 IBM MQ Telemetry 을(를) 구성할 수 있습니다. 발행물의 콘텐츠 및 송신할 토픽을 사전정의하십시오. "이상 종료 시 메시지"는 연결 특성입니다. 클라이언트를 연결하기 전에 작성하십시오.

MQTT 클라이언트의 메시지 지속성

발행 메시지는 "최소 한 번" 또는 "정확히 한 번"의 서비스 품질(QoS)로 송신되는 경우에 지속됩니다. 클라이언트에 사용자 고유의 지속 메커니즘을 구현할 수도 있고 클라이언트가 제공하는 기본 지속 메커니즘을 사용할 수도 있습니다. 지속성은 클라이언트로 송신되거나 클라이언트에서 수신된 발행에 대해 양방향으로 작용합니다.

발행물

발행물은 토픽 문자열과 연관된 `MqttMessage` 의 인스턴스입니다. MQTT 클라이언트는 IBM MQ에 송신할 발행물을 작성하고 IBM MQ에 대한 토픽을 구독하여 발행물을 수신할 수 있습니다.

MQTT 클라이언트에서 제공하는 서비스 품질(QoS)

MQTT 클라이언트는 IBM MQ 및 MQTT 클라이언트에 발행물을 전달하는 것에 대해 세 개의 서비스 품질(QoS)을 제공합니다("최대 한 번", "최소 한 번" 및 "정확히 한 번"). MQTT 클라이언트가 IBM MQ에 요청을 송신하여 구독을 작성하는 경우 "최소 한 번" 서비스 품질(QoS)을 사용하여 요청이 송신됩니다.

보유된 발행 및 MQTT 클라이언트

한 토픽에는 보유된 발행물이 하나만 있을 수 있습니다. 보유된 발행물이 있는 토픽에 대한 구독을 작성하는 경우 발행물이 즉시 사용자에게 전달됩니다.

구독

토픽 필터를 사용하여 관심있는 발행 토픽을 등록하려면 구독을 작성하십시오. 클라이언트는 다중 구독 또는 와일드카드를 사용하는 토픽 필터를 포함하는 구독을 작성하여 관심있는 다중 토픽을 등록할 수 있습니다. 필터와 일치하는 토픽의 발행물이 클라이언트로 송신됩니다. 구독은 클라이언트 연결이 끊긴 중에도 활성 상태로 남아 있을 수 있습니다. 발행물은 클라이언트가 다시 연결할 때 여기에 송신됩니다.

MQTT 서비스 제어 속성

멀티스레드 애플리케이션에서 더 나은 성능을 위해 MQXR 속성 MQTT 파일에서 해당

`com.ibm.mq.MQXR.Workers` 속성을 구성하여 MQXR MQTT 서비스가 사용하는 작업자 스레드 수를 설정할 수 있습니다. 이 서비스는 또한 MQXR MQTT 서비스가 쿼리 관리자에 연결하는 방식을 지정하는 속성을 지원합니다 **`MQIBindType`**.

다음 속성 파일에서 MQXR 서비스 MQTT 속성을 구성할 수 있습니다:

- ▶ **Windows** 시스템에서 `Windows:mqxr_win.properties`.
- ▶ **Linux** ▶ **AIX** 시스템에서 `AIX and Linux:mqxr_unix.properties`.

구성할 수 있는 속성은 다음과 같습니다:

표 198. MQTT 서비스 특성	
특성	설명
com.ibm.mq.MQXR.Workers	MQXR 서비스가 생성하는 서버 작업자 스레드 수. 이 값이 지정되지 않으면 시스템의 논리 프로세서 수와 동일하게 기본값으로 설정됩니다.
MQIBindType	MQXR 서비스의 바인딩 유형은 다음 중 하나입니다: FASTPATH, SHARED 또는 ISOLATED. 기본값은 FASTPATH입니다.

다음 중 MQXR MQTT 서비스에서 큐 관리자 채널에 적합한 바인딩 유형을 선택할 수 있습니다:

- FASTPATH 바인딩은 MQXR 서비스가 중간 에이전트 프로세스 없이 큐 관리자에 직접 연결할 수 있도록 합니다. 이 바인딩 방식은 더 빠르지만 격리성이 떨어집니다. MQXR 서비스가 중단될 경우, FASTPATH 바인딩이 큐 관리자에 영향을 미칠 수 있습니다. 이 경우 정상적인 작업을 재개하려면 큐 관리자를 다시 시작해야 할 수 있습니다.
- SHARED 바인딩은 여러 MQXR 서비스 인스턴스가 큐 관리자에 대한 단일 연결을 공유하도록 허용하여 성능과 리소스 사용 간의 균형을 맞춥니다.
- ISOLATED 바인딩은 개별 MQXR 서비스 인스턴스로, 큐 관리자와 자체 독립적인 연결을 가집니다. 이 바인딩 유형은 최고 수준의 격리 및 보안을 제공하지만 성능 저하를 수반합니다.

FASTPATH 바인딩 사용에 대한 자세한 내용은 [분산 큐 관리 시 고려해야 할 기타 사항을 참조하십시오](#)

관련 정보

[MQXR 서비스 정의](#)

[MQXR 서비스 시작 및 중지](#)

[setmqxr \(MQTT 서버 속성 설정\)](#)

개발하는 Microsoft Windows Communication Foundation 애플리케이션 IBM MQ

WCF Windows Communication Foundation Microsoft (Windows Communication Foundation) 사용자 지정 채널은 WCF 클라이언트와 서비스 IBM MQ 간에 메시지를 송수신합니다.

관련 개념

[1247 페이지의 『.NET 를 사용하는 WCF용 IBM MQ 사용자 정의 채널 소개』](#)

IBM MQ의 사용자 정의 채널은 Microsoft WCF(Windows Communication Foundation) 통합 프로그래밍 모델을 사용하는 전송 채널입니다.

[1252 페이지의 『WCF용 IBM MQ 사용자 정의 채널 사용』](#)

커뮤니케이션 Windows 파운데이션(WCF)용 사용자 지정 IBM MQ 채널을 사용하는 프로그래머를 위한 정보 개요

[1269 페이지의 『WCF 샘플 사용』](#)

WCF(Windows Communication Foundation) 샘플은 IBM MQ 사용자 정의 채널을 사용할 수 있는 방법을 보여주는 간단한 예를 제공합니다.

[FFST: WCF \(XMS\) 첫 번째 장애 지원 기술](#)

관련 태스크

[IBM MQ 에 대한 WCF 사용자 지정 채널 추적](#)

[IBM MQ 문제에 대한 WCF 사용자 지정 채널 문제 해결](#)

.NET 를 사용하는 WCF용 IBM MQ 사용자 정의 채널 소개

IBM MQ의 사용자 정의 채널은 Microsoft WCF(Windows Communication Foundation) 통합 프로그래밍 모델을 사용하는 전송 채널입니다.

Microsoft.NET 3에 도입된 Microsoft Windows Communication Foundation 프레임워크를 통해 .NET 애플리케이션 및 서비스를 연결하는 데 사용하는 전송 및 프로토콜과 별도로 이 애플리케이션 및 서비스를 개발할 수 있으므로, 서비스 또는 애플리케이션이 배치되는 환경에 따라 대체 전송 또는 구성을 사용할 수 있습니다.

필수 결합을 포함하는 채널 스택을 빌드하여 런타임 시 WCF가 연결을 관리합니다.

- 프로토콜 요소: WS-* 표준과 같은 프로토콜을 지원하기 위해 하나 이상을 추가하거나 아무 것도 추가할 수 없는 선택적 요소 세트입니다.
- 메시지 인코더: 메시지를 Wire 형식으로 직렬화하는 작업을 제어하는 스택의 필수 요소입니다.
- 전송 채널: 직렬화된 메시지를 해당 엔드포인트로 전송할 책임이 있는 스택의 필수 요소입니다.

IBM MQ의 사용자 정의 채널은 전송 채널이므로 애플리케이션에서 필요한 대로 WCF 사용자 정의 바인딩을 사용하여 메시지 인코더 및 선택적 프로토콜과 쌍을 이루어야 합니다. 이 방식으로, WCF를 사용하도록 개발된 애플리케이션은 IBM MQ의 사용자 정의 채널을 사용하여 Microsoft이(가) 제공하는 기본 제공 전송을 사용하는 것과 동일한 방식으로 데이터를 보내고 받을 수 있으므로, IBM MQ의 확장 가능하고 신뢰할 수 있는 비동기식 메시징 기능과 간단하게 통합할 수 있습니다. 지원되는 함수의 전체 목록은 [1252 페이지의 『WCF 사용자 정의 채널 특성 및 기능』](#)의 내용을 참조하십시오.

WCF의 IBM MQ 사용자 정의 채널을 사용하는 시점 및 이유

사용자 지정 IBM MQ 채널을 사용하면 WCF 클라이언트와 서비스 간에 메시지를 주고받을 수 있으며, 이는 MicrosoftWCF가 제공하는 내장 전송 방식과 동일한 방식으로 작동합니다. 이를 통해 애플리케이션은 WCF 통합 프로그래밍 모델 내에서 해당 IBM MQ 기능에 접근할 수 있습니다.

WCF의 IBM MQ 사용자 정의 채널에 대한 일반 사용 패턴 시나리오는 기본 IBM MQ 메시지의 전송을 위한 비 SOAP 인터페이스입니다.

비-SOAP/비-JMS 메시지(순수 MQMessage) 형식을 사용하여 전달된 메시지

WCF의 사용자 지정 IBM MQ 채널을 네이티브 IBM MQ 메시지 전송을 위한 비 SOAP 인터페이스로 사용할 경우, 메시지는 비 SOAP/비JMS 메시지(순수 MQMessage) 형식으로 전달됩니다 IBM MQ.

WCF 사용자는 서비스를 시작할 수 있습니다. 또는 다시 말하면 서비스 사용자는 MQMessages를 사용하여 메시지를 IBM MQ 큐에 송신할 수 있습니다. 애플리케이션은 MQMD 필드 및 페이로드를 가져오고 설정할 수 있습니다. 메시지가 큐에 IBM MQ 존재할 때, 이 메시지는 WCF 서비스 또는 C#, WindowsLinux C++, Java C 또는 z/OS. AIXNET 프레임워크에서 실행되는 비-WCF 애플리케이션과 같은 비-WCF 애플리케이션에 의해 처리될 수 있습니다.

WCF용 IBM MQ 사용자 정의 채널의 소프트웨어 요구사항

이 주제에서는 WCF용 IBM MQ 사용자 정의 채널의 소프트웨어 요구사항을 간략하게 설명합니다. WCF용 IBM MQ 사용자 정의 채널은 IBM WebSphere MQ 7.0 이상 큐 관리자에만 연결할 수 있습니다.

런타임 환경 요구사항

- Microsoft.NET 호스트 머신에 프레임워크 v4.7.2 이상이 설치되어 있어야 합니다.
- *Java and .NET Messaging and Web Services*는 기본적으로 IBM MQ 설치 프로그램의 일부로 설치됩니다. 이 컴포넌트는 사용자 정의 채널에 필요한 .NET 어셈블리를 글로벌 어셈블리 캐시에 설치합니다.

참고: V4.7.2 이상 Microsoft.NET Framework 버전이 설치 전에 설치되지 IBM MQ 애플리케이션은, 제품 IBM MQ 설치 오류 없이 진행되지만 해당 기능은 IBM MQ classes for .NET 사용할 수 없습니다. IBM MQ을(를) 설치한 후에 .NET Framework이(가) 설치된 경우에는 `WMQInstallDir\bin\amqiRegisterDotNet.cmd` 스크립트를 실행하여 IBM MQ.NET 어셈블리를 등록해야 합니다. 여기서, `WMQInstallDir`은 IBM MQ이(가) 설치된 디렉토리입니다. 이 스크립트는 GAC(Global Assembly Cache)에 필수 어셈블리를 설치합니다. 수행된 조치를 기록하는 `amqi*.log` 파일 세트가 %TEMP% 디렉토리에 작성됩니다. 이전 버전(예: .NETV3.5)에서 V4.7.2 이상으로 업그레이드하는 .NET 경우 `amqiRegisterDotNet.cmd` 스크립트를 다시 실행할 필요가 없습니다.

개발 환경 요구사항

- Microsoft Visual Studio 2015 또는 .NET4.7.2Windows 용 소프트웨어 개발 키트(SDK) 이상.

- Microsoft.NET 샘플 솔루션 파일을 빌드하려면 호스트 머신에 프레임워크 V4.7.2 이상이 설치되어 있어야 합니다.

WCF의 IBM MQ 사용자 정의 채널: 설치된 사항

IBM MQ의 사용자 정의 채널은 Microsoft WCF(Windows Communication Foundation) 통합 프로그래밍 모델을 사용하는 전송 채널입니다. 기본적으로 사용자 정의 채널은 설치의 일부로 설치됩니다.

WCF용 IBM MQ 사용자 정의 채널

사용자 정의 채널 및 종속 항목은 기본적으로 설치되는 Java and .NET Messaging and Web Services 컴포넌트에 포함되어 있습니다. IBM MQ 8.0 이전 버전에서 IBM MQ을(를) 업그레이드할 때 Java and .NET Messaging and Web Services 컴포넌트가 이전 설치에 이미 설치된 경우에는 업데이트가 기본적으로 WCF용 IBM MQ 사용자 정의 채널을 설치합니다.

