9.3

# *Tworzenie aplikacji dla produktu IBM MQ*

**Uwaga**

Przed skorzystaniem z niniejszych informacji oraz produktu, którego one dotyczą, należy zapoznać się z informacjami zamieszczonymi w sekcji “Uwagi” na stronie 1333.

Niniejsze wydanie publikacji dotyczy wersji 9, wydania 3 produktu IBM® MQ oraz wszystkich jego późniejszych wydań i modyfikacji, aż do odwołania w nowych wydaniach publikacji.

# Spis treści

# Tworzenie aplikacji dla składnika IBM MQ

Użytkownik może tworzyć aplikacje do wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i zasobami pokrewnymi. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i środowiskach.

## Czy jesteś nowym użytkownikiem w tworzeniu aplikacji dla produktu IBM MQ?

Aby uzyskać więcej informacji na temat tworzenia aplikacji dla produktu IBM MQ, należy odwiedzić serwis IBM Developer:

- IBM MQ Developer Essentials *(podstawowe informacje, uruchamianie dema, tworzenie kodu aplikacji, bardziej zaawansowane kursy)*
- IBM MQ Zasoby do pobrania dla programistów *(w tym bezpłatne edycje dla programistów i wersje próbne)*

Łatwiej jest również tworzyć aplikacje, jeśli użytkownik zna pojęcia opisane w następujących sekcjach:

- “Pojęcia związane z projektowaniem aplikacji” na stronie 7
- “Uwagi dotyczące projektowania dla aplikacji IBM MQ” na stronie 51

## Obsługa języków i środowisk obiektowych

Produkt IBM MQ zapewnia podstawową obsługę aplikacji opracowanych w następujących językach i środowiskach:

- JMS
- Java
- C++
- .NET

Patrz także “Aplikacje obiektowe” na stronie 16.

Produkt .NET obsługuje aplikacje opracowane w wielu językach. W celu zilustrowania korzystania z klas IBM MQ classes for .NET w celu uzyskania dostępu do kolejek systemu IBM MQ dokumentacja produktu MQ zawiera informacje dla następujących języków:

- C# kod przykładowy i aplikacje przykładowe
- Aplikacje przykładowe C++
- Przykładowe aplikacjeVisual Basic

Patrz sekcja “Pisanie i wdrażanie programów IBM MQ .NET” na stronie 598.

Produkt IBM MQ obsługuje moduł podstawowy .NET dla aplikacji w środowiskach Windows z produktu IBM MQ 9.1.1 oraz dla aplikacji w środowiskach Linux® z produktu IBM MQ 9.1.2. Więcej informacji na ten temat zawiera sekcja “Instalowanie produktu IBM MQ classes for .NET” na stronie 574.

Produkt IBM MQ obsługuje również klienty AMQP, które implementują protokół OASIS AMQP 1.0 .

MQ Light, Apache Klienty Qpid, takie jak Apache Qpid Proton i Apache Qpid JMS , są oparte na tym protokole.

Interfejsy API MQ Light są dostępne pod adresem IBM MQ Light.

Klienty Qpid Apache są dostępne pod adresem QPid Proton.

Następujące powiązania językowe są udostępniane w postaci, w jakiej są dostępne:

- Powiązanie Idź

- implementacja interfejsu API JavaScript współpracującego z aplikacjami Node.js

## Obsługa programowych interfejsów REST API

Produkt IBM MQ udostępnia obsługę następujących programistycznych interfejsów REST API na potrzeby wysyłania i odbierania komunikatów:

- IBM MQ messaging REST API
- z/OS IBM z/OS Connect EE
- IBM Integration Bus
- IBM DataPower Brama

Należy zapoznać się z sekcją “Tworzenie aplikacji REST przy użyciu produktu IBM MQ” na stronie 725oraz kursem Pierwsze kroki z interfejsem API REST przesyłania komunikatów produktu IBM MQ w obszarze IBM MQ produktu IBM Developer. Ten kurs zawiera przykłady w następujących językach, udostępnionych w stanie, w jakim się znajdują ("as is"), do użycia z produktem IBM MQ messaging REST API:

- Przykład użycia interfejsu REST API przesyłania komunikatów produktu MQ
- Przykład Node.js z użyciem modułu HTTPS
- Przykład Node.js z modułem Promise

## Obsługa proceduralnych języków programowania

Produkt IBM MQ zapewnia obsługę aplikacji opracowanych w następujących językach programowania proceduralnego:

- C
- Windows Visual Basic (tylko w systemachWindows )
- kompilatory
- z/OS Assembler (tylko w systemieIBM MQ for z/OS )
- z/OS PL/I (tylkoIBM MQ for z/OS )
- IBM i RPG (tylkoIBM MQ for IBM i )

Te języki korzystają z interfejsu kolejki komunikatów (MQI) w celu uzyskania dostępu do usług kolejkowania komunikatów. Patrz sekcja “Tworzenie aplikacji MQI przy użyciu produktu IBM MQ” na stronie 740. Należy zauważyć, że model obiektowy IBM MQ Object Model, używany przez obiektowe języki i struktury, udostępnia dodatkowe funkcje, które nie są dostępne dla języków proceduralnych korzystających z interfejsu MQI.

## Określanie nazwy aplikacji

ALW

Przed wprowadzeniem wersji IBM MQ 9.1.2można było określić nazwę aplikacji w aplikacjach klienckich w systemie Java lub JMS . W programie IBM MQ 9.1.2można również określić nazwę aplikacji w dodatkowych językach programowania. Aby uzyskać więcej informacji, zapoznaj się z sekcją: “Określanie nazwy aplikacji w obsługiwanych językach programowania” na stronie 54.

### Zadania pokrewne

“Tworzenie aplikacji dla składnika MQ Telemetry” na stronie 1276

“Tworzenie aplikacji Microsoft Windows Communication Foundation przy użyciu języka IBM MQ” na stronie 1300

Kanał niestandardowy produktu Microsoft Windows Communication Foundation (WCF) dla produktu IBM MQ wysyła i odbiera komunikaty między klientami i usługami WCF.

**Odsyłacze pokrewne**

"Tworzenie aplikacji dla składnika Managed File Transfer" na stronie 1254
Określ programy, które mają być uruchamiane z systemem Managed File Transfer, użyj opcji Apache Ant z opcją Managed File Transfer, dopasuj opcję Managed File Transfer do procedur zewnętrznych i steruj opcją Managed File Transfer , umieszczając komunikaty w kolejce komend agenta.

# Pojęcia związane z projektowaniem aplikacji

Do pisania aplikacji IBM MQ można użyć języka proceduralnego lub obiektowego. Przed rozpoczęciem projektowania i pisania aplikacji IBM MQ należy zapoznać się z podstawowymi pojęciami związanymi z produktem IBM MQ .

Informacje na temat typów aplikacji, które można pisać dla IBM MQ, zawierają "Tworzenie aplikacji dla składnika IBM MQ" na stronie 5 i "Działania, które mogą być wykonywane przez aplikacje" na stronie 7.

**Pojęcia pokrewne**

"Uwagi dotyczące projektowania dla aplikacji IBM MQ" na stronie 51
Po podjęciu decyzji, w jaki sposób aplikacje mogą korzystać z dostępnych platform i środowisk, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

## Działania, które mogą być wykonywane przez aplikacje

Użytkownik może tworzyć aplikacje do wysyłania i odbierania komunikatów, które są potrzebne do obsługi procesów biznesowych. Można również tworzyć aplikacje do zarządzania menedżerami kolejek i zasobami pokrewnymi.

### Działania, które aplikacje mogą wykonywać w systemie IBM MQ for Multiplatforms

Multi

W systemie Wiele platformmożna pisać aplikacje, które wykonują następujące działania:

- Wysyłanie komunikatów do innych aplikacji działających w tych samych systemach operacyjnych. Aplikacje mogą znajdować się w tym samym lub innym systemie.
- Wysyłanie komunikatów do aplikacji działających na innych platformach IBM MQ .
- Kolejkowania komunikatów z produktu CICS należy używać w następujących systemach:
  - AIX TXSeries dla AIX
  - IBM i IBM i
  - Windows Windows
- Użyj kolejkowania komunikatów z Encina dla następujących systemów:
  - AIX AIX
  - Windows Windows
- Kolejkowania komunikatów z programu Tuxedo należy używać w następujących systemach:
  - AIX AIX
  - O & T
  - Windows Windows
- Użyj programu IBM MQ jako menedżera transakcji, koordynując aktualizacje wprowadzane przez zewnętrzne menedżery zasobów w obrębie jednostek pracy systemu IBM MQ . Następujące zewnętrzne menedżery zasobów są obsługiwane i zgodne z interfejsem XA X/OPEN
  - Db2
  - Informix

  - Oracle
  - Sybase
- Przetwarzanie kilku komunikatów razem jako pojedynczej jednostki pracy, która może zostać zatwierdzona lub wycofana.
- Uruchom w pełnym środowisku IBM MQ lub w środowisku klienta IBM MQ .

## Działania, które aplikacje mogą wykonywać w systemie IBM MQ for z/OS

z/OS

W systemie z/OSmożna pisać aplikacje, które wykonują następujące działania:

- Użyj kolejkowania komunikatów w programie CICS lub IMS.
- Wysyłanie komunikatów między aplikacjami wsadowymi, CICSi IMS , wybierając najbardziej odpowiednie środowisko dla każdej funkcji.
- Wysyłanie komunikatów do aplikacji działających na innych platformach IBM MQ .
- Przetwarzanie kilku komunikatów razem jako pojedynczej jednostki pracy, która może zostać zatwierdzona lub wycofana.
- Wysyłanie komunikatów do aplikacji IMS i interakcja z nimi za pośrednictwem mostu IMS .
- Udział w jednostkach pracy koordynowanych przez RRS.

Każde środowisko w produkcie z/OS ma własne właściwości, zalety i wady. Zaletą produktu IBM MQ for z/OS jest to, że aplikacje nie są powiązane z żadnym środowiskiem, ale mogą być dystrybuowane w celu skorzystania z zalet poszczególnych środowisk. Można na przykład tworzyć interfejsy użytkownika końcowego za pomocą TSO lub CICS, uruchamiać moduły intensywnie przetwarzające w zadaniu wsadowym z/OS oraz uruchamiać aplikacje bazodanowe w systemie IMS lub CICS. We wszystkich przypadkach różne części aplikacji mogą komunikować się za pomocą komunikatów i kolejek.

Projektanci aplikacji IBM MQ muszą być świadomi różnic i ograniczeń narzucanych przez te środowiska. Na przykład:

- Produkt IBM MQ udostępnia narzędzia, które umożliwiają komunikację międzymenedżerową między menedżerami kolejek (jest to nazywane *kolejkowaniem rozproszonym*).
- Metody zatwierdzania i wycofywania zmian różnią się w różnych środowiskach wsadowych i CICS .
- IBM MQ for z/OS zapewnia obsługę w środowisku IMS programów przetwarzania komunikatów (MPP), interaktywnych programów krótkiej ścieżki (IFP) i programów przetwarzania komunikatów wsadowych (BMP). W przypadku pisania programów wsadowych w języku DL/I należy postępować zgodnie ze wskazówkami podanymi w tematach takich jak “Budowanie aplikacji wsadowych z/OS” na stronie 1054 i “Uwagi dotyczące zadań wsadowych dla systemu z/OS” na stronie 755 dla programów wsadowych w języku z/OS .
- Chociaż w jednym systemie z/OS może istnieć wiele instancji programu IBM MQ for z/OS , region CICS może w danym momencie łączyć się tylko z jednym menedżerem kolejek. Jednak więcej niż jeden region CICS może być połączony z tym samym menedżerem kolejek. W środowiskach wsadowych systemów IMS i z/OS programy mogą łączyć się z więcej niż jednym menedżerem kolejek.
- Produkt IBM MQ for z/OS umożliwia współużytkowanie kolejek lokalnych przez grupę menedżerów kolejek, zapewniając większą przepustowość i dostępność. Takie kolejki są nazywane *kolejkami współużytkowanymi*, a menedżery kolejek tworzą *grupę współużytkowania kolejek*, która może przetwarzać komunikaty w tych samych kolejkach współużytkowanych. Aplikacje wsadowe mogą łączyć się z jednym z kilku menedżerów kolejek w obrębie grupy współużytkowania kolejek, określając nazwę grupy współużytkowania kolejek zamiast określonej nazwy menedżera kolejek. Jest to nazywane *przyłączanie wsadowe grupy*lub po prostu *przyłączanie grupowe*. Patrz Kolejki współużytkowane i grupy współużytkowania kolejek.

z/OS Różnice między obsługiwanymi środowiskami i ich ograniczeniami zostały opisane w sekcji “Używanie i pisanie aplikacji w systemie IBM MQ for z/OS” na stronie 916.

**Pojęcia pokrewne**

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć języka proceduralnego lub obiektowego. Przed rozpoczęciem projektowania i pisania aplikacji IBM MQ należy zapoznać się z podstawowymi pojęciami związanymi z produktem IBM MQ .

“Uwagi dotyczące projektowania dla aplikacji IBM MQ” na stronie 51
Po podjęciu decyzji, w jaki sposób aplikacje mogą korzystać z dostępnych platform i środowisk, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

“Pisanie aplikacji proceduralnej dotyczącej kolejkowania” na stronie 744
Ten temat zawiera informacje o pisaniu aplikacji kolejkowania, nawiązywaniu i rozłączaniu połączeń z menedżerem kolejek, publikowaniu i subskrybowaniu oraz otwieraniu i zamykaniu obiektów.

“Pisanie aplikacji proceduralnych klienta” na stronie 941
Informacje potrzebne do pisania aplikacji klienckich w systemie IBM MQ przy użyciu języka proceduralnego.

“Korzystanie z IBM MQ classes for JMS/Jakarta Messaging” na stronie 84
IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging są dostawcami przesyłania komunikatów produktu Java dostarczanymi wraz z produktem IBM MQ. Oprócz implementowania interfejsów zdefiniowanych w specyfikacjach JMS i Jakarta Messaging dostawcy przesyłania komunikatów dodają dwa zestawy rozszerzeń do interfejsu API przesyłania komunikatów produktu Java .