.NET Messaging and Web Services 컴포넌트는 IBM.XMS.WCF.dll 파일 및 IBM.WMQ.WCF.dll 파일을 포함하며 이러한 파일은 WCF 인터페이스 클래스를 포함한 기본 사용자 정의 채널 어셈블리입니다. 이러한 파일은 GAC(Global Assembly Cache)에 설치되며 다음 디렉토리에서도 사용 가능합니다.
MQ_INSTALLATION_PATH\bin. 여기서, MQ_INSTALLATION_PATH은(는) IBM MQ이(가) 설치된 디렉토리입니다.

다음 표에는 사용자 정의 채널을 사용하는 데 필요한 주요 클래스가 요약되어 있습니다.

표 199. 사용자 정의 채널을 사용하는 데 필요한 주요 클래스		
	SOAP/JMS 인터페이스(기존)	비SOAP/비JMS 인터페이스(IBM MQ 8.0)
Custom Channel Assembly	IBM.XMS.WCF.dll	IBM.WMQ.WCF.dll
Transport Binding Name	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement	IBM.WMQ.WCF.WmqIbmTransportBindingElement
Transport Binding Importer	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter	IBM.WMQ.WCF.WmqIbmTransportBindingElementImporter
Transport Binding Config	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig	IBM.WMQ.WCF.WmqIbmTransportBindingElementConfig
Samples(Oneway)	SimpleOneWay_Client, SimpleOneWay_Service	MQMessaging_OneWay_Client, MQMessaging_OneWay_Service
Samples(RequestReply)	SimpleRequestReply_Client, SimpleRequestReply_Service	MQMessaging_RequestReply_Client, MQMessaging_RequestReply_Service

IBM.WMQ.WCF.dll은 SOAP/JMS와 비SOAP/비JMS 인터페이스를 모두 지원합니다. IBM.WMQ.WCF 어셈블리에서는 두 인터페이스를 모두 지원하므로 개발된 새 애플리케이션에서 이 어셈블리를 사용하는 것이 좋습니다.

MQSTR 형식화된 메시지 송신

요청 메시지의 유형이 MQSTR인 경우 MQSTR 형식으로 응답 메시지를 송신하도록 선택할 수 있습니다.

추가 URI 매개변수 **replyMessageFormat**을 사용하여 응답 메시지의 형식을 변경해야 합니다. 지원되는 값은 다음과 같습니다.

"""

" "이 기본값입니다.

응답 메시지는 바이트(MQMFT_NONE) 형식입니다. 예를 들면, 다음과 같습니다.

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.NoJndi&replyDestination=SampleReplyQ&replyMessageFormat= "
```

MQSTR

응답 메시지는 MQSTR(MQMFT_STRING) 형식입니다. 예를 들면, 다음과 같습니다.

```
"jms:/queue?  
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)  
&initialContextFactory=com.ibm.mq.jms.NoJndi&replyDestination=SampleReplyQ&replyMessageForma  
t=MQSTR"
```

참고:

1. **replyMessageFormat** 값에서는 대소문자가 구분되지 않습니다.
2. " " 또는 MQSTR 이외의 값을 사용하면 올바르지 않은 매개변수 값 예외가 발생합니다.

IBM MQ 사용자 정의 채널 샘플

이 샘플에서는 WCF의 IBM MQ 사용자 정의 채널을 사용하는 방법을 보여주는 간단한 예를 제공합니다. 샘플 및 연관된 파일은 `MQ_INSTALLATION_PATH\tools\dotnet\samples\cs\wcf` 디렉토리에 있습니다. 여기서 `MQ_INSTALLATION_PATH`은(는) IBM MQ의 설치 디렉토리입니다. IBM MQ 사용자 정의 채널 샘플에 대한 자세한 정보는 [1269 페이지의 『WCF 샘플 사용』](#)의 내용을 참조하십시오.

svcutil.exe.config

`svcutil.exe.config`은(는) Microsoft WCF `svcutil` 클라이언트 프록시 생성 도구가 사용자 정의 채널을 인식하도록 하는 데 필요한 구성 설정의 예입니다. `svcutil.exe.config` 파일은 `MQ_INSTALLATION_PATH\tools\wcf\docs\examples\` 디렉토리에 있습니다. 여기서 `MQ_INSTALLATION_PATH`은(는) IBM MQ의 설치 디렉토리입니다. `svcutil.exe.config` 사용에 대한 자세한 정보는 [1266 페이지의 『실행 중인 서비스에서 메타데이터와 svcutil 도구를 사용하여 WCF 클라이언트 프록시 및 애플리케이션 구성 파일 생성』](#)의 내용을 참조하십시오.

WCF 아키텍처

WCF용 IBM MQ 사용자 정의 채널이 IBM Message Service Client for .NET (XMS .NET) API에 통합됩니다.

SOAP/JMS 인터페이스

WCF 아키텍처는 다음 다이어그램에 표시된 바와 같습니다.

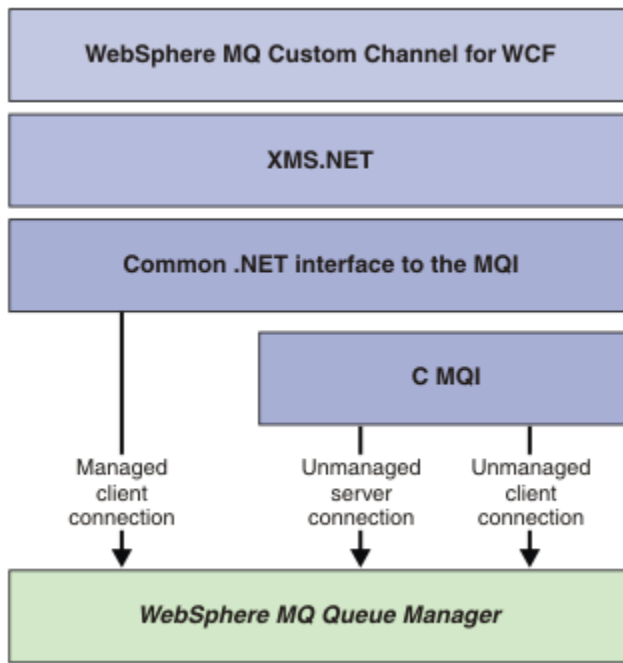


그림 166. SOAP/JMS 인터페이스용 WCF 아키텍처

제품 설치와 함께 기본적으로 모든 필수 컴포넌트가 설치됩니다.

세 가지 연결은 다음과 같습니다.

- 관리 클라이언트 연결
- 비관리 서버 연결
- 비관리 클라이언트 연결

이러한 연결에 대한 자세한 정보는 [1257 페이지의 『WCF 연결 옵션』](#)의 내용을 참조하십시오.

비SOAP/비JMS 인터페이스

WCF용 IBM MQ 사용자 정의 채널은 SOAP/JMS 인터페이스(IBM WebSphere MQ 7.0.1에서 사용 가능) 및 비 SOAP/비JMS 인터페이스를 모두 지원합니다.

WCF 아키텍처는 다음 다이어그램에 표시된 바와 같습니다.

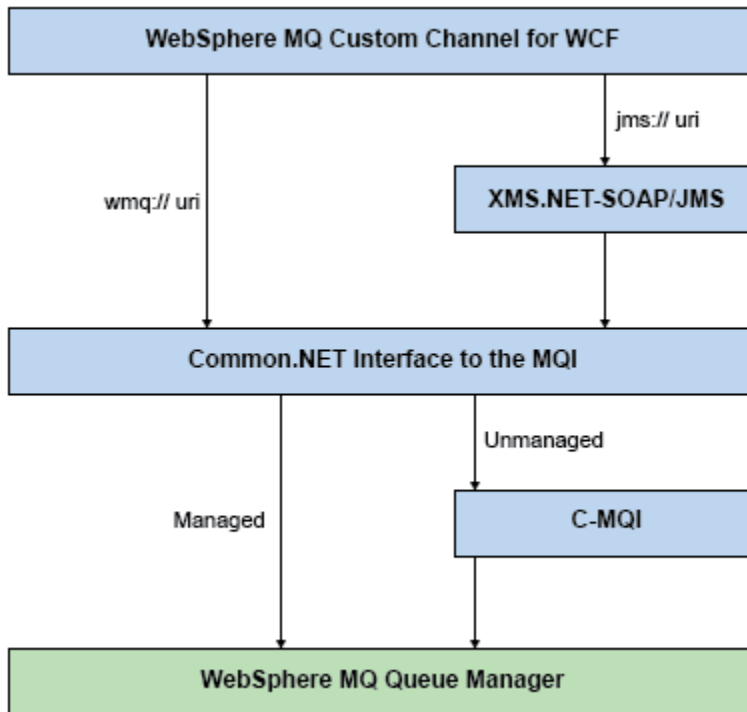


그림 167. 비SOAP/비JMS 인터페이스의 WCF 아키텍처

WCF용 IBM MQ 사용자 정의 채널 사용

커뮤니케이션 Windows 파운데이션(WCF)용 사용자 지정 IBM MQ 채널을 사용하는 프로그래머를 위한 정보 개요

커뮤니케이션 Microsoft Windows 파운데이션은 프레임워크 Microsoft.NET 3의 웹 서비스 및 메시징 지원을 뒷받침합니다. IBM MQ .NET .NETMicrosoft Framework 3에서 제공하는 내장 채널과 동일한 방식으로 WCF 내에서 사용자 지정 채널로 사용할 수 있습니다.

사용자 정의 채널을 통해 전송되는 메시지는 SOAP over JMS 구현에 따라 IBM MQ형식화됩니다. 그런 다음 애플리케이션은 WCF 또는 WebSphere SOAP over JMS 서비스 인프라를 통해 호스팅되는 서비스와 통신할 수 있습니다.

WCF 사용자 정의 채널 특성 및 기능

WCF 사용자 정의 채널 특성 및 기능에 관한 정보는 다음 주제를 사용하십시오.

WCF 사용자 정의 채널 웨이프

'IBM MQ' Microsoft 'Windows' WCF(Communication Foundation) 사용자 지정 채널 내에서 사용할 수 있는 사용자 지정 채널 모양에 대한 개요입니다.

WCF용 IBM MQ 사용자 정의 채널은 다음 두 채널 웨이프를 지원합니다.

- 단방향
- 요청-응답

WCF는 호스트되는 서비스 계약에 따라 자동으로 채널 웨이프를 선택합니다.

IsOneWay 매개변수만 사용하는 메소드를 포함하는 계약은 단방향 채널 웨이프로 서비스합니다. 예를 들어 다음과 같습니다.

```
[OperationContract(IsOneWay = true)]
void printString(String text);
```


단방향 메소드와 요청-응답 메소드의 혼합 또는 모든 요청-응답 메소드를 포함하는 계약은 요청-응답 채널 셰이프를 통해 서비스합니다. 예를 들면, 다음과 같습니다.

```
[OperationContract]
int subtract(int a, int b);

[OperationContract(IsOneWay = true)]
void printString(string text);
```

참고: 동일한 계약에서 단방향 및 요청-응답 메소드를 혼합하는 경우, 단방향 메소드는 서비스에서 널 응답을 받을 때까지 대기하므로 혼합 환경에서 작업할 때 특히 예상대로 동작하는지 확인해야 합니다.

단방향 채널

예를 들어 WCF용 IBM MQ 단방향 사용자 정의 채널은 단방향 채널 셰이프를 사용하여 WCF 클라이언트에서 메시지를 보내는 데 사용합니다. 채널은 한 방향으로만 메시지를 송신할 수 있습니다(예: 클라이언트 큐 관리자에서 WCF 서비스의 큐로 송신).

요청-응답 채널

예를 들어 WCF용 IBM MQ 요청-응답 사용자 정의 채널은 비동기식으로 두 방향으로 메시지를 송신하는 데 사용합니다. 비동기 메시징에는 동일한 클라이언트 인스턴스를 사용해야 합니다. 채널은 한 방향으로 메시지를 송신할 수 있습니다(예: 클라이언트 큐 관리자에서 WCF 서비스의 큐로). 그런 다음 WCF에서 클라이언트 큐 관리자의 큐로 응답 메시지를 송신할 수 있습니다.

WCF URI 매개변수 이름과 값

SOAP/JMS 인터페이스 및 비SOAP/비JMS 인터페이스의 URI 매개변수 이름 및 값입니다.

SOAP/JMS 인터페이스

connectionFactory

connectionFactory 매개변수는 필수입니다.

initialContextFactory

initialContextFactory 매개변수는 필수이며 WebSphere Application Server 및 다른 제품과의 호환성을 위해 "com.ibm.mq.jms.Nojndi"로 설정해야 합니다.

비SOAP/비JMS 인터페이스

URI 형식은 MA93 스펙용입니다. IBM MQ IRI 스펙의 자세한 내용은 SupportPac - MA93을 참조하십시오.

IBM MQ URI 구분

```
wmq-iri = "wmq:" [ "/" connection-name ] "/" wmq-dest ["?" parm *("&" parm)]
connection-name = tcp-connection-name / other-connection-name
tcp-connection-name = ihost [ ":" port ]
other-connection-name = 1*(iunreserved / pct-encoded)
wmq-dest = queue-dest / topic-dest
queue-dest = "msg/queue/" wmq-queue ["@" wmq-qmgr]
wmq-queue = wmq-name
wmq-qmgr = wmq-name
wmq-name = 1*48( wmq-char )
topic-dest = "msg/topic/" wmq-topic
wmq-topic = segment *( "/" segment )
```

IBM MQ IRI 예

다음 예제 URI는 서비스 요청자에게 포트 1414에서 example.com이라는 시스템에 대한 IBM MQ TCP 클라이언트 바인딩 연결을 사용할 수 있으며, 큐 관리자 QM1에서 SampleQ라는 큐에 지속 요청 메시지를 넣을 수 있음을 알립니다. IRI는 서비스 제공자가 SampleReply라는 큐에 응답을 넣도록 지정합니다.