“użycieIBM MQ classes for Java” na stronie 357
Użyj IBM MQ w środowisku Java . IBM MQ classes for Java Zezwalaj aplikacji Java na nawiązywanie połączenia z produktem IBM MQ jako klientem IBM MQ lub nawiązywanie połączenia bezpośrednio z menedżerem kolejek produktu IBM MQ .

“Tworzenie aplikacji .NET” na stronie 573
IBM MQ classes for .NET zezwól aplikacjom .NET na nawiązywanie połączeń z produktem IBM MQ jako IBM MQ MQI client lub nawiązywanie połączeń bezpośrednio z serwerem IBM MQ .

“Tworzenie aplikacji w języku C++” na stronie 545
IBM MQ udostępnia klasy C++ równoważne obiektom IBM MQ i pewne dodatkowe klasy równoważne tablicom typów danych. Udostępnia on wiele funkcji, które nie są dostępne za pośrednictwem interfejsu MQI.

“Budowanie aplikacji proceduralnej” na stronie 1029
Użytkownik może napisać aplikację IBM MQ w jednym z kilku języków proceduralnych i uruchomić ją na kilku różnych platformach.

**Zadania pokrewne**

“Korzystanie z przykładowych programów proceduralnych IBM MQ” na stronie 1089
Te przykładowe programy zostały napisane w językach proceduralnych i przedstawiają typowe zastosowania interfejsu kolejek komunikatów (Message Queue Interface-MQI). Programy IBM MQ na różnych platformach.

“Tworzenie aplikacji Microsoft Windows Communication Foundation przy użyciu języka IBM MQ” na stronie 1300
Kanał niestandardowy produktu Microsoft Windows Communication Foundation (WCF) dla produktu IBM MQ wysyła i odbiera komunikaty między klientami i usługami WCF.

Zabezpieczanie

Multi

# Aplikacje, nazwy aplikacji i instancje aplikacji

Przed rozpoczęciem projektowania i pisania aplikacji należy zapoznać się z podstawowymi pojęciami dotyczącymi aplikacji, nazw aplikacji i instancji aplikacji.

## Aplikacje

Multi

Połączenia z menedżerem kolejek są uznawane za pochodzące z tej samej *aplikacji* , jeśli udostępniają taką samą *nazwę aplikacji*. Nazwa aplikacji jest wyświetlana jako atrybut APPLTAG komendy DISPLAY CONN (*) TYPE CONN.

**Uwagi:**

1. W przypadku aplikacji korzystających z wersji produktu IBM MQ client wcześniejszej niż IBM MQ 9.1.2nazwa aplikacji jest automatycznie ustawiana przez serwer IBM MQ client. Jego wartość zależy od języka programowania aplikacji i platformy, na której działa aplikacja. Więcej informacji na ten temat zawiera sekcja PutApplName .
2. W przypadku aplikacji IBM MQ client używających serwera IBM MQ client w wersji IBM MQ 9.1.2 lub nowszej można ustawić nazwę aplikacji na konkretną wartość. W większości przypadków nie wymaga to wprowadzania zmian w kodzie aplikacji ani rekompilacji aplikacji. Więcej informacji na ten temat zawiera sekcja “Używanie nazwy aplikacji w obsługiwanych językach programowania” na stronie 55.

### Instancje aplikacji

Multi

Połączenia są dalej dzielone na *instancje aplikacji*. Instancja aplikacji jest zestawem ściśle powiązanych połączeń, które udostępniają jedną jednostkę wykonania dla tej aplikacji. Zwykle jest to pojedynczy proces systemu operacyjnego, który może mieć wiele wątków i powiązanych połączeń IBM MQ .

W systemie IBM MQ for Multiplatforms instancja aplikacji jest powiązana z konkretnym znacznikiem połączenia. Menedżer kolejek automatycznie wiąże nowe połączenia z istniejącą instancją aplikacji, gdy widzi, że są one ze sobą powiązane.

**Uwagi:**

- Jeśli używane są połączenia klienckie, procesy te mogą łączyć się z menedżerem kolejek za pośrednictwem co najmniej jednego działającego kanału.
- W aplikacjach JMS instancja aplikacji jest odwzorowywana na konkretne połączenie JMS i wszystkie powiązane sesje JMS .

Instancje aplikacji są szczególnie ważne w systemie IBM MQ for Multiplatforms w przypadku korzystania z jednolitego klastra automatycznego równoważenia aplikacji. Na platformach IBM MQ for Multiplatforms można wyświetlać obecnie połączone instancje aplikacji za pomocą komendy DISPLAY APSTATUS .

W niektórych przypadkach menedżer kolejek nie może poprawnie nawiązać połączenia z powiązaniem instancji aplikacji, w szczególności:

- Jeśli w konwersacji współużytkowanej z tego samego procesu nawiązanych jest wiele połączeń, należy użyć różnych nazw aplikacji.
- Jeśli używane są biblioteki klienta starszej wersji. Na przykład instalacje klienta IBM MQ JMS w systemie IBM MQ 9.1.2 i w wersjach wcześniejszych.

W takich sytuacjach, jeśli aplikacje nie definiują się jako możliwe do ponownego połączenia, będzie to dozwolone, ale niektóre grupy instancji aplikacji mogą być niepoprawne. Jeśli dowolne z połączeń są zadeklarowane jako MQCNO_RECONNECT, ma to znaczący negatywny wpływ na równoważenie aplikacji. Wywołanie MQCONN zostanie odrzucone z opcją MQCNO_RECONNECT_NIEKOMPATYBILNĄ.

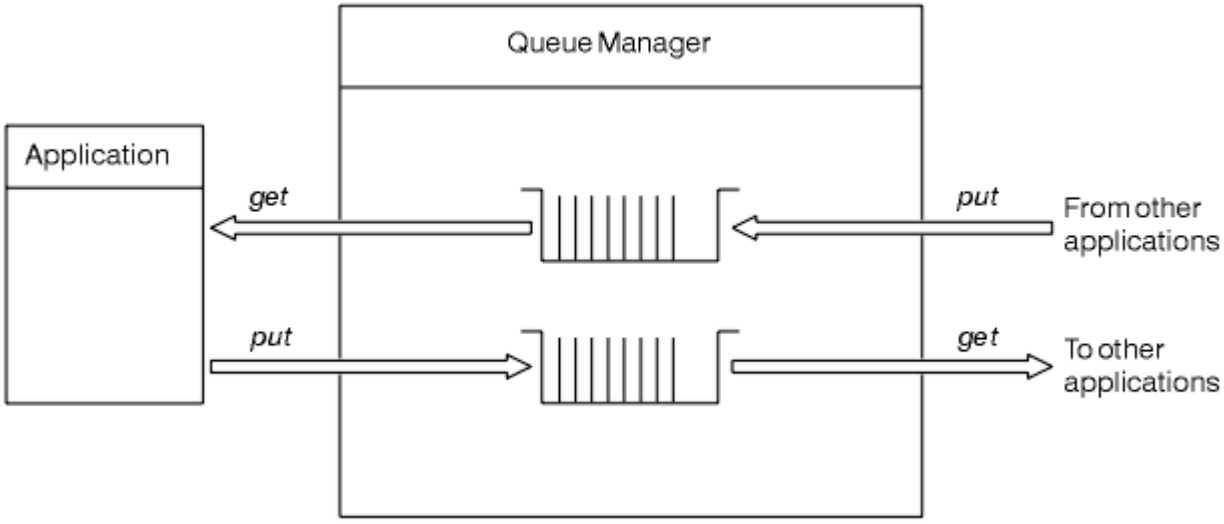
**Pojęcia pokrewne**

“Określanie nazwy aplikacji w obsługiwanych językach programowania” na stronie 54
Przed wprowadzeniem IBM MQ 9.2.0można było określić nazwę aplikacji w aplikacjach klienckich w systemie Java lub JMS . Począwszy od wersji IBM MQ 9.2.0 ta funkcja jest rozszerzona o inne języki programowania w systemie IBM MQ for Multiplatforms.

## Aplikacje korzystające z interfejsu MQI

Aplikacje IBM MQ wymagają pewnych obiektów, zanim będą mogły działać poprawnie.

Rysunek 1 na stronie 11 przedstawia aplikację, która usuwa komunikaty z kolejki, przetwarza je, a następnie wysyła wyniki do innej kolejki w tym samym menedżerze kolejek.




*Rysunek 1. Kolejki, komunikaty i aplikacje*

Aplikacje mogą umieszczać komunikaty w kolejkach lokalnych lub zdalnych (za pomocą komendy MQPUT ), ale mogą tylko pobrać komunikaty bezpośrednio z kolejek lokalnych (za pomocą komendy MQGET ).

Przed uruchomieniem tej aplikacji należy spełnić następujące warunki:

- Menedżer kolejek musi istnieć i być uruchomiony.
- Należy zdefiniować pierwszą kolejkę aplikacji, z której mają zostać usunięte komunikaty.
- Należy również zdefiniować drugą kolejkę, w której aplikacja umieszcza komunikaty.
- Aplikacja musi mieć możliwość nawiązania połączenia z menedżerem kolejek. W tym celu musi być połączony z IBM MQ. Patrz “Budowanie aplikacji proceduralnej” na stronie 1029.
- Aplikacje, które umieszczają komunikaty w pierwszej kolejce, muszą również łączyć się z menedżerem kolejek. Jeśli są zdalne, muszą być również skonfigurowane z kolejkami transmisji i kanałami. Ta część systemu nie jest wyświetlana na rysunku ( Rysunek 1 na stronie 11).

## Używanie połączeń klienckich do nawiązywania połączeń z wieloma menedżerami kolejek produktu IBM MQ

Istnieje możliwość skonfigurowania aplikacji połączonych z klientem w taki sposób, aby łączyły się z więcej niż jednym menedżerem kolejek (ze względu na równoważenie obciążenia lub dostępność usługi).

Podstawowe mechanizmy umożliwiające osiągnięcie tego celu w kliencie IBM MQ to użycie tabel definicji kanału klienta, patrz sekcja Konfigurowanie tabel definicji kanału klientalub listy połączeń.

Można również uzyskać podobne zachowanie przy użyciu zewnętrznych produktów do równoważenia obciążenia lub opakowując kod połączenia IBM MQ w kod pośredniczący, który może przekierowywać nazwy hostów lub adresy IP.

Każda z tych technik ma pewne ograniczenia i może być mniej lub bardziej odpowiednia dla konkretnych wymagań aplikacji. Poniższe sekcje, chociaż nie wyczerpujące, opisują konkretne aspekty, które należy wziąć pod uwagę, oraz wpływ tych różnych podejść na te aspekty.

IBM MQ jednolite klastry, patrz sekcja Informacje o jednolitych klastrach, udostępnia potężny mechanizm umożliwiający poziome skalowanie aplikacji w wielu menedżerach kolejek opartych na podstawowym mechanizmie CCDT w celu udostępnienia wielu miejsc docelowych. Jednolite klastry mogą udostępniać możliwości wykraczające poza to, co jest możliwe, za pomocą zewnętrznego systemu równoważenia obciążenia, nieinformującego o bazowych protokołach IBM MQ , i uniknąć niektórych problemów omówionych poniżej, dlatego należy rozważyć użycie jednolitego klastra zamiast innych technik, tam gdzie ma to zastosowanie.

**Ostrzeżenie:** Należy zachować ostrożność w przypadku korzystania z aplikacji korzystających z produktu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging, w tym z jednego z adapterów zasobów IBM MQ , które łączą się z menedżerami kolejek przy użyciu technologii równoważenia obciążenia. Jeśli wystąpią problemy, należy je odtworzyć bez próby użycia funkcji równoważenia obciążenia.

Istnieje wiele kwestii, które oznaczają, że takie połączenia są w najlepszym razie problematyczne i w najgorszym wypadku całkowicie niewiarygodne:

- Należy zachować szczególną ostrożność podczas nawiązywania połączeń z dowolną aplikacją nawiązującą wiele połączeń z menedżerem kolejek przy użyciu dowolnej formy równoważenia obciążenia. Obejmuje to wszystkie aplikacje korzystające z klas IBM MQ classes for JMS/Jakarta Messaging , ponieważ generalnie tworzą wiele połączeń IBM MQ . Jeśli używany jest zewnętrzny system równoważenia obciążenia lub kod pośredniczący kodu niestandardowego, musi on zawsze kierować połączenia z tej samej instancji aplikacji do tego samego menedżera kolejek.
- Użycie zarządzania transakcjami XA lub interfejsu JTA (Java Transaction API) zależy od możliwości spójnego nawiązania połączenia z tym samym menedżerem kolejek-w praktyce jest mało prawdopodobne, aby było to praktyczne w przypadku jakiejkolwiek formy równoważenia obciążenia.
- -Jednolite zarządzanie klastrami polega na tym, że klienty mogą bez zakłóceń nawiązywać połączenia z konkretnymi menedżerami kolejek. Nie zaleca się łączenia zewnętrznego równoważenia obciążenia z użyciem klastrów jednostajnych

Funkcji jednolitego klastra produktu IBM MQ należy używać do poziomego skalowania aplikacji w wielu menedżerach kolejek, a nie w zewnętrznych technologiach równoważenia obciążenia. Informacje na temat klastrów jednostajnych, w tym sposobu tworzenia i używania klastrów jednostajnych, zawiera sekcja Konfigurowanie klastrów jednostajnychoraz kolejne tematy.

## Terminy używane w tej informacji

**CCDT-wiele QMGR**
Oznacza plik CCDT zawierający wiele kanałów połączenia klienckiego (CLNTCONN) z tą samą grupą, czyli atrybut połączenia klienta nazwy menedżera kolejek (QMNAME CLNTCONN), w którym różne pozycje CLNTCONN są tłumaczone na różne menedżery kolejek.

Różni się to od pliku CCDT zawierającego wiele pozycji CLNTCONN, które są po prostu różnymi adresami IP lub nazwami hostów dla tego samego menedżera kolejek z wieloma instancjami, co jest rozwiązaniem, które można połączyć z kodem pośredniczącym.

Jeśli zostanie wybrane podejście z wieloma menedżerami kolejek tabeli CCDT, należy wybrać, czy pozycje mają być priorytetyzowane, czy też ma być stosowane zarządzanie obciążeniem (WLM):

**Z priorytetem**
Użyj wielu pozycji uporządkowanych alfabetycznie z atrybutami CLNTWGHT (1) i AFFINITY (PREFERRED), aby zapamiętać ostatnie dobre połączenie.

**Losowo**
Użyj atrybutów CLNTWGHT (1) i AFFINITY (NONE). Można dopasować wagę WLM na inaczej skalowanych serwerach IBM MQ , dostosowując wartość CLNTWGHT

**Uwaga:** Należy unikać dużych różnic w CLNTWGHT między kanałami.

**System równoważenia obciążeń**
Oznacza urządzenie sieciowe z wirtualnym adresem IP (VIP) skonfigurowanym z monitorowaniem portów programów nasłuchujących TCP/IP wielu menedżerów kolejek systemu IBM MQ . Sposób konfiguracji VIP w urządzeniu sieciowym zależy od używanego urządzenia sieciowego.

Poniższe opcje odnoszą się tylko do aplikacji wysyłających komunikaty lub inicjujących synchroniczne przesyłanie żądań i odpowiedzi. Uwagi dotyczące aplikacji obsługujących te komunikaty i żądania, na przykład programów nasłuchujących, są całkowicie oddzielne i zostały szczegółowo omówione w sekcji "Podłączanie obiektu nasłuchiwania komunikatów do kolejki".

## Skala zmian kodu wymagana dla istniejących aplikacji, które łączą się z pojedynczym menedżerem kolejek

**Lista CONNAME, tabela CCDT z wieloma elementami QMGR i system równoważenia obciążenia**
MQCONN ("NazwaMenedżeraKolejek") do MQCONN ("*QMNAME")

Nazwa menedżera kolejek może znajdować się w konfiguracji JNDI ( Java Naming and Directory Interface) dla aplikacji Java Platform, Enterprise Edition (Java EE). W przeciwnym razie wymagana jest jednoznakowa zmiana kodu.

**Kod pośredniczący**
Zastąp istniejącą logikę połączenia JMS lub MQI kodem pośredniczącym.

## Obsługa różnych strategii WLM

**Lista CONNAME**
Tylko z określonym priorytetem.

Może to mieć negatywny wpływ na kod.

**CCDT-wiele QMGR**
Z priorytetami lub losowymi.

Jest mało prawdopodobne, aby miało to wpływ na kod.

**System równoważenia obciążeń**
Dowolne, w tym każde połączenie dla wszystkich komunikatów.

Może to mieć pozytywny wpływ na kod.

**Kod pośredniczący**
Dowolny, w tym każdy komunikat dla wszystkich komunikatów.

Może to mieć pozytywny wpływ na kod.

## Narzut wydajności, gdy podstawowy menedżer kolejek jest niedostępny

**Lista CONNAME**
Zawsze najpierw na liście.

Może to mieć negatywny wpływ na kod.

**CCDT-wiele QMGR**
Zapamiętuje ostatnie dobre połączenie.

Może to mieć pozytywny wpływ na kod.

**System równoważenia obciążeń**
Monitorowanie portów pozwala uniknąć błędnych menedżerów kolejek.

Może to mieć pozytywny wpływ na kod.

**Kod pośredniczący**
Zapamiętaj ostatnie dobre połączenie i spróbuj ponownie inteligentnie.

Może to mieć pozytywny wpływ na kod.

## Obsługa transakcji XA

**Lista CONNAME, tabela CCDT z wieloma elementami QMGR i system równoważenia obciążenia**
Menedżer transakcji musi przechowywać informacje odtwarzania, które ponownie łączą się z tym samym zasobem menedżera kolejek.

Wywołanie MQCONN, które jest tłumaczone na różne menedżery kolejek, zwykle unieważnia to wywołanie. Na przykład w środowisku Java EE pojedyncza fabryka połączeń powinna zostać rozstrzygnięta na pojedynczy menedżer kolejek podczas używania interfejsu XA.

Może to mieć negatywny wpływ na kod.

**Kod pośredniczący**
Kod pośredniczący kodu może spełniać wymagania interfejsu XA dla menedżera transakcji, na przykład wiele fabryk połączeń.

Może to mieć pozytywny wpływ na kod.

### Elastyczność administracji w ukrywaniu zmian w infrastrukturze przed aplikacjami

**Lista CONNAME**
Tylko DNS.

Może to mieć negatywny wpływ na kod.

**CCDT-wiele QMGR**
System DNS i współużytkowany system plików, współużytkowany system plików lub plik CCDT.

Jest mało prawdopodobne, aby miało to wpływ na kod.

**System równoważenia obciążeń**
Dynamiczny wirtualny adres IP (VIP).

Może to mieć pozytywny wpływ na kod.

**Kod pośredniczący**
Pozycje tabeli CCDT menedżera kolejek lub pojedynczego menedżera kolejek.

Jest mało prawdopodobne, aby miało to wpływ na kod.

### Unikanie zakłóceń związanych z planowaną konserwacją

Istnieje jeszcze jedna sytuacja, którą należy rozważyć i zaplanować, aby uniknąć zakłóceń w działaniu aplikacji, na przykład błędów i przekroczeń limitu czasu widocznych dla użytkowników końcowych, podczas planowanej konserwacji menedżera kolejek. Najlepszym sposobem uniknięcia zakłóceń jest usunięcie całej pracy z menedżera kolejek przed jej zatrzymaniem.

Rozważmy scenariusz żądania i odpowiedzi. Użytkownik chce, aby wszystkie żądania w trakcie realizacji i odpowiedzi były przetwarzane przez aplikację, ale nie chce, aby do systemu były wprowadzane żadne dodatkowe prace. Po prostu wyciszenie menedżera kolejek nie spełnia tej potrzeby, ponieważ aplikacje z kodem powrotu otrzymują kod powrotu RC2161 wyjątek MQRC_Q_MGR_QUIESCING przed odebraniem przez nie komunikatów odpowiedzi dla żądań w trakcie przetwarzania.

Można ustawić PUT (DISABLED) dla kolejek żądań używanych do wprowadzania pracy, pozostawiając kolejki odpowiedzi zarówno PUT (ENABLED), jak i GET (ENABLED). W ten sposób można monitorować głębokość kolejek żądań, transmisji i odpowiedzi. Po ustabilizowaniu wszystkich żądań w trakcie realizacji lub po upłynieniu limitu czasu można zatrzymać menedżer kolejek.

Poprawne kodowanie w żądających aplikacjach jest jednak wymagane do obsługi kolejki żądań PUT (DISABLED), co powoduje wystąpienie błędu RC2051 MQRC_PUT_INHIBITED podczas próby wysłania komunikatu.

Należy zauważyć, że wyjątek nie występuje podczas tworzenia połączenia z produktem IBM MQani podczas otwierania kolejki żądań. Wyjątek występuje tylko wtedy, gdy podejmowana jest próba wysłania komunikatu przy użyciu wywołania MQPUT.

Budowanie kodu pośredniczącego kodu, który zawiera logikę obsługi błędów dla scenariuszy żądań i odpowiedzi, oraz zwracanie się do zespołów aplikacji o użycie takiego kodu pośredniczącego w przyszłości, może pomóc w tworzeniu aplikacji o spójnym zachowaniu.

## V 9.3.0 Tworzenie elastycznych i skalowalnych aplikacji klienckich

Ze względu na odporność na błędy i skalowalność wdrażanie aplikacji klienckich, które obsługują opcje połączeń w jednolitych klastrach, umożliwia ponowne zrównoważenie instancji aplikacji między menedżerami kolejek.

Przegląd klastrów jednostajnych znajduje się w sekcji Informacje o klastrach jednostajnych .

W idealnym przypadku to ponowne równoważenie jest niewidoczne dla aplikacji, ale tylko niektóre typy aplikacji są odpowiednie dla tego typu wdrożenia i może być konieczne pewne uwzględnienie w projekcie aplikacji.

Rozważania te dzielą się na dwie główne kategorie:

- Rzadkie *ścieżki błędów* , które mogą już istnieć dla aplikacji z możliwością ponownego połączenia, ale stają się bardziej prawdopodobne po wdrożeniu w jednolitym klastrze. Na przykład po ponownym nawiązaniu połączenia wszystkie jednostki pracy w trakcie pracy są wycofane, a kursory przeglądania są resetowane. Może to być rzadkie zdarzenie dla aplikacji, z którą można ponownie nawiązać połączenie w bieżącym środowisku i dlatego nie jest ona obsługiwana w możliwie najczystszym zakresie przez kod aplikacji. Przeglądanie logiki aplikacji w celu zapewnienia odpowiedniej obsługi w takich sytuacjach pomaga uniknąć nieoczekiwanych problemów.
- *Powinowactwa* do konkretnego menedżera kolejek. Jeśli wiadomo, że aplikacja musi zawsze nawiązywać połączenie z tym samym lub konkretnym menedżerem kolejek, aplikacja powinna być skonfigurowana do ponownego nawiązywania połączenia z tym menedżerem kolejek lub nie mieć włączonego połączenia z tym menedżerem kolejek. Jednak te powinowactwa mogą być tymczasowe, takie jak oczekiwanie na komunikat odpowiedzi. Wpływ algorytmu równoważenia na uwzględnienie tych powinowactw z kodu aplikacji został omówiony w poniższej sekcji. Więcej szczegółowych informacji na temat tych opcji oraz sposobu osiągnięcia podobnego podejścia dzięki konfiguracji, a nie kodowi aplikacji, zawiera sekcja Wpływanie na ponowne równoważenie aplikacji w jednolitych klastrach.

## Wpływanie na opcje ponownego połączenia w MQI

Więcej informacji na temat komendy MQCNO_RECONNECT zawiera sekcja Opcje ponownego połączenia .

Jeśli wiadomo, że aplikacja musi zawsze łączyć się z tym samym lub konkretnym menedżerem kolejek, powinna być skonfigurowana jako MQCNO_RECONNECT_Q_MGR lub MQCNO_RECONNECT_DISABLED.

## Wpływanie na algorytm równoważenia obciążenia w MQI

Użytkownik może jednak chcieć kontrolować lub wpływać na to zachowanie związane z równoważeniem, aby dostosować je do potrzeb konkretnych typów aplikacji, na przykład zminimalizować przerwy w realizacji transakcji lotniczych lub zapewnić, że aplikacje requestera otrzymają odpowiedzi przed przeniesieniem.

Niektóre domyślne pożądane zachowania zostały przyjęte i omówione w sekcji Wpływanie na ponowne równoważenie aplikacji w jednolitych klastrach. Można również wpłynąć na zachowanie konkretnych aplikacji w czasie konfiguracji lub wdrażania za pomocą pliku client.ini , co zostało omówione w tym temacie.

W innych sytuacjach bardziej sensowne może być zrównoważenie zachowania i wymagań w logice aplikacji. W takich przypadkach te same odpowiednie parametry aplikacji mogą zostać dostarczone do programu IBM MQ podczas nawiązywania połączenia z menedżerem kolejek w wywołaniu MQCONNX w strukturze o nazwie MQBNO (opcje równoważenia).

Jeśli zostanie określona struktura MQBNO, musi ona zawierać wszystkie informacje wymagane przez produkt IBM MQ do podjęcia decyzji o tym, kiedy i w jaki sposób aplikacja powinna zostać poproszona o ponowne nawiązanie połączenia z innym menedżerem kolejek.

Należy podać:

- **Type** aplikacji
- **Timeout** , w którym instancja jest równoważona niezależnie od stanu
- Dowolne specjalne **BalanceOptions**

Wyjątkiem jest sytuacja, w której limit czasu można pozostawić jako MQBNO_TIMEOUT_DEFAULT, jeśli jest to wymagane. W takim przypadku limit czasu jest tłumaczony na dowolną wartość w pliku client.ini , aplikacji lub sekcjach globalnych, jeśli zostanie podany, i w przeciwnym razie na podstawową wartość domyślną wynoszącą 10 sekund.

Szczegółowe informacje na temat formatu tej struktury zawiera sekcja MQBNO .

W przypadku aplikacji .NET należy zapoznać się z sekcją Wpływanie na ponowne równoważenie aplikacji w środowisku .NET , aby uzyskać więcej informacji.

## Aplikacje obiektowe

IBM MQ zapewnia obsługę systemów JMS, Java, C + + i .NET. Te języki i środowiska korzystają z modelu obiektowego IBM MQ , który udostępnia klasy udostępniające te same funkcje, co wywołania i struktury IBM MQ .

Niektóre języki i środowiska używające modelu obiektów IBM MQ udostępniają dodatkowe funkcje, które nie są dostępne w przypadku używania języków proceduralnych z interfejsem kolejki komunikatów (MQI).

Szczegółowe informacje na temat klas, metod i właściwości udostępnianych przez ten model zawiera sekcja “Model obiektu IBM MQ” na stronie 17.

**JMS**

IBM MQ udostępnia klasy, które implementują specyfikacje Jakarta Messaging 3.0 i Java Message Service 2.0 . Szczegółowe informacje na temat IBM MQ classes for JMSzawiera sekcja Korzystanie z programu IBM MQ classes for JMS. Aby uzyskać informacje na temat różnic między produktami IBM MQ classes for Java i IBM MQ classes for JMS, należy zapoznać się z sekcją “Tworzenie aplikacji JMS/Jakarta Messaging i Java” na stronie 83.

IBM MQ Message Service Client (XMS) for C/C++ i IBM MQ Message Service Client (XMS) for .NET udostępniają aplikacyjny interfejs programistyczny (API) o nazwie XMS , który ma ten sam zestaw interfejsów co Java Message Service (JMS) Interfejs API. Więcej informacji na ten temat zawiera sekcja “Tworzenie aplikacji XMS .NET” na stronie 635.

**Java**

Informacje na temat kodowania programów przy użyciu modelu obiektowego IBM MQ w systemie Javazawiera sekcja Korzystanie z programu IBM MQ classes for Java .

Produkt IBM nie zawiera żadnych dodatkowych udoskonaleń w produkcie IBM MQ classes for Java i są one funkcjonalnie ustabilizowane na poziomie dostarczanym wraz z produktem IBM MQ 8.0. Informacje na temat różnic między IBM MQ classes for Java i IBM MQ classes for JMS , które ułatwiają podjęcie decyzji, które z nich mają być używane, zawiera sekcja “Tworzenie aplikacji JMS/Jakarta Messaging i Java” na stronie 83.