```
1)wmq://example.com:1414/msg/queue/SampleQ@QM1?
ReplyTo=SampleReplyQ&persistence=MQPER_NOT_PERSISTENT
2)wmq://localhost:1414/msg/queue/Q1?
connectQueueManager=QM1&replyTo=Q2&connectionmode=managed
```

TLS 사용 연결

WCF 클라이언트/서비스를 사용하여 보안(TLS) 연결을 설정하려면 URI의 적절한 값으로 다음 특성을 설정하십시오. 보안 연결을 설정할 때 접두부로 "*"가 붙는 모든 특성은 필수입니다.

- **sslKeyRepository**: *SYSTEM 또는 *USER
- ***sslCipherSpec**: 올바른 CipherSpec(예: TLS_RSA_WITH_AES_128_CBC_SHA256).
- **sslCertRevocationCheck**: true 또는 false.
- **sslKeyResetCount**: 32kb보다 큰 값.
- **sslPeerName**: 서버 인증서의 식별 이름

예를 들면, 다음과 같습니다.

```
"wmq://localhost:1414/msg/queue/SampleQ?  
connectQueueManager=QM1&sslkeyrepository=*SYSTEM&sslcipherSpec=  
TLS_RSA_WITH_AES_128_CBC_SHA&sslcertrevocationcheck=true&sslpe  
ername=" + " + "CN=ibmwebspheremqmm&sslkeyresetcount=45000"
```

WCF 사용자 정의 채널 보장 전달

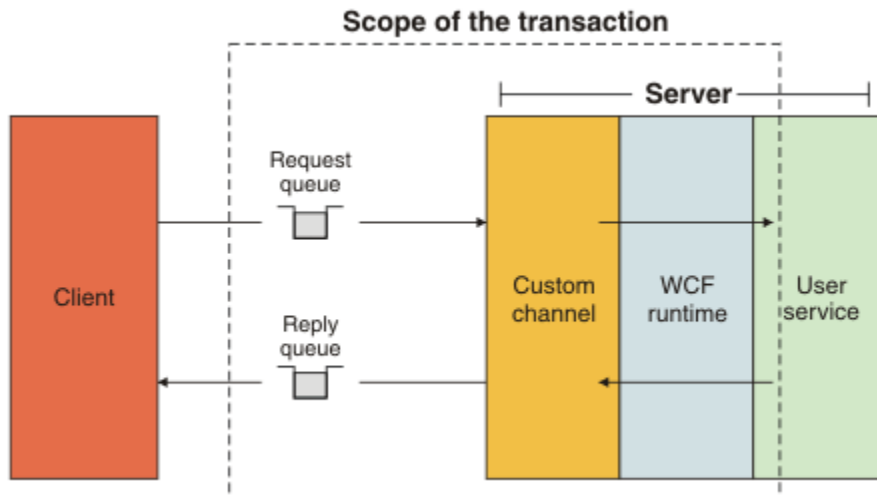
보장 전달은 서비스 요청 또는 응답이 처리되고 유실되지 않도록 보장합니다.

로컬 트랜잭션 동기점에서 요청 메시지를 수신하고 응답 메시지를 송신하므로, 런타임 실패가 발생하는 경우 이 동기점을 롤백할 수 있습니다. 이러한 실패의 예로는 서비스에서 핸들링하지 않는 예외 처리, 서비스에 메시지를 디스패치하는 데 실패 또는 응답 메시지 전달 실패가 있습니다.

AssuredDelivery는 런타임 실패가 발생하는 경우 서비스에서 수신한 모든 요청 메시지와 서비스에서 송신한 응답 메시지가 유실되지 않도록 서비스 계약에 지정할 수 있는 보장 전달 속성입니다.

시스템 실패 또는 정전이 발생하는 경우에도 메시지가 보존되려면 메시지를 지속 메시지로 송신해야 합니다. 지속 메시지를 사용하려면 클라이언트 애플리케이션에서 해당 엔드포인트 URI에 이 옵션을 지정해야 합니다.

분산 트랜잭션은 지원되지 않으며, 트랜잭션 범위는 IBM MQ에서 수행한 요청과 응답 메시지 이상으로 확장되지 않습니다. 실패가 발생한 결과 서비스에서 수행한 작업을 다시 실행하여, 메시지를 다시 수신하게 될 수 있습니다. 다음 다이어그램은 트랜잭션의 범위를 보여줍니다.



다음 예에 표시된 대로 서비스 클래스에 AssuredDelivery 속성을 적용하여 보장 전달을 사용합니다.

```
[AssuredDelivery]  
class TestCalculatorService : IWMQSampleCalculatorContract  
{  
    public int add(int a, int b)  
    {  
        int ans = a + b;  
        return ans;  
    }  
}
```

AssuredDelivery 속성을 사용할 때 다음 사항을 알고 있어야 합니다.

- 메시지를 롤백하고 다시 수신하면 실패가 재발할 가능성이 크다고 채널이 판별하면 메시지가 변조 메시지로 처리되어 재처리를 위해 요청 큐에 리턴되지 않습니다. 예를 들어 수신된 메시지가 올바르게 형식화되지 않았거나 서비스에 디스패치할 수 없는 경우입니다. 서비스 오퍼레이션에서 발생한 핸들링되지 않은 예외는 요청 큐의 백아웃 임계값 특성에 지정된 최대 횟수만큼 메시지를 다시 전달할 때까지 항상 재송신됩니다. 자세한 정보는 다음을 참조하십시오. [1255 페이지의 『WCF 사용자 정의 채널 변조 메시지』](#)
- 채널은 트랜잭션 무결성을 시행하기 위해 단일 실행 스레드를 사용하는 자동 조작으로 각 요청 메시지의 읽기, 처리 및 응답을 수행합니다. 서비스 오퍼레이션을 동시에 실행할 수 있도록 채널에서 WCF를 사용하여 채널의 다중 인스턴스를 작성할 수 있습니다. 요청 처리에 사용할 수 있는 채널 인스턴스 수는 바인딩 속성 `MaxConcurrentCalls`를 통해 제어합니다. 자세한 정보는 다음을 참조하십시오. [1262 페이지의 『WCF 바인딩 구성 옵션』](#)
- 보장 전달 기능에서는 `IOperationInvoker`와 `ILoggerHandler` WCF 확장성 지점을 모두 사용합니다. 이러한 확장성 지점을 애플리케이션에서 외부적으로 사용하는 경우 애플리케이션에서 이전에 등록된 모든 확장성 지점이 호출되는지 확인해야 합니다. `ILoggerHandler`에 대해 이 작업을 수행하지 못하면 오류가 보고되지 않게 됩니다. `IOperationInvoker`에 대해 이 작업을 수행하지 못하면 WCF가 응답을 중지할 수 있습니다.

WCF 사용자 정의 채널 보안

WCF용 IBM MQ 사용자 정의 채널에서는 큐 관리자에 대한 비관리 클라이언트 연결용으로만 TLS를 사용하도록 지원합니다.

클라이언트 채널 정의 테이블(CCDT)에서 항목을 사용하여 TLS를 지정합니다. CCDT에 관한 자세한 정보는 [클라이언트 채널 정의 테이블의 내용](#)을 참조하십시오.

WCF 클라이언트 채널 정의 테이블(CCDT)

WCF용 IBM MQ 사용자 정의 채널은 클라이언트 채널 정의 테이블(CCDT)을 사용하여 클라이언트 연결을 위한 연결 정보를 구성하도록 지원합니다.

CCDT는 다음 두 환경 변수를 통해 제어합니다.

- `MQCHLLIB`는 테이블이 있는 디렉토리를 지정합니다.
- `MQCHLTAB`는 테이블의 파일 이름을 지정합니다.

이러한 환경 변수가 정의되면 URI에 지정된 클라이언트 연결 세부 정보보다 우선합니다.

클라이언트 채널 정의 테이블에 대한 자세한 정보는 [클라이언트 채널 정의 테이블](#)을 참조하십시오.

WCF 사용자 정의 채널 변조 메시지

서비스가 요청 메시지를 처리하지 못하거나 응답 큐에 응답 메시지를 전달하지 못하면 메시지가 변조 메시지로 처리됩니다.

변조 요청 메시지

요청 메시지를 처리할 수 없으면 변조 메시지로 처리됩니다. 이 조치를 수행하면 서비스가 처리할 수 없는 동일한 메시지를 다시 받지 않게 됩니다. 처리할 수 없는 요청 메시지를 변조 메시지로 처리하려면 다음 상황 중 하나가 True여야 합니다.

- 메시지 백아웃 수가 요청 큐에 지정된 백아웃 임계값을 초과했습니다. 이 상황은 서비스의 보장 전달이 지정된 경우에만 발생합니다. 보장 전달에 대한 자세한 정보는 [1254 페이지의 『WCF 사용자 정의 채널 보장 전달』](#)의 내용을 참조하십시오.
- 메시지가 올바르게 형식화되지 않았으며 SOAP over JMS 메시지로 해석할 수 없습니다.

변조 응답 메시지

서비스가 응답 큐에 응답 메시지를 전달하지 못하면 응답 메시지가 변조 메시지로 처리됩니다. 응답 메시지의 경우 이 조치를 사용하면 문제점 판별을 지원하기 위해 나중에 응답 메시지를 검색할 수 있습니다.

변조 메시지 핸들링

변조 메시지에 수행된 조치는 큐 관리자 구성과 메시지의 보고서 옵션에 설정된 값에 따라 다릅니다. SOAP over JMS의 경우 기본적으로 다음 보고서 옵션이 요청 메시지에 설정되며 이 옵션은 구성할 수 없습니다.

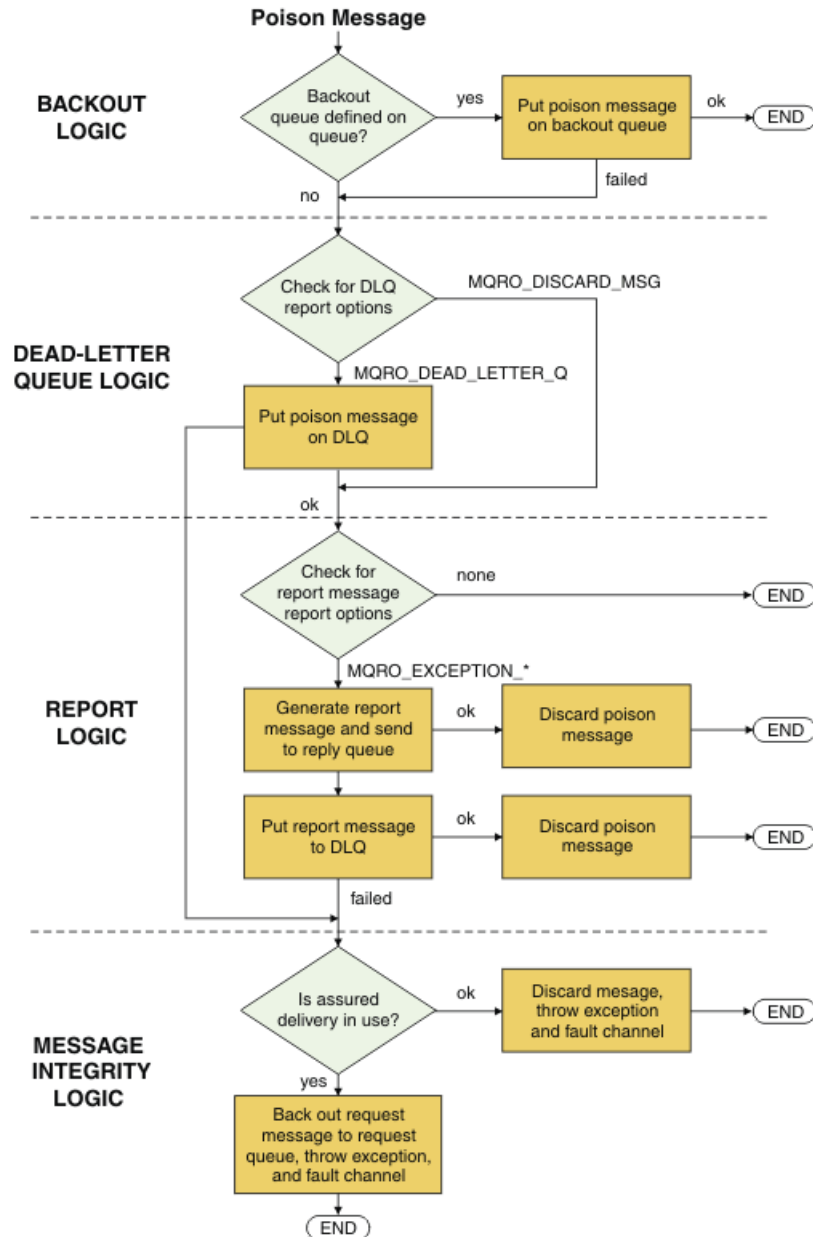
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

SOAP over JMS의 경우 응답 메시지에 기본적으로 다음 보고서 옵션이 설정되며 이 옵션은 구성할 수 없습니다.

- MQRO_DEAD_LETTER_Q

메시지가 비WCF 소스에서 입력되는 경우 해당 소스의 문서를 참조하십시오.

다음 다이어그램은 변조 메시지 핸들링에 실패하는 경우 수행할 수 있는 조치와 단계를 보여줍니다.



WCF 애플리케이션의 IBM MQ 메시지 기능

WCF 애플리케이션의 비-SOAP/비-JMS(즉, IBM MQ) 메시지 기능입니다.

비-SOAP/비-JMS 인터페이스의 경우, WCF 애플리케이션의 IBM MQ 메시지 기능은 다음과 같습니다.

- WCF 애플리케이션은 IBM MQ 애플리케이션이 처리할 수 있는 기본 IBM MQ 메시지를 송신하고 수신할 수 있습니다.
- WCF 애플리케이션에는 MQMD 및 페이로드 업데이트에 대한 전체 제어 권한이 있습니다.

- WCF 클라이언트는 모든 'IBM MQ 클라이언트(예: C, 'Java', 'JMS', '.NET' 클라이언트)에서 사용할 수 있는 'IBM MQ' 메시지를 보낼 수 있습니다.

비-SOAP/비-JMS 인터페이스용 WCF는 메시지의 메시지 페이로드 및 MQMD를 설정하기 위해 다음 클래스를 사용해야 합니다.

- 문자열 유형의 페이로드의 WmqStringMessage
- 바이트 유형의 페이로드의 WmqBytesMessage
- XML 유형의 페이로드의 WmqXmlMessage

메시지의 페이로드를 설정하려면 페이로드 유형에 따라 WmqStringMessage, WmqBytesMessage 또는 WmqXmlMessage 클래스의 '**Data**' 속성을 사용합니다. 예를 들어, 다음 코드를 사용하여 문자열 유형의 페이로드를 설정하십시오.