**C++**

IBM MQ udostępnia klasy C++ równoważne obiektom IBM MQ i pewne dodatkowe klasy równoważne tablicom typów danych. Udostępnia on wiele funkcji, które nie są dostępne za pośrednictwem interfejsu MQI. Korzystanie z języka C++ zawiera informacje o programowaniu programów korzystających z modelu obiektów IBM MQ w języku C + +. Klienty usługi komunikatów dla środowisk C/C++ i .NET udostępniają aplikacyjny interfejs programistyczny (API) o nazwie XMS , który ma taki sam zestaw interfejsów jak Java Message Service (JMS) Interfejs API.

**.NET**

Informacje na temat kodowania programów .NET za pomocą klas IBM MQ .NET zawiera sekcja Tworzenie aplikacji .NET . Klienty usługi komunikatów dla środowisk C/C++ i .NET udostępniają interfejs API o nazwie XMS , który ma taki sam zestaw interfejsów jak Java Message Service (JMS) Interfejs API.

**Pojęcia pokrewne**

“Tworzenie aplikacji MQI przy użyciu produktu IBM MQ” na stronie 740
IBM MQ zapewnia obsługę języków C, Visual Basic, COBOL, Assembler, RPG, pTALi PL/I. Te języki proceduralne korzystają z interfejsu kolejki komunikatów (MQI) w celu uzyskania dostępu do usług kolejkowania komunikatów.

Przegląd techniczny

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć języka proceduralnego lub obiektowego. Przed rozpoczęciem projektowania i pisania aplikacji IBM MQ należy zapoznać się z podstawowymi pojęciami związanymi z produktem IBM MQ .

**Odsyłacze pokrewne**

Informacje dodatkowe o tworzeniu aplikacji

## Model obiektu IBM MQ

Model obiektu IBM MQ składa się z klas, metod i właściwości.

Model obiektowy IBM MQ składa się z następujących elementów:

- *Klasy* reprezentujące znane pojęcia związane z produktem IBM MQ , takie jak menedżery kolejek, kolejki i komunikaty.
- *Metody* dla każdej klasy odpowiadającej wywołaniom MQI.
- *Właściwości* dla każdej klasy odpowiadającej atrybutom obiektów IBM MQ .

Podczas tworzenia aplikacji IBM MQ za pomocą modelu obiektów IBM MQ należy utworzyć instancje tych klas w aplikacji. Instancja klasy w programowaniu obiektowym jest nazywana *obiektem*. Po utworzeniu obiektu można wykonywać z nim interakcję, sprawdzając lub ustawiając wartości właściwości obiektu (odpowiednik wywołania MQINQ lub MQSET) oraz wywołując metodę względem obiektu (odpowiednik innych wywołań MQI).

### Klasy

Model obiektów IBM MQ udostępnia następujący podstawowy zestaw klas.

Rzeczywista implementacja modelu różni się nieznacznie w zależności od różnych obsługiwanych środowisk obiektowych.

**MQQueueManager**
Obiekt klasy MQQueueManager reprezentuje połączenie z menedżerem kolejek. Zawiera on metody do metod Connect (), Disconnect (), Commit () i Backout () (odpowiednik MQCONN lub MQCONNX, MQDISC, MQCMIT i MQBACK). Ma on właściwości odpowiadające atrybutom menedżera kolejek. Uzyskanie dostępu do właściwości atrybutu menedżera kolejek powoduje niejawne nawiązanie połączenia z menedżerem kolejek, jeśli nie jest on jeszcze połączony. Zniszczenie obiektu MQQueueManager powoduje niejawne rozłączenie z menedżerem kolejek.

**MQQUEUE**
Obiekt klasy MQQueue reprezentuje kolejkę. Zawiera metody do umieszczania () i pobierania () komunikatów do i z kolejki (odpowiednik MQPUT i MQGET). Ma właściwości odpowiadające atrybutom kolejki. Uzyskanie dostępu do właściwości atrybutu kolejki lub wywołanie metody Put () lub Get () powoduje niejawne otwarcie kolejki (odpowiednik wywołania MQOPEN). Zniszczenie obiektu MQQueue powoduje niejawne zamknięcie kolejki (odpowiednik MQCLOSE).

**Temat MQ**
Obiekt klasy MQTopic reprezentuje temat. Zawiera metody do umieszczania () (publish) i pobierania (Get) (receive lub subscribe) komunikatów do i z tematu (odpowiednik MQPUT i MQGET). Ma właściwości odpowiadające atrybutom tematu. Dostęp do obiektu MQTopic można uzyskać tylko w celu publikowania lub subskrybowania, a nie jednocześnie. W przypadku odbierania komunikatów obiekt MQTopic może zostać utworzony z niezarządzaną lub zarządzaną subskrypcją i jako trwały lub nietrły subskrybent-dla tych różnych scenariuszy udostępniono wiele przeciążonych konstruktorów.

**Komunikat MQ**
Obiekt klasy MQMessage reprezentuje komunikat, który ma zostać umieszczony w kolejce lub odebrany z kolejki. Zawiera bufor i obudowuje zarówno dane aplikacji, jak i strukturę MQMD. Ma właściwości odpowiadające polom MQMD i metodom, które umożliwiają zapisywanie i odczytywanie danych użytkownika różnych typów (na przykład łańcuchów, długich liczb całkowitych, krótkich liczb całkowitych, pojedynczych bajtów) do i z buforu.

**Opcje MQPutMessage**
Obiekt klasy opcji MQPutMessagereprezentuje strukturę MQPMO. Ma on właściwości odpowiadające polu MQPMO.

**Opcje MQGetMessage**
Obiekt klasy opcji MQGetMessagereprezentuje strukturę MQGMO. Ma on właściwości odpowiadające polu MQGMO.

**Proces MQProcess**
Obiekt klasy MQProcess reprezentuje definicję procesu (używany z wyzwalaniem). Ma właściwości, które reprezentują atrybuty definicji procesu.

Multi **MQDistributionList**
Obiekt klasy MQDistributionList reprezentuje listę dystrybucyjną (używaną do wysyłania wielu komunikatów z pojedynczym MQPUT). Zawiera on listę obiektów elementów MQDistributionList.

Multi **Element MQDistributionList**
Obiekt klasy elementu MQDistributionListreprezentuje pojedyncze miejsce docelowe listy dystrybucyjnej. Zawiera on struktury MQOR, MQRR i MQPMR oraz właściwości odpowiadające polom tych struktur.

### odwołania do obiektów

W programie IBM MQ korzystającym z interfejsu MQI produkt IBM MQ zwraca do programu uchwyty połączeń i uchwyty obiektów.

Te uchwyty muszą być przekazywane jako parametry w kolejnych wywołaniach IBM MQ . W modelu obiektowym IBM MQ te uchwyty są ukryte przed aplikacją. Zamiast tego utworzenie obiektu z klasy powoduje zwrócenie odwołania do obiektu do aplikacji. Jest to odwołanie do obiektu, które jest używane podczas wywoływania metod i uzyskiwania dostępu do właściwości dla obiektu.

### Kody powrotu

Wywołanie metody lub ustawienie wartości właściwości powoduje ustawienie kodów powrotu.

Te kody powrotu są kodem zakończenia i kodem przyczyny i same są właściwościami obiektu. Wartości kodu zakończenia i kodu przyczyny są takie same, jak zdefiniowane dla interfejsu MQI, z pewnymi dodatkowymi wartościami specyficznymi dla środowiska obiektowego.

## Komunikaty produktu IBM MQ

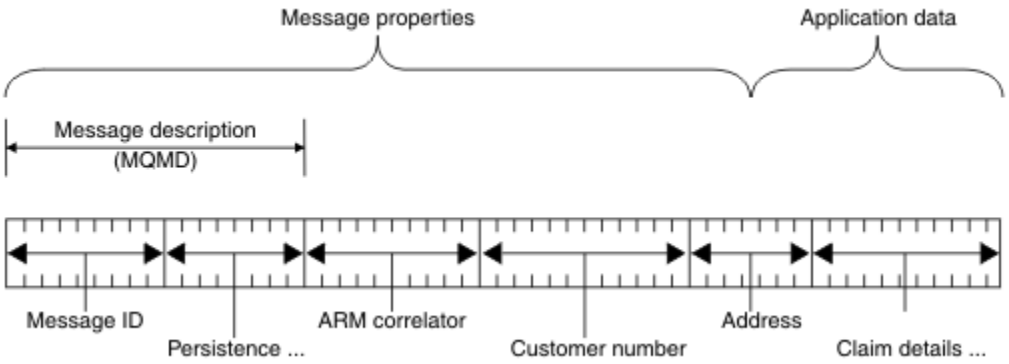
Komunikat IBM MQ składa się z właściwości komunikatu i danych aplikacji. Deskryptor komunikatu kolejkowania komunikatów (MQMD) zawiera informacje sterujące, które towarzyszą danych aplikacji, gdy komunikat jest przesyłany między aplikacjami wysyłającymi i odbierającymi.

### Części komunikatu

Komunikaty IBM MQ składają się z dwóch części:

- Właściwości komunikatu
- Dane aplikacji

Plik Rysunek 2 na stronie 19 reprezentuje komunikat i pokazuje, w jaki sposób jest on logicznie podzielony na właściwości komunikatu i dane aplikacji.




*Rysunek 2. Reprezentacja komunikatu*

Dane aplikacji przenoszone w komunikacie IBM MQ nie są zmieniane przez menedżer kolejek, chyba że przeprowadzana jest na nich konwersja danych. Ponadto program IBM MQ nie nakłada żadnych ograniczeń na treść tych danych. Długość danych w każdym komunikacie nie może przekraczać wartości atrybutu **MaxMsgLength** zarówno dla kolejki, jak i dla menedżera kolejek.

ALW W systemie AIX, Linux, and Windowswartość domyślna atrybutu *MaxMsgLength* menedżera kolejek i kolejki wynosi 4 MB (4 194 304 bajty), a w razie potrzeby można ją zmienić maksymalnie do 100 MB (104 857 600 bajtów).

IBM i W systemie IBM iwartość domyślna atrybutu *MaxMsgLength* menedżera kolejek i kolejki wynosi 4 MB (4 194 304 bajty), a w razie potrzeby można ją zmienić maksymalnie do 100 MB (104 857 600 bajtów). Jeśli w systemie IBM imają być używane komunikaty IBM MQ większe niż 15 MB, należy zapoznać się z sekcją “Budowanie aplikacji proceduralnej w systemie IBM i” na stronie 1036.

z/OS W systemie z/OSatrybut **MaxMsgLength** menedżera kolejek ma wartość 100 MB, a atrybut **MaxMsgLength** kolejki ma wartość domyślną 4 MB (4 194 304 bajtów), którą można zmienić do maksymalnie 100 MB, jeśli jest to wymagane.

W pewnych okolicznościach komunikaty mogą być nieco krótsze niż wartość atrybutu **MaxMsgLength** . Więcej informacji na ten temat zawiera sekcja “Dane w komunikacie” na stronie 782.

Komunikat jest tworzony podczas używania wywołań MQI MQPUT lub MQPUT1 . Jako dane wejściowe dla tych wywołań należy podać informacje sterujące (takie jak priorytet komunikatu i nazwa kolejki odpowiedzi) oraz dane, a następnie wywołanie umieszcza komunikat w kolejce. Więcej informacji na temat tych wywołań zawierają MQPUT i MQPUT1 .

### deskryptor komunikatu

Dostęp do informacji o sterowaniu komunikatami można uzyskać za pomocą struktury MQMD, która definiuje *deskryptor komunikatu*.

Pełny opis struktury MQMD zawiera sekcja MQMD-deskryptor komunikatu.

Opis sposobu użycia pól w strukturze MQMD, które zawierają informacje o pochodzeniu komunikatu, zawiera sekcja “Kontekst komunikatu” na stronie 48.

Istnieją różne wersje deskryptora komunikatu. Dodatkowe informacje dotyczące grupowania i segmentowania komunikatów (patrz sekcja “Grupy komunikatów” na stronie 45) są dostępne w wersji 2 deskryptora komunikatu (lub w MQMDE). Jest to ten sam deskryptor, co deskryptor komunikatu wersji 1, ale zawiera dodatkowe pola. Te pola są opisane w sekcji MQMDE-Rozszerzenie deskryptora komunikatu.

## Typy komunikatów

Istnieją cztery typy komunikatów definiowane przez produkt IBM MQ.

Są to następujące cztery komunikaty:

- Datagram
- Komunikaty żądań
- Komunikaty odpowiedzi
- Komunikaty raportu
  - Typy komunikatów raportu
  - Opcje komunikatów raportu

Aplikacje mogą używać pierwszych trzech typów komunikatów do przekazywania informacji między sobą. Czwarty typ, raport, jest przeznaczony dla aplikacji i menedżerów kolejek do raportowania informacji o zdarzeniach, takich jak wystąpienie błędu.

Każdy typ komunikatu jest identyfikowany przez wartość MQMT_ *. Można również zdefiniować własne typy komunikatów. Zakres wartości, których można użyć, zawiera sekcja MsgType.

## Datagramy

Należy użyć *datagramu* , jeśli nie jest wymagana odpowiedź od aplikacji, która odbiera komunikat (to znaczy pobiera komunikat z kolejki).

Przykładem aplikacji, która może używać datagramów, jest aplikacja, która wyświetla informacje o locie w poczekalni lotniska. Komunikat może zawierać dane dla całego ekranu z informacjami o locie. Jest mało prawdopodobne, aby taka aplikacja zażądała potwierdzenia dla komunikatu, ponieważ prawdopodobnie nie ma znaczenia, czy komunikat nie został dostarczony. Aplikacja wysyła komunikat aktualizacji po krótkim czasie.

## Komunikaty żądań

Użyj *komunikatu żądania* , aby otrzymać odpowiedź od aplikacji, która odbiera komunikat.

Przykładem aplikacji, która może korzystać z komunikatów żądań, jest aplikacja, która wyświetla saldo konta rozliczeniowego. Komunikat żądania może zawierać numer konta, a komunikat odpowiedzi może zawierać saldo konta.