```
WmqStringMessage strMsg = new WmqStringMessage();
//Setting the Message PayLoad
strMsg.Data = "Hello World";
//MQMD property
strMsg.Format = WmqMessageFormat.MQFMT_STRING;
```

WCF 연결 옵션

WCF용 IBM MQ 사용자 정의 채널을 큐 관리자에 연결하는 세 가지 모드가 있습니다. 요구사항에 가장 적합한 연결 유형을 고려하십시오.

연결 옵션에 대한 자세한 정보는 542 페이지의 [『연결 차이점』](#)의 내용을 참조하십시오.

WCF 아키텍처에 대한 자세한 정보는 1250 페이지의 [『WCF 아키텍처』](#)의 내용을 참조하십시오.

비관리 클라이언트 연결

이 모드에서 설정된 연결은 IBM MQ 클라이언트로 로컬 시스템이나 원격 시스템에서 실행 중인 IBM MQ 서버에 연결합니다.

WCF용 IBM MQ 사용자 정의 채널을 IBM MQ 클라이언트로 사용하기 위해 IBM MQ 서버 또는 별도의 시스템에 IBM MQ MQI client와 함께 설치할 수 있습니다.

비관리 서버 연결

서버 바인딩 모드에서 사용할 때 WCF용 IBM MQ 사용자 정의 채널은 네트워크를 통해 통신하지 않고 큐 관리자 API를 사용합니다. 바인딩 연결을 사용하면 네트워크 연결을 사용하는 것보다 IBM MQ 애플리케이션의 성능이 향상됩니다.

바인딩 연결을 사용하려면 IBM MQ 서버에서 WCF용 IBM MQ 사용자 정의를 설치해야 합니다.

관리 클라이언트 연결

이 모드에서 설정된 연결은 IBM MQ 클라이언트로 로컬 시스템이나 원격 시스템에서 실행 중인 IBM MQ 서버에 연결합니다.

이 모드에서 연결하는 .NET 3에 대한 IBM MQ 사용자 정의 채널 클래스는 .NET 관리 코드에 남아 있으며 기본 서비스를 호출하지 않습니다. 관리 코드에 대한 자세한 정보는 Microsoft 문서를 참조하십시오.

관리 클라이언트를 사용하는 데 관한 많은 제한사항이 있습니다. 이러한 제한사항에 대한 자세한 정보는 542 페이지의 [『관리 클라이언트 연결』](#)의 내용을 참조하십시오.

WCF의 IBM MQ 사용자 정의 채널 작성 및 구성

WCF의 IBM MQ 사용자 정의 채널은 Microsoft에서 제공한 전송 WCF 채널과 동일한 방식으로 작동합니다. WCF용 IBM MQ 사용자 정의 채널은 두 방법 중 하나로 작성할 수 있습니다.

이 태스크 정보

애플리케이션에서 사용할 수 있는 완전한 채널 스택을 작성할 수 있도록 IBM MQ 사용자 정의 채널은 WCF 전송 채널로 WCF와 통합되므로, 메시지 인코더 및 선택적 프로토콜 채널과 쌍으로 연결되어야 합니다. 전체 채널 스택을 성공적으로 작성하려면 두 요소가 필요합니다.

1. 바인딩 정의: 일반 구성 설정 외에 전송 채널, 메시지 인코더 및 프로토콜을 비롯하여 애플리케이션 채널 스택을 빌드하는 데 필요한 요소를 지정합니다. 사용자 정의 채널에서 바인딩 정의는 WCF 사용자 정의 바인딩의 양식으로 작성해야 합니다.
2. 엔드포인트 정의: 서비스 계약을 바인딩 정의와 링크하고 애플리케이션이 연결할 수 있는 위치를 설명하는 실제 연결 URI도 제공합니다. 사용자 정의 채널의 경우 URI는 SOAP JMS URI의 양식을 사용합니다.

이러한 정의는 다음 두 방법 중 하나로 작성할 수 있습니다.

- 관리 방식: 애플리케이션 구성 파일(예: `app.config`)에 세부사항을 제공하여 정의를 작성합니다.
- 프로그래밍 방식: 애플리케이션 코드에서 정의를 작성합니다.

정의를 작성하는 데 사용할 메소드는 다음과 같이 애플리케이션의 요구사항을 기반으로 결정합니다.

- 구성의 관리 방식 메소드는 애플리케이션을 다시 빌드하지 않고 서비스와 클라이언트 배치 후 세부 정보를 대체하는 유연성을 제공합니다.
- 구성의 프로그래밍 방식 메소드를 사용하면 구성 오류에서 보호하는 기능이 향상되며 런타임 시 동적으로 구성을 생성할 수 있습니다.

애플리케이션 구성 파일에 바인딩 및 엔드포인트 정보를 제공하여 관리 시 WCF 사용자 정의 채널 작성

WCF의 IBM MQ 사용자 정의 채널은 전송 레벨 WCF 채널입니다. 사용자 정의 채널을 사용하려면 엔드포인트와 바인딩을 정의해야 하며, 애플리케이션 구성 파일에 바인딩과 엔드포인트 정보를 제공하여 이러한 정의를 수행할 수 있습니다.

전송 레벨 WCF 채널인 WCF의 IBM MQ 사용자 정의 채널을 구성하여 사용하려면 바인딩과 엔드포인트 정의를 정의해야 합니다. 바인딩이 채널의 구성 정보를 보유하고 엔드포인트 정의가 연결 세부 정보를 보유합니다. 이러한 정의는 다음 두 방법으로 작성할 수 있습니다.

- 여기에 설명된 대로 프로그래밍 방식으로 애플리케이션 코드에서 직접 작성합니다. [1260 페이지의 『바인딩 및 엔드포인트 정보를 프로그래밍 방식으로 제공하여 WCF 사용자 정의 채널 작성』](#)
- 다음 프로시저에 설명된 대로 관리 시 애플리케이션 구성 파일에 자세한 내용을 제공하여 작성합니다.

클라이언트 또는 서비스 애플리케이션 구성 파일의 이름은 일반적으로 `yourappname.exe.config`입니다. 여기서, `yourappnam`은 애플리케이션의 이름입니다. 다음 방식으로 Microsoft 서비스 구성 편집기 도구 `SvcConfigEditor.exe`을(를) 사용하여 애플리케이션 구성 파일을 가장 쉽게 수정할 수 있습니다.

- `SvcConfigEditor.exe` 구성 편집기 도구를 시작하십시오. 도구의 기본 설치 위치는 `Drive:\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcConfigEditor.exe`입니다. 여기서, `Drive`는 설치 드라이브의 이름입니다.

1단계: WCF를 사용하여 사용자 정의 채널을 찾을 수 있도록 바인딩 요소 확장자 추가

1. **고급 > 확장자 > 바인딩 요소**를 마우스의 오른쪽 단추로 클릭하여 메뉴를 열고 **새로 작성**을 클릭하십시오.
2. 이 표에 표시된 대로 필드를 완료하십시오.

표 200. 새 바인딩 요소 필드	
필드	값
이름	IBM.XMS.WCF.SoapJmsIbmTransportChannel
Type	GAC(Global Assembly Cache)에서 IBM.XMS.WCF.dll로 이동하고 IBM.XMS.WCFSoapJmsIbmTransportBindingElementConfig를 선택하십시오.

2단계: WCF 메시지 인코더와 사용자 정의 채널을 쌍으로 결합하는 사용자 정의 바인딩 정의 작성

1. 바인딩을 마우스의 오른쪽 단추로 클릭하여 메뉴를 열고 새 바인딩 구성을 선택하십시오.
2. 이 표에 표시된 대로 필드를 완료하십시오.

표 201. 새 바인딩 구성 필드	
필드	값
이름	CustomBinding_WMQ
BindingElement 1	textMessageEncoding (MessageVersion: Soap11)
BindingElement 2	IBM.XMS.WCF.S SoapJmsIbmTransportChannel

3단계: 바인딩 특성 지정

1. 1259 페이지의 『2단계: WCF 메시지 인코더와 사용자 정의 채널을 쌍으로 결합하는 사용자 정의 바인딩 정의 작성』에서 작성한 바인딩에서 *IBM.XMS.WCF.S SoapJmsIbmTransportChannel* 전송 바인딩을 선택하십시오.
2. 1262 페이지의 『WCF 바인딩 구성 옵션』에 설명된 대로 특성의 기본값을 필요한 대로 변경하십시오.

4단계: 엔드포인트 정의 작성

1259 페이지의 『2단계: WCF 메시지 인코더와 사용자 정의 채널을 쌍으로 결합하는 사용자 정의 바인딩 정의 작성』에서 작성한 사용자 정의 바인딩을 참조하는 엔드포인트 정의를 작성하고 서비스의 연결 세부 정보를 제공하십시오. 이 정보를 지정하는 방식은 정의 대상(클라이언트 애플리케이션 또는 서비스 애플리케이션)에 따라 달라집니다.

클라이언트 애플리케이션에서 다음과 같이 엔드포인트 정의를 클라이언트 섹션에 추가하십시오.

1. **클라이언트 > 엔드포인트**를 마우스의 오른쪽 단추로 클릭하여 메뉴를 열고 **새 클라이언트 엔드포인트**를 선택하십시오.
2. 이 표에 표시된 대로 필드를 완료하십시오.

표 202. 새 클라이언트 엔드포인트 필드	
필드	값
이름	Endpoint_WMQ
주소	서비스에 액세스하는 데 필요한 <i>WMQ</i> 연결 세부사항을 설명하는 <i>SOAP/JMS URI</i> 입니다. 자세한 내용은 1261 페이지의 『WCF 엔드포인트 URI 주소 형식용 IBM MQ 사용자 정의 채널』의 내용을 참조하십시오.
바인딩	customBinding
BindingConfiguration	CustomBinding_WMQ
계약	서비스 계약 인터페이스의 이름

서비스 애플리케이션에서는 다음과 같이 서비스 정의를 서비스 섹션에 추가하십시오.

1. **서비스**를 마우스의 오른쪽 단추로 클릭하여 메뉴를 열고 **새 서비스**를 선택한 다음 호스트할 서비스 클래스를 선택하십시오.
2. 새 서비스의 **엔드포인트** 섹션에 엔드포인트 정의를 추가하고 이 표에 표시된 대로 필드를 완료하십시오.

표 203. 새 서비스 엔드 포인트 필드	
필드	값
이름	Endpoint_WMQ
주소	서비스에 액세스하는 데 필요한 WMQ 연결 세부사항을 설명하는 SOAP/JMS URI입니다. 자세한 내용은 1261 페이지의 『WCF 엔드포인트 URI 주소 형식용 IBM MQ 사용자 정의 채널』의 내용을 참조하십시오.
바인딩	customBinding
BindingConfiguration	CustomBinding_WMQ
계약	서비스 구현 클래스 이름

바인딩 및 엔드포인트 정보를 프로그래밍 방식으로 제공하여 WCF 사용자 정의 채널 작성

WCF의 IBM MQ 사용자 정의 채널은 전송 레벨 WCF 채널입니다. 사용자 정의 채널을 사용하려면 엔드포인트 및 바인딩을 정의해야 하며, 이러한 정의는 애플리케이션 코드에서 직접 프로그래밍 방식으로 수행할 수 있습니다.

전송 레벨 WCF 채널인 WCF의 IBM MQ 사용자 정의 채널을 구성하여 사용하려면 바인딩과 엔드포인트 정의를 정의해야 합니다. 바인딩이 채널의 구성 정보를 보유하고 엔드포인트 정의가 연결 세부 정보를 보유합니다. 자세한 정보는 1269 페이지의 『WCF 샘플 사용』의 내용을 참조하십시오.

이러한 정의는 다음 두 방법으로 작성할 수 있습니다.

- 1258 페이지의 『애플리케이션 구성 파일에 바인딩 및 엔드포인트 정보를 제공하여 관리 시 WCF 사용자 정의 채널 작성』에 설명된 대로 관리 시 애플리케이션 구성 파일에 자세한 내용을 제공하여 작성합니다.
- 다음 하위 주제에 설명된 대로 애플리케이션 코드에서 직접 프로그래밍 방식으로 작성합니다.

프로그래밍 방식으로 바인딩 및 엔드포인트 정보 정의: SOAP/JMS 인터페이스

SOAP/JMS 인터페이스의 경우, 애플리케이션 코드에서 직접 프로그래밍 방식으로 엔드포인트 및 바인딩을 정의할 수 있습니다.

이 태스크 정보

바인딩 및 엔드포인트 정보를 프로그래밍 방식으로 제공하려면 다음 단계를 완료하여 애플리케이션에 필수 코드를 추가하십시오.

프로시저

1. 애플리케이션에 다음 코드를 추가하여 채널의 전송 바인딩 요소의 인스턴스를 작성하십시오.

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
```