Aby połączyć komunikat odpowiedzi z komunikatem żądania, dostępne są dwie opcje:

- Ustaw aplikację obsługującą komunikat żądania jako odpowiedzialną za umieszczanie informacji w komunikacie odpowiedzi, który odnosi się do komunikatu żądania.
- Użyj pola raportu w deskryptorze komunikatu żądania, aby określić treść pól *MsgId* i *CorrelId* komunikatu odpowiedzi:
  - Można zażądać skopiowania pola *MsgId* lub *CorrelId* oryginalnego komunikatu do pola *CorrelId* komunikatu odpowiedzi (domyślnym działaniem jest skopiowanie pola *MsgId*).
  - Można zażądać wygenerowania nowej wartości pola *MsgId* dla komunikatu odpowiedzi lub skopiowania wartości pola *MsgId* oryginalnego komunikatu do pola *MsgId* komunikatu odpowiedzi (domyślnym działaniem jest wygenerowanie nowego identyfikatora komunikatu).

## Komunikaty odpowiedzi

Podczas odpowiadania na inny komunikat należy użyć *komunikatu odpowiedzi* .

Podczas tworzenia komunikatu odpowiedzi należy przestrzegać wszystkich opcji, które zostały ustawione w deskryptorze komunikatu, na który jest wysyłana odpowiedź. Opcje raportu określają zawartość pól identyfikatora komunikatu (*MsgId*) i identyfikatora korelacji (*CorrelId*). Te pola umożliwiają aplikacji, która otrzymuje odpowiedź, skorelowanie odpowiedzi z pierwotnym żądaniem.

## Komunikaty raportu

*Komunikaty raportów* informują aplikacje o zdarzeniach, takich jak wystąpienie błędu podczas przetwarzania komunikatu.

Mogą być generowane przez:

- Menedżer kolejek,
- Agent kanału komunikatów (na przykład, jeśli nie może dostarczyć komunikatu) lub
- Aplikacja (na przykład, jeśli nie może użyć danych w komunikacie).

Komunikaty raportu mogą być generowane w dowolnym momencie i mogą pojawiać się w kolejce, gdy aplikacja ich nie oczekuje.

### Typy komunikatów raportu

Po umieszczeniu komunikatu w kolejce można wybrać opcję odbierania:

- *Komunikat raportu o wyjątku*. Jest on wysyłany w odpowiedzi na komunikat z ustawioną flagą wyjątków. Jest on generowany przez agent kanału komunikatów (MCA) lub aplikację.
- *Komunikat raportu utraty ważności*. Oznacza to, że aplikacja podjęła próbę pobrania komunikatu, który osiągnął próg utraty ważności. Komunikat jest oznaczany do usunięcia. Ten typ raportu jest generowany przez menedżer kolejek.
- *Komunikat raportu potwierdzenia przybycia (COA)*. Oznacza to, że komunikat osiągnął swoją kolejkę docelową. Jest on generowany przez menedżer kolejek.
- *Komunikat raportu potwierdzenia dostarczenia (COD)*. Oznacza to, że komunikat został pobrany przez aplikację odbierającą. Jest on generowany przez menedżer kolejek.
- *Komunikat raportu powiadomienia o działaniu pozytywnym (PAN)*. Oznacza to, że żądanie zostało pomyślnie obsłużone (oznacza to, że działanie żądane w komunikacie zostało wykonane pomyślnie). Ten typ raportu jest generowany przez aplikację.
- *Komunikat raportu powiadomienia o negatywnym działaniu (NAN)*. Oznacza to, że żądanie nie zostało pomyślnie obsłużone (co oznacza, że działanie żądane w komunikacie nie zostało wykonane pomyślnie). Ten typ raportu jest generowany przez aplikację.

**Uwaga:** Każdy typ komunikatu raportu zawiera jeden z następujących elementów:

- Cała oryginalna wiadomość
- Pierwsze 100 bajtów danych w oryginalnym komunikacie
- Brak danych z oryginalnego komunikatu

Podczas umieszczania komunikatu w kolejce można zażądać więcej niż jednego typu komunikatu raportu. Jeśli zostanie wybrany komunikat z potwierdzeniem dostarczenia i opcje komunikatu raportu o wyjątku, jeśli dostarczenie komunikatu nie powiedzie się, zostanie wyświetlony komunikat z raportem o wyjątku. Jeśli jednak zostanie wybrana tylko opcja komunikatu raportu potwierdzenia dostarczenia, a dostarczenie komunikatu nie powiedzie się, nie zostanie wyświetlony komunikat raportu o wyjątku.

Żądane komunikaty raportu, gdy spełnione są kryteria generowania konkretnego komunikatu, są jedynymi komunikatami, które użytkownik otrzymuje.

### Opcje komunikatów raportu

Komunikat można *odrzucić* po wystąpieniu wyjątku. Jeśli wybrano opcję usuwania i zażądano komunikatu raportu o wyjątku, komunikat raportu jest kierowany do *ReplyToQ* i *ReplyToQMgr*, a oryginalny komunikat jest odrzucany.

**Uwaga:** Korzyścią z tego jest możliwość zmniejszenia liczby komunikatów przesyłanych do kolejki niedostarczonych komunikatów. Oznacza to jednak, że aplikacja, o ile nie wysyła tylko komunikatów datagramu, musi zajmować się zwróconymi komunikatami. Po wygenerowaniu komunikatu raportu o wyjątku dziedziczy on trwałość oryginalnego komunikatu.

Jeśli nie można dostarczyć komunikatu raportu (jeśli na przykład kolejka jest pełna), komunikat raportu jest umieszczany w kolejce niedostarczonych komunikatów.

Jeśli ma zostać odebrany komunikat raportu, należy określić nazwę kolejki odpowiedzi w polu *ReplyToQ* . W przeciwnym razie wywołanie MQPUT lub MQPUT1 oryginalnego komunikatu nie powiedzie się i zostanie wyświetlony komunikat MQRC_MISSING_REPLY_TO_Q.

Można użyć innych opcji raportu w deskryptorze komunikatu (MQMD), aby określić treść pól *MsgId* i *CorrelId* dowolnych komunikatów raportu, które są tworzone dla komunikatu:

- Można zażądać, aby plik *MsgId* lub plik *CorrelId* oryginalnego komunikatu został skopiowany do pola *CorrelId* komunikatu raportu. Domyślnym działaniem jest skopiowanie identyfikatora komunikatu. Użyj parametru MQRO_COPY_MSG_ID_TO_CORRELID, ponieważ umożliwia on nadawcy komunikatu korelację komunikatu odpowiedzi lub komunikatu raportu z komunikatem oryginalnym. Identyfikator korelacji komunikatu odpowiedzi lub komunikatu raportu jest identyczny z identyfikatorem oryginalnego komunikatu.
- Można zażądać wygenerowania nowego elementu *MsgId* dla komunikatu raportu lub skopiowania elementu *MsgId* oryginalnego komunikatu do pola *MsgId* komunikatu raportu. Działanie domyślne polega na wygenerowaniu nowego identyfikatora komunikatu. Użyj MQRO_NEW_MSG_ID, ponieważ zapewnia, że każdy komunikat w systemie ma inny identyfikator komunikatu i można go jednoznacznie odróżnić od wszystkich innych komunikatów w systemie.
- Wyspecjalizowane aplikacje mogą wymagać użycia identyfikatora MQRO_PASS_MSG_ID lub MQRO_PASS_CORREL_ID. Należy jednak zaprojektować aplikację, która odczytuje komunikaty z kolejki, aby upewnić się, że działa ona poprawnie, jeśli na przykład kolejka zawiera wiele komunikatów o takim samym identyfikatorze.

  Aplikacje serwera muszą sprawdzić ustawienia tych flag w komunikacie żądania oraz odpowiednio ustawić pola *MsgId* i *CorrelId* w komunikacie odpowiedzi lub w komunikacie raportu.

  Aplikacje, które działają jako pośrednicy między aplikacją requestera a aplikacją serwera, nie muszą sprawdzać ustawień tych flag. Jest to spowodowane tym, że te aplikacje zwykle muszą przekazać komunikat do aplikacji serwera z niezmienionymi polami *MsgId*, *CorrelId*i *Report* . Dzięki temu aplikacja serwera może skopiować plik *MsgId* z oryginalnego komunikatu w polu *CorrelId* komunikatu odpowiedzi.

Podczas generowania raportu dotyczącego komunikatu aplikacje serwera muszą sprawdzić, czy którakolwiek z tych opcji została ustawiona.

Więcej informacji na temat korzystania z komunikatów raportu zawiera sekcja Raport.

Aby wskazać rodzaj raportu, menedżery kolejek używają różnych kodów sprzężenia zwrotnego. Kody te są umieszczane w polu *Feedback* deskryptora komunikatu raportu. Menedżery kolejek mogą również zwracać kody przyczyny MQI w polu *Feedback* . IBM MQ definiuje zakres kodów sprzężenia zwrotnego używanych przez aplikacje.

Więcej informacji na temat informacji zwrotnych i kodów przyczyny zawiera sekcja Informacje zwrotne.

Przykładem programu, który może korzystać z kodu sprzężenia zwrotnego, jest program monitorujący obciążenia innych programów obsługujących kolejkę. Jeśli istnieje więcej niż jedna instancja programu obsługującego kolejkę, a liczba komunikatów przychodzących do kolejki nie jest już uzasadniona, taki program może wysłać komunikat raportu (z kodem informacji zwrotnej MQFB_QUIT) do jednego z programów obsługujących, aby wskazać, że program powinien zakończyć swoje działanie. (Program monitorujący może użyć wywołania MQINQ, aby dowiedzieć się, ile programów obsługuje kolejkę).

### Multi *Raporty i segmentowane komunikaty*

Nieobsługiwane w systemie IBM MQ for z/OS.

Jeśli komunikat jest segmentowany i użytkownik prosi o wygenerowanie raportów, może zostać wygenerowanych więcej raportów, niż byłoby to możliwe, gdyby komunikat nie był segmentowany.

Opis komunikatów segmentowanych zawiera sekcja “Segmentacja komunikatów” na stronie 817.

## Dla raportów wygenerowanych przez program IBM MQ

W przypadku segmentowania komunikatów lub zezwolenia na to menedżerowi kolejek istnieje tylko jeden przypadek, w którym można oczekiwać otrzymania pojedynczego raportu dla całego komunikatu. Jest to sytuacja, w której zażądano tylko raportów COD i określono opcję MQGMO_COMPLETE_MSG w aplikacji pobierającej.

W innych przypadkach aplikacja musi być przygotowana do obsługi kilku raportów, zwykle po jednym dla każdego segmentu.

**Uwaga:** Jeśli komunikaty są segmentowane, a wymagane jest zwrócenie tylko pierwszych 100 bajtów oryginalnych danych komunikatu, należy zmienić ustawienie opcji raportu, aby żądać raportów bez danych dla segmentów, które mają przesunięcie 100 lub więcej. Jeśli ta czynność nie zostanie wykonana, a użytkownik pozostawi to ustawienie, tak aby każdy segment żądał 100 bajtów danych, i pobierze komunikaty raportu z pojedynczym MQGET z parametrem MQGMO_COMPLETE_MSG, raporty będą składane w duży komunikat zawierający 100 bajtów odczytanych danych z każdym odpowiednim przesunięciem. W takim przypadku potrzebny jest duży bufor lub należy określić parametr MQGMO_ACCEPT_TRUNCATED_MSG.

## Dla raportów wygenerowanych przez aplikacje

Jeśli aplikacja generuje raporty, należy zawsze kopiować nagłówki IBM MQ , które są obecne na początku oryginalnych danych komunikatu, do danych komunikatu raportu.

Następnie dodaj do raportu danych komunikatu wszystkie dane oryginalnego komunikatu (lub inną ilość, którą zwykle uwzględniasz), albo nie dodawaj żadnych, 100 bajtów lub wszystkich danych oryginalnego komunikatu.

Nagłówki IBM MQ , które muszą zostać skopiowane, można rozpoznać, sprawdzając kolejne nazwy formatów, rozpoczynając od deskryptora MQMD i przechodząc przez wszystkie obecne nagłówki. Następujące nazwy `Format` wskazują te nagłówki IBM MQ :

- MQMDE,
- MQDLH
- MQXQH
- MQIIH.
- MQH*

MQH* oznacza dowolną nazwę rozpoczynającą się od znaków MQH.

Nazwa `Format` występuje w określonych pozycjach dla MQDLH i MQXQH, ale dla innych nagłówków IBM MQ występuje w tej samej pozycji. Długość nagłówka jest zawarta w polu, które występuje również w tej samej pozycji dla MQMDE, MQIMSi wszystkich nagłówków MQH*.

Jeśli używana jest struktura MQMD w wersji 1 i tworzone są raporty dotyczące segmentu, komunikatu w grupie lub komunikatu, dla którego dozwolona jest segmentacja, dane raportu muszą zaczynać się od MQMDE. W polu *`OriginalLength`* ustaw długość oryginalnych danych komunikatu, z wyłączeniem długości wszystkich nagłówków IBM MQ , które można znaleźć.

## Pobieranie raportów

W przypadku pytania o raporty COA lub COD można poprosić o ich ponowne zmontowanie za pomocą komendy MQGMO_COMPLETE_MSG.

Komenda MQGET z opcją MQGMO_COMPLETE_MSG jest spełniona, gdy w kolejce znajduje się wystarczająca liczba komunikatów raportu (pojedynczego typu, na przykład COA i z tym samym *`GroupId`*), które reprezentują jeden kompletny komunikat oryginalny. Dzieje się tak nawet wtedy, gdy same komunikaty raportu nie zawierają pełnych oryginalnych danych; pole *`OriginalLength`* w każdym komunikacie raportu podaje długość oryginalnych danych reprezentowanych przez ten komunikat raportu, nawet jeśli same dane nie są dostępne.

Tej techniki można użyć nawet wtedy, gdy w kolejce istnieje kilka różnych typów raportów (na przykład zarówno COA, jak i COD), ponieważ komenda MQGET z opcją MQGMO_COMPLETE_MSG składa komunikaty raportów tylko wtedy, gdy mają ten sam kod *Feedback* . Jednak zwykle nie można używać tej techniki dla raportów o wyjątkach, ponieważ generalnie mają one różne kody *Feedback* .