2. 필수 바인딩 특성을 설정하십시오(예를 들어, 다음 코드를 애플리케이션에 추가하여 ClientConnectionMode를 설정하여).

```
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
```

3. 애플리케이션에 다음 코드를 추가하여 전송 채널을 메시지 인코더와 쌍을 이루게 하는 사용자 정의 바인딩을 작성하십시오.

```
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
```

4. SOAP/JMS URI를 작성하십시오.

서비스에 액세스하는 데 필요한 'IBM MQ' 연결 세부 정보를 설명하는 SOAP/JMS URI를 엔드포인트 주소로 제공해야 합니다. 사용자가 지정하는 주소는 채널이 서비스 애플리케이션 또는 클라이언트 애플리케이션에 대해 사용 중인지 여부에 달려 있습니다.

- 클라이언트 애플리케이션의 경우, SOAP/JMS URI는 다음과 같이 EndpointAddress로서 작성되어야 합니다.

```
EndpointAddress address = new EndpointAddress("jms:/queue?  
destination=SampleQ@QM1&connectionFactory=  
=connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

- 서비스 애플리케이션의 경우, SOAP/JMS URI는 다음과 같이 URI로서 작성되어야 합니다.

```
Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=  
connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

엔드포인트 주소에 대한 자세한 정보는 [1261 페이지의 『WCF 엔드포인트 URI 주소 형식용 IBM MQ 사용자 정의 채널』](#)의 내용을 참조하십시오.

프로그래밍 방식으로 바인딩 및 엔드포인트 정보 정의: 비-SOAP/비-JMS 인터페이스
비-SOAP/비-JMS 인터페이스의 경우, 애플리케이션 코드에서 직접 프로그래밍 방식으로 엔드포인트 및 바인딩을 정의할 수 있습니다.

이 태스크 정보

바인딩 및 엔드포인트 정보를 프로그래밍 방식으로 제공하려면 다음 단계를 완료하여 애플리케이션에 필수 코드를 추가하십시오.

프로시저

1. 다음 코드를 애플리케이션에 추가하여 WmqBinding을 작성하십시오.

```
WmqBinding binding = new WmqBinding();
```

이 코드는 비-SOAP/비-JMS 인터페이스에 필요한 WmqMsgEncodingElement 및 WmqIbmTransportBindingElement가 쌍을 이루게 하는 바인딩을 작성합니다.

2. 서비스를 액세스하는 데 필요한 IBM MQ 연결 세부사항을 설명하는 wmq:// URI를 제공합니다.

wmq:// URI를 제공하는 방법은 채널이 서비스 애플리케이션 또는 클라이언트 애플리케이션에 사용되는지 여부에 따라 다릅니다.

- 클라이언트 애플리케이션의 경우, wmq:// URI는 다음과 같이 EndpointAddress로서 작성되어야 합니다.

```
EndpointAddress address = new EndpointAddress  
("wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
```

- 서비스 애플리케이션의 경우, wmq:// URI는 다음과 같이 URI로서 작성되어야 합니다.

```
Uri sampleAddress = new Uri(  
"wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
```

WCF 엔드포인트 URI 주소 형식용 IBM MQ 사용자 정의 채널

웹 서비스는 위치와 연결 세부 정보를 제공하는 URI(Universal Resource Identifier)를 사용하여 지정됩니다. URI 형식은 SOAP/JMS 인터페이스 또는 비SOAP/비JMS 인터페이스를 사용하는지에 따라 달라집니다.

SOAP/JMS 인터페이스

SOAP용 IBM MQ 전송에서 지원되는 URI 형식을 사용하면 대상 서비스에 액세스할 때 SOAP/ IBM MQ 특정 매개변수와 옵션을 통해 포괄적인 수준으로 제어할 수 있습니다. 이 형식은 WebSphere Application Server 및 CICS와(와) 호환되므로 두 제품 모두와 IBM MQ의 통합을 용이하게 합니다.

URI 구문은 다음과 같습니다.

```
jms:/queue? name=value&name=value...
```

여기서 name은 매개변수 이름이고 value는 해당 값이며, name = value 요소는 두 번째 및 후속 발생에서 앰퍼샌드(&)를 앞에 붙여서 몇 번이고 반복될 수 있습니다.

매개변수 이름은 IBM MQ 오브젝트의 이름이므로 대소문자를 구분합니다. 매개변수가 두 번 이상 지정되면 마지막 매개변수가 적용됩니다. 즉, 클라이언트 애플리케이션이 URI에 추가하여 매개변수 값을 대체할 수 있습니다. 인식되지 않는 추가 매개변수가 포함되는 경우 이 매개변수는 무시됩니다.

XML 문자열에 URI를 저장하는 경우 앰퍼샌드 문자를 "&"로 표시해야 합니다. 마찬가지로, URI가 스크립트에 코딩된 경우에는 &와 같은 문자를 이스케이프 처리하십시오. 그렇지 않으면 셀을 통해 해석됩니다.

다음은 Axis 서비스를 위한 간단한 URI의 예입니다.

```
jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.NoJndi
```

다음은 .NET 서비스용 간단한 URI의 예입니다.

```
jms:/queue?destination=myQ&connectionFactory=()&targetService=MyService.asmx
&initialContextFactory=com.ibm.mq.jms.NoJndi
```

필수 매개변수만 제공되고(targetService는 .NET 서비스에만 필요) connectionFactory에는 옵션이 제공되지 않습니다.

이 Axis 예에서 connectionFactory에는 여러 옵션이 포함됩니다.

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&initialContextFactory=com.ibm.mq.jms.NoJndi
```

이 Axis 예에서 connectionFactory의 sslPeerName 옵션도 지정되었습니다. sslPeerName 자체의 값에는 동일한 값 쌍과 유효 공백이 포함됩니다.

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
sslPeerName(CN=MQ Test 1,0=IBM,S=Hampshire,C=GB)
&initialContextFactory=com.ibm.mq.jms.NoJndi
```

비SOAP/비JMS 인터페이스

비SOAP/비JMS 인터페이스의 URI 형식을 사용하면 대상 서비스에 액세스할 때 IBM MQ 특정 매개변수와 옵션을 통해 포괄적인 수준으로 제어할 수 있습니다.

URI 구문은 다음과 같습니다.

```
wmq://example.com:1415/msg/queue/INS.QUOTE.REQUEST@MOTOR.INS ?ReplyTo=msg/queue/
INS.QUOTE.REPLY@BRANCH452&persistence=MQPER_NOT_PERSISTENT
```

이 IRI는 IBM MQ TCP 클라이언트 바인딩 연결을 사용하여 포트 1415에서 example.com이라는 기계에 연결할 수 있으며, MOTOR.INS라는 큐 관리자에서 INS.QUOTE.REQUEST라는 큐에 지속 요청 메시지를 넣을 수 있음을 서비스 요청자에게 알립니다. IRI는 서비스 제공자가 BRANCH452 큐 관리자에서 INS.QUOTE.REPLY라는 큐에 응답을 넣도록 지정합니다. URI 형식은 SupportPac MA93에 대해 지원된 형식입니다. IBM MQ IRI 스펙에 대한 자세한 내용은 [SupportPac MA93: IBM MQ - 서비스 정의](#)를 참조하십시오.

WCF 바인딩 구성 옵션

구성 옵션을 사용자 정의 채널 바인딩 정보에 적용하는 방법은 두 가지가 있습니다. 특성은 관리 시 설정하거나 프로그래밍 방식으로 설정합니다.

바인딩 구성 옵션은 다음 두 방법 중 하나로 설정할 수 있습니다.

1. 관리 방식: 애플리케이션 구성 파일(예: app.config)에 있는 사용자 정의 바인딩 정의의 전송 섹션에 바인딩 특성 설정을 지정해야 합니다.

2. 프로그래밍 방식: 사용자 정의 바인딩을 초기화하는 중에 특성을 지정하려면 애플리케이션 코드를 수정해야 합니다.

관리 시 바인딩 특성 설정

바인딩 특성 설정은 애플리케이션 구성 파일(예: app.config)에 지정할 수 있습니다. 구성 파일은 다음 예에 표시된 대로 **svcutil**에서 생성합니다.

SOAP/JMS 인터페이스

```
<customBinding>
...
  <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
    maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>
...
</customBinding>
```

비SOAP/비JMS 인터페이스

```
<customBinding>
  <IBM.WMQ.WCF.WmqMsgEncodingElement/>
  <IBM.WMQ.WCF.WmqIbmTransportChannel maxBufferPoolSize="524288"
    maxMessageSize="65536" clientConnectionMode="managedclient"/>
</customBinding>
```

프로그래밍 방식으로 바인딩 특성 설정

클라이언트 연결 모드를 지정하기 위해 WCF 바인딩 특성을 추가하려면 사용자 정의 바인딩 초기화 중에 특성을 지정하도록 서비스 코드를 수정해야 합니다.

다음 예를 사용하여 비관리 클라이언트 연결 모드를 지정하십시오.

```
SoapJmsIbmTransportBindingElement
transportBindingElement = new SoapJmsIbmTransportBindingElement();
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.CLIENT_UNMANAGED;

Binding sampleBinding = new CustomBinding(new TextMessageEncodingBindingElement(),
                                           transportBindingElement);
```

WCF 바인딩 특성

표 204. 관리 시 또는 프로그래밍 방식으로 설정할 때 바인딩 특성 값				
특성 이름	클라이언트 또는 서비스 애플리케이션	관리 값	프로그래밍 값	설명
maxBufferPoolSize	둘 다	0 - 64비트 사인된 정수	0 - 64비트 사인된 정수	채널 인스턴스의 WCF 메시지 버퍼를 저장하는 데 사용할 수 있는 메모리의 최대 크기를 지정합니다.
maxMessageSize	둘 다	1 - 32비트 사인된 정수	1 - 32비트 사인된 정수	개별 WCF 메시지에 사용할 수 있는 최대 메모리를 지정합니다.