Tej techniki można użyć do uzyskania pozytywnego wskazania, że cały komunikat został odebrany. Jednak w większości przypadków konieczne jest uwzględnienie możliwości, że niektóre segmenty nadchodzą, podczas gdy inne mogą generować wyjątek (lub utratę ważności, jeśli jest to dozwolone). W tym przypadku nie można użyć opcji MQGMO_COMPLETE_MSG, ponieważ zwykle można uzyskać różne kody *Feedback* dla różnych segmentów i uzyskać więcej niż jeden raport dla segmentu. Można jednak użyć opcji MQGMO_ALL_SEGMENTS_AVAILABLE.

Aby to umożliwić, może być konieczne pobranie raportów po ich nadejściu i zbudowanie obrazu w aplikacji tego, co stało się z oryginalnym komunikatem. Można użyć pola *GroupId* w komunikacie raportu, aby skorelować raporty z *GroupId* oryginalnego komunikatu oraz pola *Feedback* , aby zidentyfikować typ każdego komunikatu raportu. Sposób wykonania tej czynności zależy od wymagań aplikacji.

Jedno z nich jest następujące:

- Poproś o raporty COD i raporty o wyjątkach.
- Po określonym czasie sprawdź, czy za pomocą komendy MQGMO_COMPLETE_MSG odebrano pełny zestaw raportów COD. Jeśli tak, aplikacja wie, że cały komunikat został przetworzony.
- Jeśli nie, a raporty wyjątków dotyczące tego komunikatu są dostępne, należy obsłużyć problem, tak jak w przypadku komunikatów niesegmentowanych, ale w pewnym momencie należy wyczyścić segmenty osierocone.
- Jeśli istnieją segmenty, dla których nie ma żadnych raportów, oryginalne segmenty (lub raporty) mogą oczekiwać na ponowne połączenie kanału lub sieć może być w pewnym momencie przeciążona. Jeśli w ogóle nie otrzymano żadnych raportów o wyjątkach (lub jeśli uważasz, że te raporty mogą być tylko tymczasowe), możesz zdecydować, aby aplikacja poczekała trochę dłużej.

  Podobnie jak wcześniej, jest to podobne do rozważań dotyczących komunikatów niesegmentowanych, z tą różnicą, że należy również rozważyć możliwość czyszczenia segmentów osieroconych.

Jeśli oryginalny komunikat nie jest krytyczny (na przykład jeśli jest to zapytanie lub komunikat, który można powtórzyć później), należy ustawić czas utraty ważności, aby zapewnić usunięcie segmentów osieroconych.

### Menedżery kolejek o wcześniejszej wersji

Jeśli raport jest generowany przez menedżer kolejek, który obsługuje segmentację, ale jest odbierany w menedżerze kolejek, który nie obsługuje segmentacji, struktura MQMDE (identyfikująca elementy *Offset* i *OriginalLength* reprezentowane przez raport) jest zawsze uwzględniana w danych raportu oprócz zera, 100 bajtów lub wszystkich oryginalnych danych w komunikacie.

Jeśli jednak segment komunikatu przechodzi przez menedżer kolejek, który nie obsługuje segmentacji, to w przypadku wygenerowania w nim raportu struktura MQMDE w oryginalnym komunikacie jest traktowana wyłącznie jako dane. Dlatego nie jest on uwzględniany w danych raportu, jeśli zażądano zerowych bajtów oryginalnych danych. Bez MQMDE komunikat raportu może nie być użyteczny.

Zażądaj co najmniej 100 bajtów danych w raportach, jeśli istnieje możliwość, że komunikat może przechodzić przez menedżera kolejek o wcześniejszej wersji.

## Format informacji sterujących komunikatu i danych komunikatu

Menedżer kolejek jest zainteresowany tylko formatem informacji sterujących w komunikacie, podczas gdy aplikacje, które obsługują komunikat, są zainteresowane zarówno formatem informacji sterujących, jak i danymi.

## Format informacji sterujących komunikatu

Informacje sterujące w polach łańcucha znaków deskryptora komunikatu muszą znajdować się w zestawie znaków używanym przez menedżer kolejek.

Ten zestaw znaków jest definiowany przez atrybut **CodedCharSetId** obiektu menedżera kolejek. Informacje sterujące muszą znajdować się w tym zestawie znaków, ponieważ gdy aplikacje przekazują komunikaty z jednego menedżera kolejek do innego, agenty kanału komunikatów przesyłające komunikaty używają wartości tego atrybutu do określenia, jaka konwersja danych ma zostać wykonana.

## Format danych komunikatu

Można określić jedną z następujących opcji:

- Format danych aplikacji
- Zestaw znaków danych znakowych
- Format danych liczbowych

W tym celu należy użyć następujących pól:

***Format***
Wskazuje odbiorcy komunikatu format danych aplikacji w komunikacie.

Gdy menedżer kolejek tworzy komunikat, w pewnych okolicznościach używa pola *Format* do identyfikacji formatu tego komunikatu. Jeśli na przykład menedżer kolejek nie może dostarczyć komunikatu, umieszcza komunikat w kolejce niedostarczonych komunikatów. Dodaje nagłówek (zawierający więcej informacji sterujących) do komunikatu i zmienia pole *Format* , aby to wyświetlić.

Menedżer kolejek ma pewną liczbę *wbudowanych formatów* o nazwach rozpoczynających się od łańcucha MQ, na przykład MQFMT_STRING. Jeśli nie spełniają one potrzeb użytkownika, można zdefiniować własne formaty ( *formaty zdefiniowane przez użytkownika* ), ale nie można w ich przypadku używać nazw rozpoczynających się od łańcucha znaków MQ .

Podczas tworzenia i używania własnych formatów należy napisać wyjście konwersji danych w celu obsługi programu pobierającego komunikat przy użyciu komendy MQGMO_CONVERT.

***CodedCharSetId***
Definiuje zestaw znaków danych znakowych w komunikacie. Aby ustawić ten zestaw znaków na wartość menedżera kolejek, można ustawić w tym polu stałą MQCCSI_Q_MGR lub MQCCSI_INHERIT.

Po uzyskaniu komunikatu z kolejki należy porównać wartość pola *CodedCharSetId* z wartością oczekiwaną przez aplikację. Jeśli te dwie wartości są różne, może być konieczne przekształcenie dowolnych danych znakowych w komunikacie lub użycie wyjścia komunikatu konwersji danych, jeśli jest ono dostępne.

***Encoding***
W tej sekcji opisano format danych liczbowych komunikatu, który zawiera liczby binarne, liczby dziesiętne upakowane i liczby zmiennopozycyjne. Jest on zwykle kodowany zgodnie z konkretnym komputerem, na którym działa menedżer kolejek.

Podczas umieszczania komunikatu w kolejce w polu *Encoding* zwykle należy podać stałą MQENC_NATIVE. Oznacza to, że kodowanie danych komunikatu jest takie samo jak kodowanie na komputerze, na którym działa aplikacja.

Po uzyskaniu komunikatu z kolejki należy porównać wartość pola *Encoding* w deskryptorze komunikatu z wartością stałej MQENC_NATIVE na komputerze. Jeśli te dwie wartości są różne, może być konieczne przekształcenie dowolnych danych liczbowych w komunikacie lub użycie wyjścia komunikatu konwersji danych, jeśli jest ono dostępne.

## *Konwersja danych aplikacji*

Dane aplikacji mogą wymagać konwersji na zestaw znaków i kodowanie wymagane przez inną aplikację w przypadku różnych platform.

Może on zostać przekształcony w nadawczym menedżerze kolejek lub w odbiorczym menedżerze kolejek. Jeśli biblioteka wbudowanych formatów nie spełnia potrzeb, można zdefiniować własne. Typ konwersji zależy od formatu komunikatu określonego w polu formatu deskryptora komunikatu (MQMD).

**Uwaga:** Komunikaty z określoną wartością MQFMT_NONE nie są przekształcane.

## Konwersja w nadawczym menedżerze kolejek

Atrybut kanału CONVERT należy ustawić na wartość YES, jeśli do konwersji danych aplikacji potrzebny jest agent kanału komunikatów wysyłających (MCA).

Konwersja jest wykonywana w nadawczym menedżerze kolejek dla niektórych formatów wbudowanych oraz dla formatów zdefiniowanych przez użytkownika, jeśli podano odpowiednią procedurę zewnętrzną.

**Wbudowane formaty**

takie jak:

- Komunikaty, które są znakami (przy użyciu nazwy formatu MQFMT_STRING)
- Komunikaty zdefiniowane przez IBM MQ , na przykład formaty komend programowalnych

  Produkt IBM MQ używa komunikatów w formacie Programmable Command Format na potrzeby komunikatów administracyjnych i zdarzeń (w tym przypadku używana jest nazwa formatu MQFMT_ADMIN). Dla własnych komunikatów można użyć tego samego formatu (przy użyciu nazwy formatu MQFMT_PCF) i skorzystać z wbudowanej konwersji danych.

Wszystkie wbudowane formaty menedżera kolejek mają nazwy rozpoczynające się od MQFMT. Są one wymienione i opisane w sekcji Format.

**Formaty zdefiniowane przez aplikację**

W przypadku formatów zdefiniowanych przez użytkownika konwersja danych aplikacji musi być wykonywana przez program obsługi wyjścia konwersji danych (więcej informacji na ten temat zawiera sekcja “Zapisywanie wyjść konwersji danych” na stronie 1013 ). W środowisku klient-serwer wyjście jest ładowane na serwerze i odbywa się tam konwersja.

## Konwersja w odbierającym menedżerze kolejek

Dane komunikatów aplikacji mogą być przekształcane przez odbierający menedżer kolejek zarówno dla formatów wbudowanych, jak i zdefiniowanych przez użytkownika.

Konwersja jest wykonywana podczas przetwarzania wywołania MQGET, jeśli określono opcję MQGMO_CONVERT. Szczegółowe informacje na ten temat zawiera sekcja Opcje .

## Kodowane zestawy znaków

Produkty IBM MQ obsługują kodowane zestawy znaków udostępniane przez bazowy system operacyjny.

Podczas tworzenia menedżera kolejek używany identyfikator kodowanego zestawu znaków (CCSID) menedżera kolejek jest oparty na identyfikatorze środowiska bazowego. Jeśli jest to mieszana strona kodowa, program IBM MQ używa części SBCS mieszanej strony kodowej jako identyfikatora CCSID menedżera kolejek.

W przypadku ogólnej konwersji danych, jeśli bazowy system operacyjny obsługuje strony kodowe DBCS, produkt IBM MQ może jej użyć.

Szczegółowe informacje na temat obsługiwanych kodowanych zestawów znaków znajdują się w dokumentacji używanego systemu operacyjnego.

Podczas pisania aplikacji obejmujących wiele platform należy wziąć pod uwagę konwersję danych aplikacji, nazwy formatów i procedury zewnętrzne. Informacje na temat wywoływania i zapisywania wyjść konwersji danych zawiera sekcja “Zapisywanie wyjść konwersji danych” na stronie 1013 .

## Priorytety komunikatów

Priorytet komunikatu można ustawić na wartość numeryczną lub pozwolić, aby komunikat nabył domyślny priorytet kolejki.

Priorytet komunikatu (w polu *Priority* struktury MQMD) ustawia się podczas umieszczania komunikatu w kolejce. Można ustawić wartość liczbową dla priorytetu lub pozwolić, aby komunikat nabył domyślny priorytet kolejki.

Atrybut **MsgDeliverySequence** kolejki określa, czy komunikaty w kolejce są przechowywane w kolejności FIFO (pierwszy przyszedł-pierwszy wyszedł), czy w kolejności FIFO (pierwszy przyszedł-pierwszy wyszedł). Jeśli ten atrybut jest ustawiony na wartość MQMDS_PRIORITY, komunikaty są umieszczane w kolejce z priorytetem określonym w polu *Priority* ich deskryptorów komunikatów, ale jeśli jest ustawiony na wartość MQMDS_FIFO, komunikaty są umieszczane w kolejce z domyślnym priorytetem kolejki. Komunikaty o równym priorytecie są przechowywane w kolejce w kolejności nadejścia.

Atrybut **DefPriority** kolejki ustawia domyślną wartość priorytetu dla komunikatów umieszczanych w tej kolejce. Ta wartość jest ustawiana podczas tworzenia kolejki, ale można ją zmienić później. Kolejki aliasowe i lokalne definicje kolejek zdalnych mogą mieć inne priorytety domyślne niż kolejki podstawowe, na które są tłumaczone. Jeśli w ścieżce rozstrzygania znajduje się więcej niż jedna definicja kolejki (patrz sekcja “Tłumaczenie nazw” na stronie 768 ), priorytet domyślny jest pobierany z wartości (w czasie operacji put) atrybutu **DefPriority** kolejki określonej w komendzie otwierania.

Wartością atrybutu **MaxPriority** menedżera kolejek jest maksymalny priorytet, który można przypisać do komunikatu przetwarzanego przez ten menedżer kolejek. Nie można zmienić wartości tego atrybutu. W programie IBM MQatrybut ma wartość 9. Można tworzyć komunikaty o priorytetach z zakresu od 0 (najniższy) do 9 (najwyższy).

## Właściwości komunikatu

Właściwości komunikatu umożliwiają aplikacji wybieranie komunikatów do przetworzenia lub pobieranie informacji o komunikacie bez uzyskiwania dostępu do nagłówków MQMD lub MQRFH2 . Ułatwiają również komunikację między aplikacjami IBM MQ i JMS .

Właściwość komunikatu to dane powiązane z komunikatem, składające się z nazwy tekstowej i wartości określonego typu. Właściwości komunikatu są używane przez selektory komunikatów do filtrowania publikacji do tematów lub do selektywnego pobierania komunikatów z kolejek. Właściwości komunikatu mogą być używane do dołączania danych biznesowych lub informacji o stanie bez konieczności zapisywania ich w danych aplikacji. Aplikacje nie muszą uzyskiwać dostępu do danych w nagłówkach MQ Message Descriptor (MQMD) lub MQRFH2 , ponieważ do pól w tych strukturach danych można uzyskać dostęp jako do właściwości komunikatu za pomocą wywołań funkcji interfejsu kolejki komunikatów (Message Queue Interface-MQI).