표 204. 관리 시 또는 프로그래밍 방식으로 설정할 때 바인딩 특성 값 (계속)

특성 이름	클라이언트 또는 서비스 애플리케이션	관리 값	프로그래밍 값	설명
clientConnectionMode	둘 다	0(기본값) 1	AS_URI(기본값) CLIENT_UNMANAGED	전송 채널의 클라이언트 연결 모드를 지정합니다. 0은 클라이언트 연결 모드가 URI에서 지정된 대로임을 나타냅니다. 클라이언트 연결을 사용하는 경우에만 사용합니다. 클라이언트 연결 모드가 URI에 지정된 대로임을 지정합니다. 클라이언트 연결 모드가 설정되지 않은 경우 0이 기본값입니다. 1은 클라이언트 연결 모드가 관리되지 않은 클라이언트임을 나타냅니다. 클라이언트 연결을 사용하는 경우에만 사용합니다.
MaxConcurrentCalls	클라이언트	범위는 0 - 2 147 483 647입니다. 16이 기본값입니다.	범위는 0 - 2 147 483 647입니다. 16이 기본값입니다.	이 특성은 동시에 개별 클라이언트 프록시에서 발생할 수 있는 동시 조작의 최대수를 정의합니다. 더 많은 조작이 시작되면 진행 중인 조작이 완료되거나 제한시간이 초과될 때까지 큐에 넣습니다. 이 설정은 개별 프록시에서 이용할 수 있는 최대 스레드와 자원을 제어하는 데 사용할 수 있습니다. 0은 이 한계를 제거하므로 모든 조작을 동시에 시도할 수 있습니다.
MaxConcurrentCalls	서비스	범위는 1 - 2 147 483 647입니다. 16이 기본값입니다.	범위는 1 - 2 147 483 647입니다. 16이 기본값입니다.	이 특성은 보장된 전달 기능을 사용하는 경우에만 사용됩니다(보장된 전달에 대한 자세한 정보는 1254 페이지의 『WCF 사용자 정의 채널 보장 전달』 참조). 지정된 엔드포인트에 대해 동시에 진행할 수 있는 동시 조작의 최대수를 지정합니다. 이 설정을 변경할 때 주의해야 합니다. 각 동시 조작에는 추가 자원이 필요합니다. 특히 요청을 처리할 스레드 풀의 연관된 스레드와 사용자 정의 채널의 새 인스턴스가 필요합니다. 과도하게 할당하면 생산성이 저하되며 성능에 심각한 영향을 줄 수 있습니다. 이 특성을 지원하려면 스레드 풀을 적절하게 구성해야 합니다.

WCF용 서비스 빌드 및 호스트

WCF 서비스를 작성하고 구성하는 방법을 설명하는 Microsoft WCF(Windows Communication Foundation) 서비스 개요입니다.

WCF용 IBM MQ 사용자 정의 채널 및 이 채널을 사용하는 WCF 서비스는 다음 메소드로 호스팅할 수 있습니다.

- 자체 호스팅
- Windows 서비스

WCF용 IBM MQ 사용자 정의 채널은 Windows Process Activation Service에서 호스팅할 수 없습니다.

다음 주제는 관련 단계를 시연하기 위해 몇 가지 간단한 자체 호스팅 예를 제공합니다. 자세한 정보와 최신 세부 정보를 포함하는 Microsoft WCF 온라인 문서는 Microsoft MSDN 웹 사이트(<https://msdn.microsoft.com>)에 있습니다.

메소드 1을 사용하여 WCF 서비스 애플리케이션 빌드: 애플리케이션 구성 파일을 사용하여 관리 방식으로 자체 호스팅

애플리케이션 구성 파일을 작성했으므로, 서비스의 인스턴스를 열고 지정된 코드를 애플리케이션에 추가하십시오.

시작하기 전에

1258 페이지의 『애플리케이션 구성 파일에 바인딩 및 엔드포인트 정보를 제공하여 관리 시 WCF 사용자 정의 채널 작성』에 설명된 대로 서비스의 애플리케이션 구성 파일을 작성하거나 편집하십시오.

이 태스크 정보

1. 서비스 호스트에서 서비스의 인스턴스를 인스턴스화하고 여십시오. 서비스 유형은 서비스 구성 파일에 지정된 서비스 유형과 같아야 합니다.
2. 다음 코드를 애플리케이션에 추가하십시오.

```
ServiceHost service = new ServiceHost(typeof(MyService));
service.Open();
...
service.Close();
```

메소드 2를 사용하여 WCF 서비스 애플리케이션 빌드: 애플리케이션에서 직접 프로그래밍 방식으로 자체 호스팅

바인딩 특성을 추가하고 필수 서비스 클래스의 인스턴스로 서비스 호스트를 작성하며 서비스를 엽니다.

시작하기 전에

1. 사용자 정의 채널 IBM.XMS.WCF.dll 파일에 대한 참조를 프로젝트에 추가하십시오. IBM.XMS.WCF.dll 은(는) *WMQInstallDir\bin*에 있습니다. 여기서 *WMQInstallDir*은 IBM MQ이(가) 설치된 디렉토리입니다.
2. *using* 문을 IBM.XMS.WCF 네임스페이스에 추가하십시오(예: *using IBM.XMS.WCF*).
3. 1260 페이지의 『바인딩 및 엔드포인트 정보를 프로그래밍 방식으로 제공하여 WCF 사용자 정의 채널 작성』에 설명된 대로 채널 바인딩 요소와 엔드포인트의 인스턴스를 작성하십시오.

이 태스크 정보

채널의 바인딩 특성을 변경해야 하는 경우 다음 단계를 완료하십시오.

1. 다음 예에 표시된 대로 *transportBindingElement*에 바인딩 특성을 추가하십시오.

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
Uri address = new Uri("jms:/queue?destination=SampleQQ@QM1&connectionFactory=
connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. 필수 서비스 클래스의 인스턴스로 서비스 호스트를 작성합니다.

```
ServiceHost service = new ServiceHost(typeof(MyService));
```

3. 서비스를 엽니다.

```
service.AddServiceEndpoint(typeof(IMyServiceContract), binding, address);
service.Open();
...
service.Close();
```

HTTP 엔드포인트를 사용하여 메타데이터 공개

WCF용 IBM MQ 사용자 정의 채널을 사용하도록 구성된 서비스의 메타데이터를 공개하기 위한 지시사항입니다.

이 태스크 정보

서비스 메타데이터를 노출해야 하는 경우(예를 들어, svcutil과(와) 같은 도구가 오프라인 WSDL 파일 대신 실행 중인 서비스에서 직접 액세스할 수 있도록), HTTP 엔드포인트로 서비스 메타데이터를 노출시켜서 이를 수행해야 합니다. 이 엔드포인트를 추가하는 데 다음 단계를 사용할 수 있습니다.

1. 메타데이터를 ServiceHost에 공개해야 하는 기본 주소를 추가합니다. 예를 들어 다음과 같습니다.

```
ServiceHost service = new ServiceHost(typeof(TestService),
    new Uri("http://localhost:8000/MyService"));
```

2. 다음과 같이 서비스를 열기 전에 ServiceHost에 다음 코드를 추가하십시오.

```
ServiceMetadataBehavior metadataBehavior = new ServiceMetadataBehavior();
metadataBehavior.HttpGetEnabled = true;
service.Description.Behaviors.Add(metadataBehavior);
service.AddServiceEndpoint(typeof(IMetadataExchange),
    MetadataExchangeBindings.CreateMexHttpBinding(), "mex");
```

결과

이제 메타데이터를 다음 주소에서 사용할 수 있습니다. <http://localhost:8000/MyService>

WCF의 클라이언트 애플리케이션 빌드

Microsoft WCF(Windows Communication Foundation) 클라이언트 애플리케이션의 생성 및 빌드 개요

WCF 서비스의 클라이언트 애플리케이션을 작성할 수 있습니다. 애플리케이션에서 직접 사용할 수 있는 필수 구성과 프로시 파일을 작성하기 위해 일반적으로 Microsoft ServiceModel Metadata Utility Tool(Svcutil.exe)을 사용하여 클라이언트 애플리케이션을 생성합니다.

실행 중인 서비스에서 메타데이터와 svcutil 도구를 사용하여 WCF 클라이언트 프록시 및 애플리케이션 구성 파일 생성

Microsoft svcutil.exe 도구를 사용하여 WCF용 IBM MQ 사용자 정의 채널을 사용하도록 구성된 서비스의 클라이언트를 생성하기 위한 지시사항

시작하기 전에

다음은 svcutil 도구를 사용하여 애플리케이션에서 직접 사용할 수 있는 필수 구성 및 프로시 파일을 작성하기 위한 세 가지 필수조건입니다.

- svcutil 도구를 시작하려면 WCF 서비스가 실행 중이어야 합니다.
- 실행 중인 서비스에서 직접 클라이언트를 생성하려면 WCF 서비스에서 IBM MQ 사용자 정의 채널 엔드포인트 참조 외에도 HTTP 포트를 사용하여 메타데이터를 공개해야 합니다.
- 사용자 정의 채널은 svcutil의 구성 데이터에 등록되어야 합니다.

이 태스크 정보

다음 단계는 IBM MQ 사용자 정의 채널을 사용하도록 구성되었지만 런타임 시 개별 HTTP 포트를 통해 메타데이터도 공개하는 서비스의 클라이언트를 생성하는 방법을 설명합니다.

1. WCF 서비스를 시작하십시오(svcutil 도구를 시작하려면 서비스가 실행 중이어야 함).

- 설치 루트의 svcutil.exe 구성 파일에 있는 세부사항을 활성 svcutil 구성 파일(일반적으로 C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config)에 추가하십시오. 그러면 svcutil이 IBM MQ 사용자 정의 채널을 인식합니다.
- 명령 프롬프트에서 svcutil을 실행하십시오. 예를 들어 다음과 같습니다.

```
svcutil /language:C# /r: installlocation\bin\IBM.XMS.WCF.dll
/config:app.config http://localhost:8000/IBM.XMS.WCF/samples
```

- 생성된 app.config 및 YourService.cs 파일을 Microsoft Visual Studio 클라이언트 프로젝트에 복사하십시오.

다음에 수행할 작업

서비스 메타데이터를 직접 검색할 수 없으면 svcutil을 사용하여 wsdl에서 대신 클라이언트 파일을 생성할 수 있습니다. 자세한 정보는 1267 페이지의 [『WSDL과 svcutil을 사용하여 WCF 클라이언트 프록시와 애플리케이션 구성 파일 생성』](#)의 내용을 참조하십시오.

WSDL과 svcutil을 사용하여 WCF 클라이언트 프록시와 애플리케이션 구성 파일 생성

서비스의 메타데이터를 사용할 수 없는 경우 WSDL에서 WCF 클라이언트를 생성하는 지시사항입니다.

실행 중인 서비스의 메타데이터에서 클라이언트를 생성하기 위해 서비스의 메타데이터를 직접 검색할 수 없는 경우 svcutil을 사용하여 WSDL에서 대신 클라이언트 파일을 생성할 수 있습니다. IBM MQ 사용자 정의 채널을 사용하도록 지정하려면 WSDL을 다음과 같이 수정해야 합니다.

- 네임스페이스 정의와 정책 정보를 추가하십시오.

```
<wsdl:definitions
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
  utility-1.0.xsd">

  <wsp:Policy wsu:Id="CustomBinding_IWMQSampleContract_policy">
    <wsp:ExactlyOne>
      <wsp:All>
        <xms:xms xmlns:xms="http://sample.schemas.ibm.com/policy/xms" />
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>

  ...

</wsdl:definitions>
```

- 새 정책 섹션을 참조하고 기본 바인딩 요소에서 transport 정의를 제거하도록 바인딩 섹션을 수정하십시오.

```
<wsdl:definitions ...>

  <wsdl:binding ...>
    <wsp:PolicyReference URI="#CustomerBinding_IWMQSampleContract_policy" />
    <[soap]:binding ... transport="" />
    ...
  </wsdl:binding>
</wsdl:definitions>
```

- 명령 프롬프트에서 svcutil을 실행하십시오. 예를 들어 다음과 같습니다.

```
svcutil /language:C# /r: MQ_INSTALLATION_PATH\bin\IBM.XMS.WCF.dll
/config:app.config MQ_INSTALLATION_PATH\src\samples\WMQAxis\default\service
\soap.server.stockQuoteAxis_Wmq.wsdl
```

애플리케이션 구성 파일이 있는 클라이언트 프록시를 사용하여 WCF 클라이언트 애플리케이션 빌드

시작하기 전에

1258 페이지의 [『애플리케이션 구성 파일에 바인딩 및 엔드포인트 정보를 제공하여 관리 시 WCF 사용자 정의 채널 작성』](#)에 설명된 대로 클라이언트의 애플리케이션 구성 파일을 작성하거나 편집하십시오.

이 태스크 정보

클라이언트 프록시의 인스턴스를 인스턴스화하고 여십시오. 생성된 프록시에 전달된 매개변수는 클라이언트 구성 파일에 지정된 엔드포인트 이름(예: Endpoint_WMQ)과 같아야 합니다.

```
MyClientProxy myClient = new MyClientProxy("Endpoint_WMQ");
try {
    myClient.myMethod("HelloWorld!");
    myClient.Close();
}
catch (TimeoutException e) {
    Console.Out.WriteLine(e);
    myClient.Abort();
}
catch (CommunicationException e) {
    Console.Out.WriteLine(e);
    myClient.Abort();
}
catch (Exception e) {
    Console.Out.WriteLine(e);
    myClient.Abort();
}
```

프로그래밍 방식 구성이 있는 클라이언트 프록시를 사용하여 WCF 클라이언트 애플리케이션 빌드

시작하기 전에

1. 사용자 정의 채널 IBM.XMS.WCF.dll 파일에 대한 참조를 프로젝트에 추가하십시오. IBM.XMS.WCF.dll 은(는) *WMQInstallDir\bin*에 있습니다. 여기서 *WMQInstallDir*은 IBM MQ이(가) 설치된 디렉토리입니다.
2. *using* 문을 IBM.XMS.WCF 네임스페이스에 추가하십시오(예: *using IBM.XMS.WCF*).
3. 1260 페이지의 『바인딩 및 엔드포인트 정보를 프로그래밍 방식으로 제공하여 WCF 사용자 정의 채널 작성』에 설명된 대로 채널의 엔드포인트와 바인딩 요소의 인스턴스를 작성하십시오.