Użycie właściwości komunikatu w produkcie IBM MQ imituje użycie właściwości w produkcie JMS. Oznacza to, że można ustawić właściwości w aplikacji JMS i pobrać je w proceduralnej aplikacji IBM MQ lub w inny sposób. Aby udostępnić właściwość aplikacji JMS , należy przypisać do niej przedrostek usr. Jest on następnie dostępny (bez przedrostka) jako właściwość użytkownika komunikatu JMS . Na przykład właściwość IBM MQ *usr.myproperty* (łańcuch znaków) jest dostępna dla aplikacji JMS za pomocą wywołania JMS `message.getStringProperty('myproperty')`. Należy zauważyć, że aplikacje JMS nie mogą uzyskać dostępu do właściwości z przedrostkiem "usr", jeśli zawierają co najmniej dwa znaki U+002E (".") . Właściwość bez przedrostka i bez U+002E (".") jest traktowany tak, jakby miał przedrostek "usr". I odwrotnie, dostęp do właściwości użytkownika ustawionej w aplikacji JMS można uzyskać w aplikacji IBM MQ , dodając "usr". Przedrostek nazwy właściwości, do której wysłano zapytanie w wywołaniu MQINQMP.

### *Właściwości i długość komunikatu*

Atrybut menedżera kolejek *MaxPropertiesLength* służy do sterowania wielkością właściwości, które mogą przepływać z dowolnym komunikatem w menedżerze kolejek systemu IBM MQ .

Ogólnie, jeśli do ustawiania właściwości używana jest komenda MQSETMP, wielkością właściwości jest długość nazwy właściwości w bajtach plus długość wartości właściwości w bajtach, która została

przekazana do wywołania MQSETMP. Zestaw znaków nazwy właściwości i wartość właściwości mogą ulec zmianie podczas transmisji komunikatu do miejsca docelowego, ponieważ mogą one zostać przekształcone w kod Unicode; w takim przypadku wielkość właściwości może ulec zmianie.

W przypadku wywołania MQPUT lub MQPUT1 właściwości komunikatu nie są liczone jako długość komunikatu dla kolejki i menedżera kolejek, ale są liczone jako długość właściwości postrzeganych przez menedżer kolejek (niezależnie od tego, czy zostały ustawione przy użyciu wywołań MQI właściwości komunikatu).

Jeśli wielkość właściwości przekracza maksymalną długość właściwości, komunikat jest odrzucany z opcją MQRC_PROPERTIES_TOO_BIG. Ponieważ wielkość właściwości zależy od ich reprezentacji, należy ustawić maksymalną długość właściwości na poziomie brutto.

Istnieje możliwość, że aplikacja pomyślnie umieściła komunikat z buforem większym niż wartość parametru *MaxMsgLength*, jeśli bufor zawiera właściwości. Jest to spowodowane tym, że nawet jeśli właściwości komunikatu są reprezentowane jako elementy MQRFH2 , nie są one uwzględniane w długości komunikatu. Pola nagłówka MQRFH2 są dodawane do długości właściwości tylko wtedy, gdy zawiera jeden lub więcej folderów, a każdy folder w nagłówku zawiera właściwości. Jeśli w nagłówku MQRFH2 znajduje się co najmniej jeden folder, a żaden folder nie zawiera właściwości, pola nagłówka MQRFH2 są zliczane do długości komunikatu.

W wywołaniu MQGET właściwości komunikatu nie są uwzględniane w długości komunikatu w odniesieniu do kolejki i menedżera kolejek. Ponieważ jednak właściwości są zliczane oddzielnie, możliwe jest, że bufor zwracany przez wywołanie MQGET jest większy niż wartość atrybutu *MaxMsgLength* .

Aplikacje nie wysyłają zapytań o wartość parametru *MaxMsgLength* , a następnie przydzielają bufor o tej wielkości przed wywołaniem MQGET. Zamiast tego należy przydzielić bufor, który jest wystarczająco duży. Jeśli wykonanie komendy MQGET nie powiedzie się, należy przydzielić bufor o wielkości podanej w parametrze *DataLength* .

Parametr *DataLength* wywołania MQGET zwraca długość (w bajtach) danych aplikacji i wszystkie właściwości zwrócone w podanym buforze, jeśli uchwyt komunikatu nie jest określony w strukturze MQGMO.

Parametr *Buffer* wywołania MQPUT zawiera dane komunikatu aplikacji do wysłania oraz wszystkie właściwości reprezentowane w danych komunikatu.

Istnieje limit długości wynoszący 100 MB dla właściwości komunikatu, z wyłączeniem deskryptora komunikatu lub rozszerzenia dla każdego komunikatu.

Wielkość właściwości w jej wewnętrznej reprezentacji to długość nazwy, plus wielkość jej wartości, plus niektóre dane sterujące dla właściwości. Istnieją również pewne dane sterujące dla zestawu właściwości po dodaniu jednej właściwości do komunikatu.

### *Nazwy właściwości*

Nazwa właściwości jest łańcuchem znaków. Niektóre ograniczenia dotyczą jego długości i zestawu znaków, które mogą być używane.

Nazwa właściwości jest łańcuchem znaków, w którym rozróżniana jest wielkość liter, z ograniczeniem do +4095 znaków, o ile nie jest to ograniczone przez kontekst. Ten limit jest zawarty w stałej MQ_MAX_PROPERTY_NAME_LENGTH.

Jeśli ta maksymalna długość zostanie przekroczona podczas używania wywołania MQI właściwości komunikatu, wywołanie zakończy się niepowodzeniem z kodem przyczyny MQRC_PROPERTY_NAME_LENGTH_ERR.

Ponieważ w produkcie JMSnie ma maksymalnej długości nazwy właściwości, aplikacja JMS może ustawić poprawną nazwę właściwości JMS , która nie jest poprawną nazwą właściwości IBM MQ zapisaną w strukturze MQRFH2 .

W tym przypadku podczas analizowania używane są tylko pierwsze 4095 znaków nazwy właściwości. Następujące znaki są obcinane. Może to spowodować, że aplikacja korzystająca z selektorów nie będzie mogła dopasować łańcucha wyboru lub łańcucha, który nie jest oczekiwany, ponieważ więcej

niż jedna właściwość może zostać obcięta do tej samej nazwy. Po obcięciu nazwy właściwości produkt WebSphereMQ wysyła komunikat dziennika błędów.

Wszystkie nazwy właściwości muszą być zgodne z regułami zdefiniowanymi w specyfikacji języka Java dla identyfikatorów Java , z wyjątkiem tego, że znak Unicode U+002E (.) jest dozwolony jako część nazwy, ale nie jako początek. Reguły dla identyfikatorów Java są równoważne regułom zawartym w specyfikacji JMS dla nazw właściwości.

Znaki spacji, tabulacji lub nowego wiersza oraz operatory porównania są zabronione. Osadzone wartości NULL są dozwolone w nazwie właściwości, ale nie są zalecane. Jeśli używane są osadzone wartości puste, zapobiega to używaniu stałej MQVS_NULL_TERMINATED, gdy jest ona używana ze strukturą MQCHARV do określania łańcuchów o zmiennej długości.

Nazwy właściwości powinny być proste, ponieważ aplikacje mogą wybierać komunikaty na podstawie nazw właściwości i konwersja między zestawem znaków nazwy a selektorem może spowodować nieoczekiwane niepowodzenie wyboru.

Nazwy właściwości IBM MQ używają znaku U+002E (.) do logicznego grupowania właściwości. Spowoduje to podział przestrzeni nazw dla właściwości. Właściwości z następującymi przedrostkami, w dowolnej kombinacji małych i wielkich liter, są zarezerwowane do użycia przez produkt:

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

Dobrym sposobem na uniknięcie konfliktów nazw jest upewnienie się, że wszystkie aplikacje poprzedzają swoje właściwości komunikatu nazwą domeny internetowej. Na przykład w przypadku tworzenia aplikacji przy użyciu nazwy domeny `ourcompany.com` można nadać wszystkim właściwościom nazwę z przedrostkiem `com.ourcompany`. Ta konwencja nazewnictwa umożliwia również łatwy wybór właściwości, na przykład aplikacja może wysłać zapytanie o wszystkie właściwości komunikatu rozpoczynające się od łańcucha `com.ourcompany.%`.

Więcej informacji na temat używania nazw właściwości zawiera sekcja Ograniczenia nazw właściwości .

*Ograniczenia dotyczące nazw właściwości*
Po nadaniu nazwy właściwości należy przestrzegać określonych reguł.

Do nazw właściwości mają zastosowanie następujące ograniczenia:

1. Właściwość nie może zaczynać się od następujących łańcuchów:
   - "JMS"-zarezerwowane do użytku przez IBM MQ classes for JMS.
   - "usr.JMS"-niepoprawne.

   Jedynymi wyjątkami są następujące właściwości udostępniające synonimy właściwości JMS :

| **Właściwość** | **Synonim dla** |
|---|---|
| JMSCorrelationID | Katalog główny.MQMD.CorrelId lub jms.Cid |
| JMSDeliveryMode | Katalog główny.MQMD.Persistence lub jms.Dlv |
| JMSDestination | jms.Dst |
| JMSExpiration | Katalog główny.MQMD.Expiry lub jms.Exp |

| Właściwość | Synonim dla |
|---|---|
| JMSMessageID | Katalog główny.MQMD.MsgId |
| JMSPriority | Katalog główny.MQMD.Priority lub jms.Pri |
| JMSRedelivered | Katalog główny.MQMD.BackoutCount |
| JMSReplyTo (łańcuch zakodowany jako identyfikator URI) | Katalog główny.MQMD.ReplyToQ lub root.MQMD.ReplyToQMgr lub jms.Rto |
| JMSTimestamp | Katalog główny.MQMD.PutDate lub Root.MQMD.PutTime lub jms.Tms |
| JMSType | mcd.Type lub mcd.Set lub mcd.Fmt |
| JMSXAppID | Katalog główny.MQMD.PutApplName |
| JMSXDeliveryCount | Katalog główny.MQMD.BackoutCount |
| JMSXGroupID | Katalog główny.MQMD.GroupId lub jms.Gid |
| JMSXGroupSeq | Katalog główny.MQMD.MsgSeqNumber lub jms.Seq |
| JMSXUserID | Katalog główny.MQMD.UserIdentifier |

Te synonimy umożliwiają aplikacji MQI dostęp do właściwości JMS w podobny sposób jak w przypadku aplikacji klienckiej IBM MQ classes for JMS . Spośród tych właściwości za pomocą interfejsu MQI można ustawiać tylko wartości JMSCorrelationID, JMSReplyTo, JMSType, JMSXGroupIDi JMSXGroupSeq .

Należy zauważyć, że właściwości JMS_IBM_ * dostępne w produkcie IBM MQ classes for JMS nie są dostępne przy użyciu interfejsu MQI. Aplikacje MQI mogą uzyskać dostęp do pól, do których odwołują się właściwości JMS_IBM_ *, w inny sposób.

2. Właściwość nie może być wywoływana w połączeniu z małymi lub wielkimi literami, "NULL", "TRUE", "FALSE", "NOT", "AND", "OR", "BETWEEN", "LIKE", "IN", "IS" i "ESCAPE". Są to nazwy słów kluczowych SQL używanych w łańcuchach wyboru.
3. Nazwa właściwości rozpoczynająca się od " mq " w dowolnej kombinacji małych lub wielkich liter i nie rozpoczynającej się od "mq_usr" może zawierać tylko jeden znak "." znak (U+002E). Wiele "." znaki nie są dozwolone we właściwościach z tymi przedrostkami.
4. Dwa "." Znaki muszą zawierać inne znaki między nimi; w hierarchii nie może występować pusty punkt. Podobnie nazwa właściwości nie może kończyć się znakiem "." .
5. Jeśli aplikacja ustawia właściwość "a.b", a następnie właściwość "a.b.c", nie jest jasne, czy w hierarchii "b" znajduje się wartość, czy inna grupa logiczna. Taka hierarchia jest "treścią mieszaną" i nie jest to obsługiwane. Ustawianie właściwości powodujących mieszaną treść jest niedozwolone.

Ograniczenia te są wymuszane przez mechanizm sprawdzania poprawności w następujący sposób:

- Poprawność nazw właściwości jest sprawdzana podczas ustawiania właściwości za pomocą wywołania MQSETMP-ustawianie właściwości komunikatu , jeśli podczas tworzenia uchwytu komunikatu zażądano sprawdzenia poprawności. Jeśli podjęta próba sprawdzenia poprawności właściwości nie powiedzie się z powodu błędu w specyfikacji nazwy właściwości, kod zakończenia to MQCC_FAILED z następującej przyczyny:
  - Błąd MQRC_PROPERTY_NAME_ERROR z przyczyn 1-4.
  - MQRC_MIXED_CONTENT_NOT_ALLOWED z powodu 5.
- Poprawność nazw właściwości określonych bezpośrednio jako elementy MQRFH2 nie jest gwarantowana przez wywołanie MQPUT.

*Pola deskryptora komunikatu jako właściwości*
Większość pól deskryptora komunikatu może być traktowana jako właściwości. Nazwa właściwości jest tworzona przez dodanie przedrostka do nazwy pola deskryptora komunikatu.

Jeśli aplikacja MQI chce zidentyfikować właściwość komunikatu zawartą w polu deskryptora komunikatu, na przykład w łańcuchu selektora lub przy użyciu funkcji API właściwości komunikatu, należy użyć następującej składni:

| Nazwa właściwości | Pole deskryptora komunikatu |
|---|---|
| Root.MQMD.*Pole* | *Pole* |

W deklaracji języka C należy podać wartość *Field* , tak samo jak w przypadku pól struktury MQMD. Na przykład nazwa właściwości `Root.MQMD.AccountingToken` uzyskuje dostęp do pola `AccountingToken` deskryptora komunikatu.

Pola `StrucId` i `Version` deskryptora komunikatu nie są dostępne przy użyciu przedstawionej składni.

Pola deskryptora komunikatu nie są nigdy reprezentowane w nagłówku MQRFH2 , tak jak w przypadku innych właściwości.

Jeśli dane komunikatu rozpoczynają się od komunikatu MQMDE, który jest honorowany przez menedżer kolejek, dostęp do pól MQMDE można uzyskać przy użyciu opisanej notacji `Root.MQMD.`*`Field`* . W tym przypadku pola MQMDE są traktowane jako logicznie część deskryptora MQMD z perspektywy właściwości. Patrz Przegląd produktu MQMDE.

### *Typy i wartości danych właściwości*

Właściwość może być wartością boolowską, łańcuchem bajtowym, łańcuchem znaków, liczbą zmiennopozycyjną lub liczbą całkowitą. Właściwość może przechowywać dowolną poprawną wartość w zakresie typu danych, o ile nie jest to ograniczone przez kontekst.

Typ danych wartości właściwości musi mieć jedną z następujących wartości:

- MQBOOL
- MQBYTE []
- MQCHAR []
- MQFLOAT32
- MQFLOAT64
- MQINT8
- MQINT16
- MQINT32
- MQINT64

Właściwość może istnieć, ale nie ma zdefiniowanej wartości. Jest to właściwość o wartości NULL. Właściwość o wartości NULL różni się od właściwości bajtowej (MQBYTE []) lub właściwości łańcucha znaków (MQCHAR []), ponieważ ma zdefiniowaną, ale pustą wartość, czyli wartość o zerowej długości.

Łańcuch bajtów nie jest poprawnym typem danych właściwości w systemie JMS lub XMS. Zaleca się, aby nie używać właściwości łańcucha bajtów w folderze *usr* .

## Wybieranie komunikatów z kolejek

Komunikaty z kolejek można wybierać przy użyciu pól MsgId i CorrelId w wywołaniu MQGET lub przy użyciu pola SelectionString w wywołaniu MQOPEN lub MQSUB.

### *Selektory*

Selektor komunikatów to łańcuch o zmiennej długości używany przez aplikację do rejestrowania zainteresowania tylko tymi komunikatami, które mają właściwości zgodne z zapytaniem SQL (Structured Query Language) reprezentowanym przez łańcuch wyboru.

## Wybór przy użyciu wywołań funkcji MQSUB i MQOPEN

Aby dokonać wyboru przy użyciu wywołań MQSUB i MQOPEN, należy użyć obiektu *SelectionString*, który jest strukturą typu MQCHARV.

Struktura *SelectionString* służy do przekazywania do menedżera kolejek łańcucha wyboru o zmiennej długości.

Identyfikator CCSID powiązany z łańcuchem selektora jest ustawiany w polu VSCCSID struktury MQCHARV. Użyta wartość musi być identyfikatorem CCSID, który jest obsługiwany dla łańcuchów selektora. Listę obsługiwanych stron kodowych zawiera sekcja Konwersja stron kodowych .

Określenie identyfikatora CCSID, dla którego nie jest obsługiwana przez IBM MQ konwersja Unicode, powoduje błąd MQRC_SOURCE_CCSID_ERROR. Ten błąd jest zwracany w momencie przedstawienia selektora menedżerowi kolejek, czyli w wywołaniu MQSUB, MQOPEN lub MQPUT1 .

Wartością domyślną w polu *VSCCSID* jest MQCCSI_APPL, która wskazuje, że identyfikator CCSID łańcucha wyboru jest równy identyfikatorowi CCSID menedżera kolejek lub identyfikatorowi CCSID klienta w przypadku połączenia za pośrednictwem klienta. Stała MQCCSI_APPL może jednak zostać przesłonięta przez aplikację ponownie ją definiującą przed kompilacją.

Jeśli selektor MQCHARV reprezentuje łańcuch o wartości NULL, nie jest dokonywany żaden wybór dla tego konsumenta komunikatów, a komunikaty są dostarczane w taki sposób, jakby selektor nie był używany.

Maksymalna długość łańcucha wyboru jest ograniczona tylko tym, co można opisać za pomocą pola MQCHARV *VSLength*.

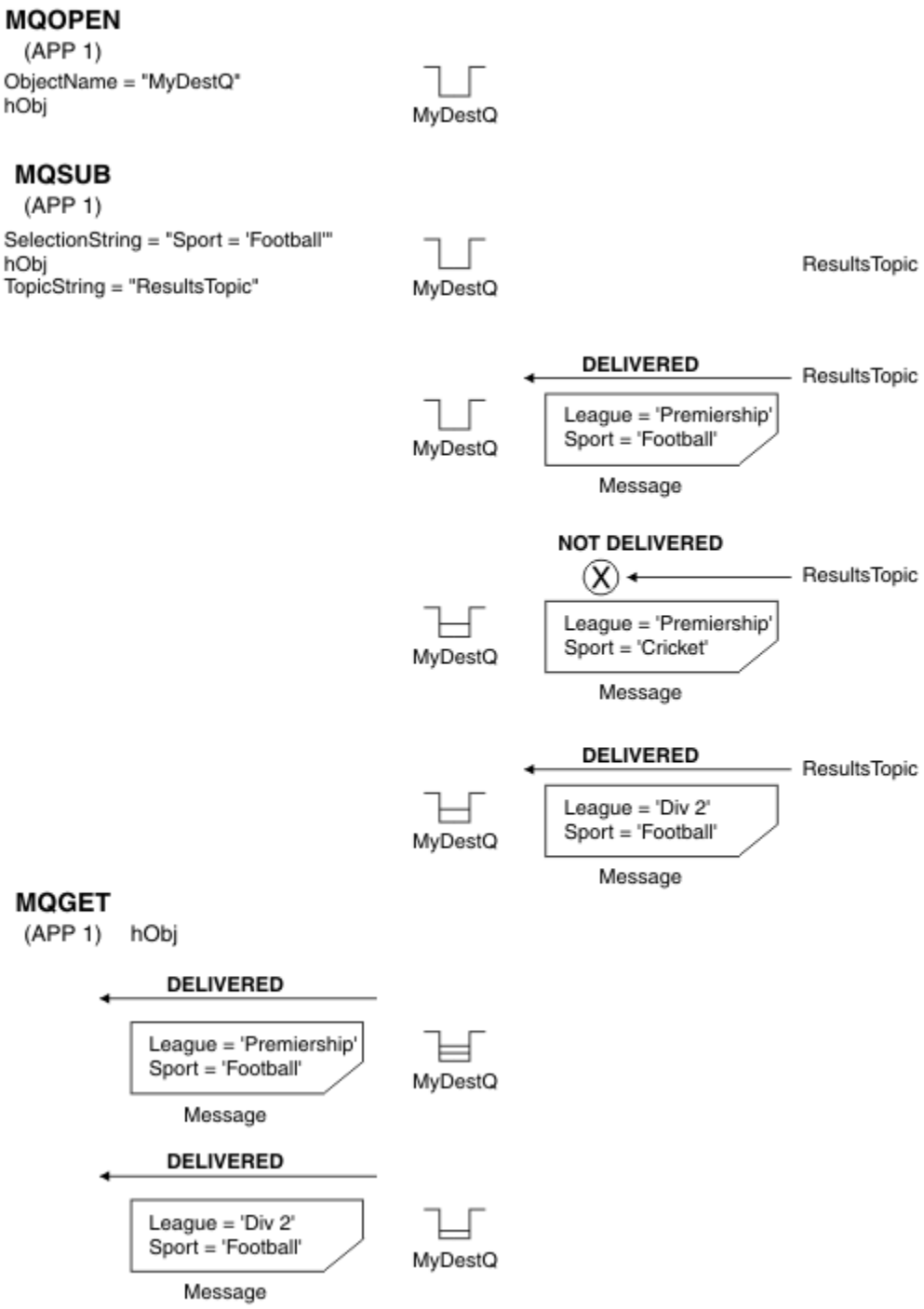
Element SelectionString jest zwracany w danych wyjściowych wywołania MQSUB przy użyciu opcji subskrypcji MQSO_RESUME, jeśli podano bufor i w elemencie VSBufSizewystępuje dodatnia długość buforu. Jeśli bufor nie zostanie podany, w polu VSLength tabeli MQCHARV zostanie zwrócona tylko długość łańcucha wyboru. Jeśli podany bufor jest mniejszy niż obszar wymagany do zwrócenia pola, w podanym buforze są zwracane tylko bajty VSBufSize .

Aplikacja nie może zmienić łańcucha wyboru bez uprzedniego zamknięcia uchwytu do kolejki (dla MQOPEN) lub subskrypcji (dla MQSUB). Nowy łańcuch wyboru można następnie określić w kolejnym wywołaniu MQOPEN lub MQSUB.

**MQOPEN**
Użyj komendy MQCLOSE, aby zamknąć otwarty uchwyt, a następnie określ nowy łańcuch wyboru w kolejnym wywołaniu MQOPEN.

**MQSUB**
Użyj komendy MQCLOSE, aby zamknąć zwrócony uchwyt subskrypcji (hSub), a następnie określ nowy łańcuch wyboru w kolejnym wywołaniu MQSUB.

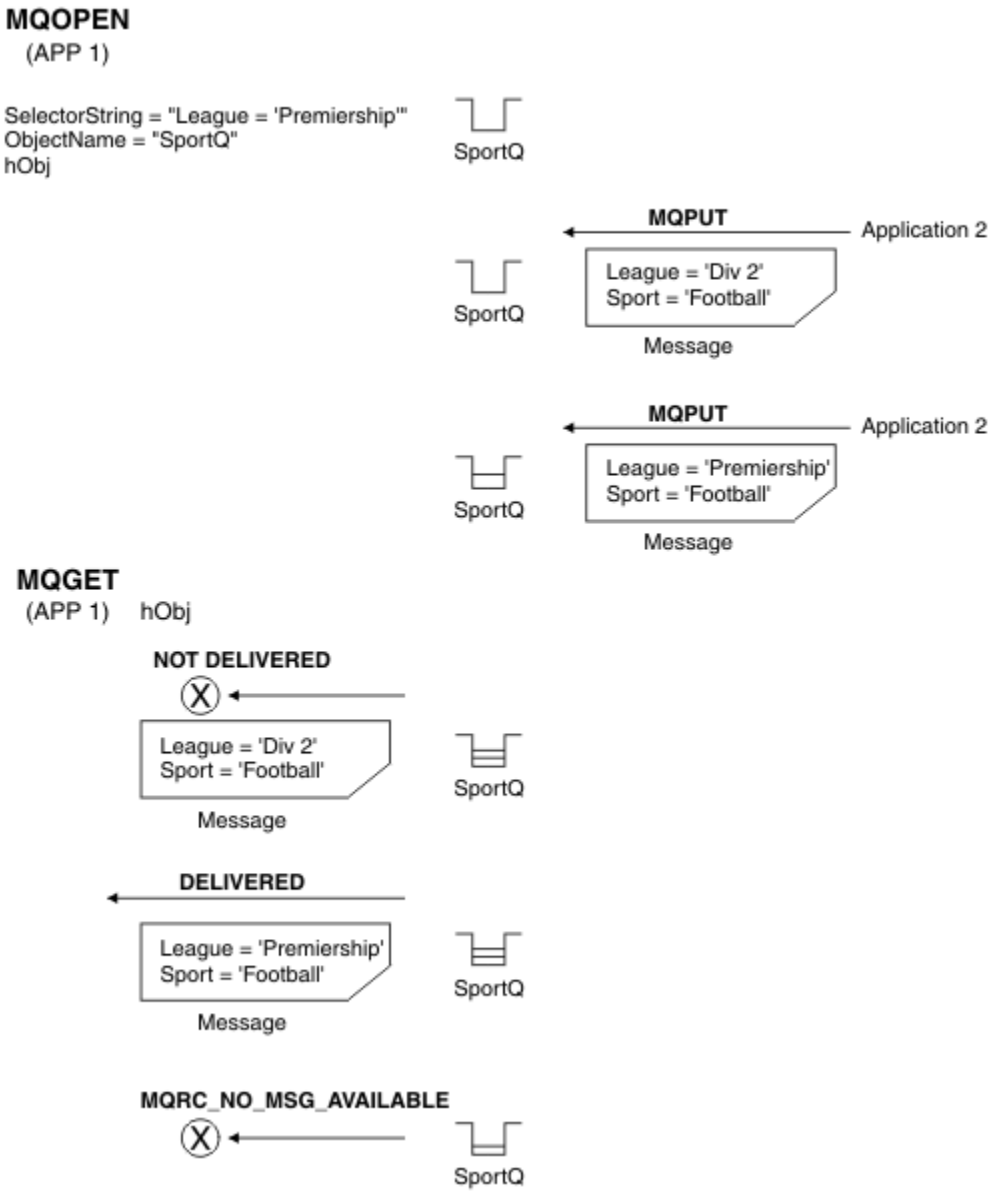
Rysunek 3 na stronie 33 przedstawia proces wyboru przy użyciu wywołania MQSUB.




*Rysunek 3. Wybór przy użyciu wywołania MQSUB*

Selektor można przekazać w wywołaniu do MQSUB za pomocą pola *SelectionString* w strukturze MQSD. Efektem przekazania selektora do MQSUB jest udostępnienie w kolejce docelowej tylko tych komunikatów publikowanych w subskrybowanym temacie, które są zgodne z podanym łańcuchem wyboru.

Rysunek 4 na stronie 34 przedstawia proces wyboru przy użyciu wywołania MQOPEN.




*Rysunek 4. Wybór przy użyciu wywołania MQOPEN*

Selektor można przekazać w wywołaniu do MQOPEN za pomocą pola *SelectionString* w strukturze MQOD. Efektem przekazania selektora w wywołaniu MQOPEN jest dostarczenie do konsumenta komunikatów tylko tych komunikatów w otwartej kolejce, które są zgodne z selektorem.

Głównym zastosowaniem selektora w wywołaniu MQOPEN jest przypadek połączenia punkt z punktem, w którym aplikacja może odbierać tylko te komunikaty w kolejce, które są zgodne z selektorem. W poprzednim przykładzie przedstawiono prosty scenariusz, w którym dwa komunikaty są umieszczane w kolejce otwartej przez MQOPEN, ale tylko jeden jest odbierany przez aplikację, ponieważ jest to jedyny komunikat zgodny z selektorem.

Należy zauważyć, że kolejne wywołania MQGET powodują wywołanie MQRC_NO_MSG_AVAILABLE, ponieważ w kolejce nie istnieją dalsze komunikaty zgodne z podanym selektorem.

**Pojęcia pokrewne**

“Reguły i ograniczenia dotyczące łańcuchów wyboru” na stronie 41
Należy zapoznać się z tymi regułami dotyczącymi sposobu interpretowania łańcuchów wyboru i ograniczeń dotyczących znaków, aby uniknąć potencjalnych problemów podczas korzystania z selektorów.

### *Zachowanie wyboru*

Przegląd zachowania wyboru IBM MQ .

Pola w strukturze MQMDE są traktowane jako