이 태스크 정보

채널의 바인딩 특성을 변경해야 하는 경우 다음 단계를 완료하십시오.

1. 다음 그림에 표시된 대로 *transportBindingElement*에 바인딩 특성을 추가하십시오.

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
EndpointAddress address =
    new EndpointAddress("jms:/queue?destination=SampleQ@QM1&connectionFactory=
connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.NoJndi");
```

2. 다음 그림에 표시된 대로 클라이언트 프록시를 작성하십시오. 여기서 *binding*과 *endpoint address*는 1단계에서 구성되어 전달된 바인딩과 엔드포인트 주소입니다.

```
MyClientProxy myClient = new MyClientProxy(binding, endpoint address);
try {
    myClient.myMethod("HelloWorld!");
    myClient.Close();
}
catch (TimeoutException e) {
    Console.Out.WriteLine(e);
    myClient.Abort();
}
catch (CommunicationException e) {
    Console.Out.WriteLine(e);
    myClient.Abort();
}
catch (Exception e) {
    Console.Out.WriteLine(e);
    myClient.Abort();
}
```

WCF 샘플 사용

WCF(Windows Communication Foundation) 샘플은 IBM MQ 사용자 정의 채널을 사용할 수 있는 방법을 보여주는 간단한 예를 제공합니다.

샘플 프로젝트를 빌드하려면 Microsoft.NET 3.5 SDK 또는 Microsoft Visual Studio 2008이 필요합니다.

단순 단방향 및 서버 WCF 샘플

이 샘플에서는 단방향 채널 셰이프를 사용하여 WCF 클라이언트에서 WCF (Windows Communication Foundation) 서비스를 시작하는 데 사용되는 IBM MQ 사용자 정의 채널을 보여줍니다.

이 태스크 정보

이 서비스는 문자열을 콘솔에 출력하는 단일 메소드를 구현합니다. [1266 페이지의 『실행 중인 서비스에서 메타데이터와 svcutil 도구를 사용하여 WCF 클라이언트 프록시 및 애플리케이션 구성 파일 생성』](#)에 설명된 별도로 노출된 HTTP 엔드포인트에서 서비스 메타데이터를 검색하기 위해 svcutil 도구를 사용하여 클라이언트가 생성되었습니다.

다음 프로시저에 설명된 대로 샘플은 특정 자원 이름으로 구성됩니다. 자원 이름을 변경해야 하는 경우, 클라이언트 애플리케이션의 `MQ_INSTALLATION_PATH`
`\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\app.config` 파일 및 서비스 애플리케이션의 `MQ_INSTALLATION_PATH`
`\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\TestServices.cs` 파일에서도 해당 값을 변경해야 합니다. 여기서, `MQ_INSTALLATION_PATH`은(는) IBM MQ의 설치 디렉토리입니다. JMS 엔드포인트 URI 형식화에 대한 자세한 정보는 IBM MQ 제품 문서에서 *IBM MQ Transport for SOAP*의 내용을 참조하십시오. 샘플 솔루션과 소스를 수정해야 하는 경우 IDE가 필요합니다(예: Microsoft Visual Studio 8 이상).

프로시저

1. *QM1*이라는 큐 관리자를 작성하십시오.
2. *SampleQ*라는 큐 목적지를 작성하십시오.
3. 리스너가 메시지를 대기하도록 서비스를 시작하십시오. `MQ_INSTALLATION_PATH`
`\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\bin\Release\TestService.exe` 파일을 실행하십시오. 여기서 `MQ_INSTALLATION_PATH`은(는) IBM MQ의 설치 디렉토리입니다.
4. 클라이언트를 한 번 실행하십시오. `MQ_INSTALLATION_PATH`
`\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\bin\Release\TestClient.exe` 파일을 실행하십시오. 여기서 `MQ_INSTALLATION_PATH`은(는) IBM MQ의 설치 디렉토리입니다.
클라이언트 애플리케이션이 5번 루프하여 *SampleQ*에 5개의 메시지를 송신합니다.

결과

서비스 애플리케이션이 *SampleQ*에서 메시지를 가져와 화면에 Hello World를 5번 표시합니다.

다음에 수행할 작업

단순 요청-응답 클라이언트 및 서버 WCF 샘플

이 샘플은 요청-응답 채널 셰이프를 사용하여 WCF 클라이언트에서 WCF (Windows Communication Foundation) 서비스를 시작하는 데 사용되는 IBM MQ 사용자 정의 채널을 보여줍니다.

이 태스크 정보

이 서비스는 두 개의 숫자를 추가하고 뺀 다음 결과를 리턴하는 간단한 계산기 메소드를 제공합니다. [1266 페이지의 『실행 중인 서비스에서 메타데이터와 svcutil 도구를 사용하여 WCF 클라이언트 프록시 및 애플리케이션 구성 파일 생성』](#)에 설명된 별도로 노출된 HTTP 엔드포인트에서 서비스 메타데이터를 검색하기 위해 svcutil 도구를 사용하여 클라이언트가 생성되었습니다.

다음 프로시저에 설명된 대로 샘플은 특정 자원 이름으로 구성됩니다. 자원 이름을 변경해야 하는 경우, 클라이언트 애플리케이션의 `MQ_INSTALLATION_PATH`
`\Tools\wcf\samples\WCF\requestreply\client\app.config` 파일 및 서비스 애플리케이션의 `MQ_INSTALLATION_PATH`
`\Tools\wcf\samples\WCF\requestreply\service\RequestReplyService.cs` 파일에서도 해당 값을 변경해야 합니다. 여기서 `MQ_INSTALLATION_PATH`은(는) IBM MQ의 설치 디렉토리입니다. JMS 엔드포인트 URI 형식화에 대한 자세한 정보는 IBM MQ 제품 문서에서 *IBM MQ Transport for SOAP*의 내용을 참조하십시오. 샘플 솔루션과 소스를 수정해야 하는 경우 IDE가 필요합니다(예: Microsoft Visual Studio 8 이상).

프로시저

1. `QM1`이라는 큐 관리자를 작성하십시오.
2. `SampleQ`라는 큐 목적지를 작성하십시오.
3. `SampleReplyQ`라는 큐 목적지를 작성하십시오.
4. 리스너가 메시지를 대기하도록 서비스를 시작하십시오. `MQ_INSTALLATION_PATH`
`\Tools\wcf\samples\WCF\requestreply\service\bin\Release\SimpleRequestReply_Service.exe` 파일을 실행하십시오. 여기서 `MQ_INSTALLATION_PATH`은(는) IBM MQ의 설치 디렉토리입니다.
5. 클라이언트를 한 번 실행하십시오. `MQ_INSTALLATION_PATH`
`\Tools\wcf\samples\WCF\requestreply\client\bin\Release\SimpleRequestReply_Client.exe` 파일을 실행하십시오. 여기서 `MQ_INSTALLATION_PATH`은(는) IBM MQ의 설치 디렉토리입니다.

결과

클라이언트가 실행된 경우 다음 프로세스가 시작되어 4번 반복되므로, 각 방법으로 총 5개의 메시지가 송신됩니다.

1. 클라이언트가 `SampleQ`에 요청 메시지를 넣고 응답을 기다립니다.
2. 서비스가 `SampleQ`에서 요청 메시지를 가져옵니다.
3. 서비스가 메시지의 콘텐츠를 사용하여 값을 추가하고 뺍니다.
4. 그런 다음 서비스가 `SampleReplyQ`의 메시지에 요청을 넣고 클라이언트가 새 메시지를 넣을 때까지 기다립니다.
5. 클라이언트가 `SampleReplyQ`에서 메시지를 가져오고 화면에 결과를 표시합니다.

다음에 수행할 작업

샘플에서 IBM MQ 호스팅하는 서비스에 .NET 대한 WCF 클라이언트

샘플 클라이언트 애플리케이션과 샘플 서비스 프록시 애플리케이션은 .NET와 Java에 모두 제공됩니다. 이 샘플은 주식 시세 요청을 받은 다음 주식 시세를 제공하는 주식 시세 서비스를 기반으로 합니다.

시작하기 전에

이 샘플에서는 .NET SOAP over JMS 서비스 호스팅 환경이 IBM MQ에 올바르게 설치되어 구성되어 있어야 하며 로컬 큐 관리자에서 액세스할 수 있어야 합니다.

.NET SOAP over JMS 서비스 호스팅 환경이 IBM MQ에 올바르게 설치되어 구성되어 있으며 로컬 큐 관리자에서 액세스할 수 있으면 추가 구성 단계를 완료해야 합니다.

1. 환경 `WMQSOAP_HOME` 변수를 설치 IBM MQ 디렉터리로 설정하십시오. 예를 들어: `C:\Program Files\IBM\MQ`
2. Java 컴파일러 `javac`이(가) 사용 가능하고 `PATH`에 있는지 확인하십시오.
3. 설치 이미지의 `prereqs/axis` 디렉터리에서 `axis.jar` 파일을 프로덕션 IBM MQ 디렉터리로 복사하십시오. 예를 들어: `C:\Program Files\IBM\MQ\java\lib\soap`
4. `PATH`에 추가: `MQ_INSTALLATION_PATH\Java\lib` 여기서, `MQ_INSTALLATION_PATH`은(는) IBM MQ이(가) 설치된 디렉토리를 나타냅니다(예: `C:\Program Files\IBM\MQ`).

5. .NET의 위치가 `MQ_INSTALLATION_PATH\bin\amqwcallsWSDL.cmd`에 제대로 지정되어 있는지 확인하십시오. 여기서, `MQ_INSTALLATION_PATH`은(는) IBM MQ이(가) 설치된 디렉토리를 나타냅니다(예: `C:\Program Files\IBM\MQ`). 의 위치는 예를 들어 다음과 .NET 같이 지정할 수 있습니다: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

이전 단계가 완료되면 서비스를 테스트하고 실행합니다.

1. SOAP over JMS 작업 디렉토리로 이동합니다.

2. 다음 명령 중 하나를 입력하여 확인 테스트를 실행하고 서비스 리스너를 실행 상태로 둡니다.

- For .NET: `MQ_INSTALLATION_PATH\Tools\soap\samples\runivt dotnet hold` 여기서 는 IBM MQ 가 `MQ_INSTALLATION_PATH` 설치된 디렉토리를 나타냅니다.
- AXIS의 경우: `MQ_INSTALLATION_PATH\Tools\soap\samples\runivt Dotnet2AxisClient hold` 여기서 는 IBM MQ 가 `MQ_INSTALLATION_PATH` 설치된 디렉토리를 나타냅니다.

`hold` 인수를 사용하면 테스트가 완료된 후에도 리스너가 계속 실행됩니다.

이 구성 중에 오류가 보고되면 다음 방식으로 프로시저를 재시작하도록 모든 변경을 제거할 수 있습니다.

1. 생성된 SOAP over JMS 디렉토리를 삭제하십시오.
2. 큐 관리자를 삭제합니다.

이 태스크 정보

이 샘플은 WCF 클라이언트가 SOAP over JMS.NET 샘플 서비스에 연결하는 방법을 보여줍니다. 이 연결은 일방향 채널 형식을 사용하여 IBM MQ 수행됩니다. 서비스는 단순 StockQuote 예를 구현하며, 이는 텍스트 문자열을 콘솔에 출력합니다.

[1267 페이지의 『WSDL과 svcutil을 사용하여 WCF 클라이언트 프록시와 애플리케이션 구성 파일 생성』](#)에 설명된 대로 클라이언트 파일을 생성하기 위해 WSDL을 사용하여 클라이언트가 생성되었습니다.

다음 프로시저에 설명된 대로 샘플은 특정 자원 이름으로 구성됩니다. 자원 이름을 변경해야 하는 경우, 클라이언트 애플리케이션의 `MQ_INSTALLATION_PATH`

`\tools\wcf\samples\WMQNET\default\client\app.config` 파일 및 서비스 애플리케이션의 `MQ_INSTALLATION_PATH`

`\tools\wcf\samples\WMQNET\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl` 1 파일에서도 해당 값을 변경해야 합니다. 여기서 `MQ_INSTALLATION_PATH`은(는) IBM MQ의 설치 디렉토리를 나타냅니다. JMS 엔드포인트 URI 형식화에 대한 자세한 정보는 IBM MQ 제품 문서에서 *IBM MQ Transport for SOAP*를 참조하십시오.

프로시저

클라이언트를 한 번 실행하십시오. `MQ_INSTALLATION_PATH`

`\tools\wcf\samples\WMQNET\default\client\bin\Release\TestClient.exe` 파일을 실행하십시오. 여기서 `MQ_INSTALLATION_PATH`은(는) IBM MQ의 설치 디렉토리를 나타냅니다.

클라이언트 애플리케이션은 5번 루프를 실행하여 샘플 큐에 5개의 메시지를 송신합니다.

결과

서비스 애플리케이션이 샘플 큐에서 메시지를 가져오고 화면에 Hello World를 5번 표시합니다.

샘플에서 IBM MQ 호스팅하는 Axis Java 서비스에 대한 WCF 클라이언트

샘플 클라이언트 애플리케이션과 샘플 서비스 프록시 애플리케이션은 Java와 .NET에 모두 제공됩니다. 이 샘플은 주식 시세 요청을 받은 다음 주식 시세를 제공하는 주식 시세 서비스를 기반으로 합니다.

시작하기 전에

이 샘플에서는 .NET SOAP over JMS 서비스 호스팅 환경이 IBM MQ에 올바르게 설치되어 구성되어 있어야 하며 로컬 큐 관리자에서 액세스할 수 있어야 합니다.

.NET SOAP over JMS 서비스 호스팅 환경이 IBM MQ에 올바르게 설치되어 구성되어 있으며 로컬 큐 관리자에서 액세스할 수 있으면 추가 구성 단계를 완료해야 합니다.

1. 환경 **WMQSOAP_HOME** 변수를 설치 IBM MQ 디렉터리로 설정하십시오. 예를 들어: C:\Program Files\IBM\MQ
2. Java 컴파일러 javac이(가) 사용 가능하고 PATH에 있는지 확인하십시오.
3. 설치 이미지의 prereqs/axis 디렉터리에서 axis.jar 파일을 설치 IBM MQ 디렉터리로 복사하십시오.
4. PATH에 추가: **MQ_INSTALLATION_PATH\Java\lib** 여기서, **MQ_INSTALLATION_PATH**은(는) IBM MQ이(가) 설치된 디렉토리를 나타냅니다(예: C:\Program Files\IBM\MQ).
5. .NET의 위치가 **MQ_INSTALLATION_PATH\bin\amqwcallsd\cmd**에 제대로 지정되어 있는지 확인하십시오. 여기서, **MQ_INSTALLATION_PATH**은(는) IBM MQ이(가) 설치된 디렉토리를 나타냅니다(예: C:\Program Files\IBM\MQ). 의 위치는 예를 들어 다음과 .NET 같이 지정할 수 있습니다: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

이전 단계가 완료되면 서비스를 테스트하고 실행합니다.

1. SOAP over JMS 작업 디렉터리로 이동합니다.
2. 다음 명령 중 하나를 입력하여 확인 테스트를 실행하고 서비스 리스너를 실행 상태로 둡니다.
 - For .NET: **MQ_INSTALLATION_PATH\Tools\soap\samples\runivt dotnet hold** 여기서 는 IBM MQ 가 **MQ_INSTALLATION_PATH** 설치된 디렉토리를 나타냅니다.
 - AXIS의 경우: **MQ_INSTALLATION_PATH\Tools\soap\samples\runivt Dotnet2AxisClient hold** 여기서 는 IBM MQ 가 **MQ_INSTALLATION_PATH** 설치된 디렉토리를 나타냅니다.

hold 인수를 사용하면 테스트가 완료된 후에도 리스너가 계속 실행됩니다.

이 구성 중에 오류가 보고되면 다음 방식으로 프로시저를 재시작하도록 모든 변경을 제거할 수 있습니다.

1. 생성된 SOAP over JMS 디렉토리를 삭제하십시오.
2. 큐 관리자를 삭제합니다.

이 태스크 정보

이 샘플은 WCF 클라이언트가 Axis Java SOAP over JMS 샘플 서비스에 제공하는 일방향 채널 IBM MQ 형식을 사용하여 연결하는 방법을 보여줍니다. 서비스는 단순 StockQuote 예를 구현하며, 이는 현재 디렉토리에 저장된 파일로 텍스트 문자열을 출력합니다.

1267 페이지의 『WSDL과 svcutil을 사용하여 WCF 클라이언트 프록시와 애플리케이션 구성 파일 생성』에 설명된 대로 클라이언트 파일을 생성하기 위해 WSDL을 사용하여 클라이언트가 생성되었습니다.

이 샘플은 이 단락에 설명된 대로 특정 자원 이름으로 구성됩니다. 자원 이름을 변경해야 하는 경우, 클라이언트 애플리케이션의 **MQ_INSTALLATION_PATH**
\tools\wcf\samples\WMQAxis\default\client\app.config 파일 및 서비스 애플리케이션의 **MQ_INSTALLATION_PATH**
\tools\wcf\samples\WMQAxis\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl 파일에서도 해당 값을 변경해야 합니다. 여기서 **MQ_INSTALLATION_PATH**은(는) IBM MQ의 설치 디렉토리를 나타냅니다.

프로시저

클라이언트를 한 번 실행하십시오. **MQ_INSTALLATION_PATH**
\tools\wcf\samples\WMQAxis\default\client\bin\Release\TestClient.exe 파일을 실행하십시오. 여기서 **MQ_INSTALLATION_PATH**은(는) IBM MQ의 설치 디렉토리를 나타냅니다.
클라이언트 애플리케이션은 5번 루프를 실행하여 샘플 큐에 5개의 메시지를 송신합니다.

결과

서비스 애플리케이션이 샘플 큐에서 메시지를 가져오고 현재 디렉토리의 파일에 Hello World를 5번 추가합니다.

WebSphere Application Server 샘플에서 호스트하는 Java 서비스에 대한 WCF 클라이언트

샘플 클라이언트 애플리케이션과 샘플 서비스 프록시 애플리케이션은 WebSphere Application Server 6에 제공됩니다. 요청-응답 서비스도 제공됩니다.

시작하기 전에

이 샘플은 다음 IBM MQ 구성도 사용해야 합니다.

표 205. IBM MQ 필수 구성	
오브젝트	필수 이름
큐 관리자	QM1
로컬 큐	HelloWorld
로컬 큐	HelloWorldReply

이 샘플에는 WebSphere Application Server 6 호스트 환경도 올바르게 설치되어 구성되어야 합니다. WebSphere Application Server 6에서는 기본적으로 바인딩 모드 연결을 사용하여 IBM MQ에 연결합니다. 그러므로 WebSphere Application Server 6은 큐 관리자와 동일한 시스템에 설치되어야 합니다.

WAS 환경이 구성된 후 다음 추가 구성 단계를 완료해야 합니다.

1. WebSphere Application Server JNDI 저장소에서 다음 JNDI 오브젝트를 작성하십시오.
 - a. HelloWorld 라는 JMS 큐 대상
 - JNDI 이름을 `.jms/HelloWorld`로 설정하십시오.
 - 큐 이름을 HelloWorld로 설정하십시오.
 - b. HelloWorldQCF 라는 JMS 큐 연결 팩토리
 - JNDI 이름을 `.jms/HelloWorldQCF`로 설정하십시오.
 - 큐 관리자 이름을 QM1로 설정하십시오.
 - c. WebServicesReplyQCF 라는 JMS 큐 연결 팩토리
 - JNDI 이름을 `.jms/WebServicesReplyQCF`로 설정하십시오.
 - 큐 관리자 이름을 QM1로 설정하십시오.
2. 다음 구성을 사용하여 WebSphere Application Server 에서 HelloWorldPort 라는 메시지 리스너 포트를 작성하십시오.
 - 연결 팩토리 JNDI 이름을 `.jms/HelloWorldQCF`로 설정하십시오.
 - 목적지 JNDI 이름을 `.jms/HelloWorld`로 설정하십시오.
3. 다음과 같이 WebSphere Application Server에 웹 서비스 HelloWorldEJBear.ear 애플리케이션을 설치하십시오.
 - a. 애플리케이션 > 새 애플리케이션 > 새 엔터프라이즈 애플리케이션을 클릭하십시오.
 - b. `MQ_INSTALLATION_PATH\tools\wcf\samples\WAS\HelloWorldsEJBear.ear`(으)로 이동하십시오. 여기서, `MQ_INSTALLATION_PATH`은(는) IBM MQ의 설치 디렉토리입니다.
 - c. 마법사에서 기본 옵션을 변경하지 않고 애플리케이션을 설치한 다음 애플리케이션을 재시작하십시오.

WAS 구성이 완료되면 다음과 같이 한 번 실행하여 서비스를 테스트하십시오.

1. SOAP over JMS 작업 디렉토리로 이동합니다.
2. 다음 명령을 입력하여 샘플을 실행하십시오. `MQ_INSTALLATION_PATH\tools\wcf\samples\WAS\TestClient.exe`. 여기서 `MQ_INSTALLATION_PATH`는 IBM MQ의 설치 디렉토리입니다.

이 태스크 정보

이 샘플은 요청-응답 채널 셰이프를 사용하여 WCF 클라이언트에서 IBM MQ에 포함된 WCF 샘플에 제공된 WebSphere Application Server SOAP over JMS 샘플 서비스로의 연결을 보여줍니다. 메시지는 IBM MQ 큐를 사용하여 WCF와 WebSphere Application Server 간에 플로우됩니다. 이 서비스는 문자열을 사용하여 클라이언트에 인사말을 리턴하는 HelloWorld(...) 메소드를 구현합니다.

클라이언트는 1266 페이지의 『실행 중인 서비스에서 메타데이터와 svcutil 도구를 사용하여 WCF 클라이언트 프록시 및 애플리케이션 구성 파일 생성』에 설명된 대로 개별적으로 공개된 HTTP 엔드포인트에서 서비스 메타데이터를 검색하기 위해 svcutil 도구를 사용하여 생성됩니다.

다음 프로시저에 설명된 대로 샘플은 특정 자원 이름으로 구성됩니다. 자원 이름을 변경해야 하는 경우, 클라이언트 애플리케이션의 `MQ_INSTALLATION_PATH`

`\tools\wcf\samples\WAS\default\client\app.config` 파일 및 서비스 애플리케이션의 `MQ_INSTALLATION_PATH \tools\wcf\samples\WAS\HelloWorldsEJBear.ear`에서도 해당 값을 변경해야 합니다. 여기서, `MQ_INSTALLATION_PATH`은(는) IBM MQ의 설치 디렉토리입니다.

서비스 및 클라이언트는 IBM Developer 기사 *SOAP over JMS* 및 *WebSphere Studio*를 사용하여 JMS 웹 서비스 빌드에 요약된 서비스 및 클라이언트를 기반으로 합니다. IBM MQ WCF 사용자 정의 채널과 호환 가능한 SOAP over JMS 웹 서비스 개발에 대한 자세한 정보는 https://www.ibm.com/developerworks/websphere/library/techarticles/0402_du/0402_du.html의 내용을 참조하십시오.

프로시저

클라이언트를 한 번 실행하십시오. `MQ_INSTALLATION_PATH`

`\tools\wcf\samples\WAS\default\client\bin\Release\TestClient.exe` 파일을 실행하십시오. 여기서 `MQ_INSTALLATION_PATH`은(는) IBM MQ의 설치 디렉토리입니다.

클라이언트 애플리케이션에서 동시에 두 서비스 메소드를 모두 시작하여 두 메시지를 샘플 큐에 송신합니다.

결과

서비스 애플리케이션은 샘플 큐에서 메시지를 가져오고 클라이언트 애플리케이션이 콘솔에 출력하는 HelloWorld(...) 메소드 호출에 응답을 제공합니다.

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 라이선스까지 부여하는 것은 아닙니다. 라이선스에 대한 의문사항은 다음으로 문의하십시오.

07326

서울특별시 영등포구
국제금융로 10, 3IFC
한국 아이.비.엠 주식회사
U.S.A.

2바이트(DBCS) 정보에 관한 라이선스 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 라이선스 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 라이선스가 부여된 프로그램 및 프로그램에 대해 사용 가능한 모든 라이선스가 부여된 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이선스 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 추정

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 라이선스:

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 소프트웨어로 확인하는 경우에는 사진과 컬러 삽화가 제대로 나타나지 않을 수도 있습니다.

프로그래밍 인터페이스 정보

제공되는 경우, 프로그래밍 인터페이스 정보는 본 프로그램과 함께 사용할 애플리케이션 소프트웨어를 개발하는데 도움을 주기 위한 것입니다.

이 책에는 고객이 IBM MQ의 서비스를 이용하기 위한 프로그램을 작성할 수 있도록 하는 의도된 프로그래밍 인터페이스에 대한 정보가 포함되어 있습니다.

그러나 본 정보에는 진단, 수정 및 성능 조정 정보도 포함되어 있습니다. 진단, 수정 및 성능 조정 정보는 응용프로그램 소프트웨어의 디버거를 돕기 위해 제공된 것입니다.

중요사항: 이 진단, 수정 및 튜닝 정보는 변경될 수 있으므로 프로그래밍 인터페이스로 사용하지 마십시오.

상표

IBM, 로고, ibm.com® IBM, 는 IBM Corporation의 상표로, 전 세계 여러 관할권에 등록되어 있습니다. 현재 상표 IBM 목록은 웹사이트 "저작권 및 상표 정보" 페이지(www.ibm.com/legal/copytrade.shtml)에서 확인할 수 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.

본 제품에는 리눅스 커널 프로젝트(Eclipse, <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: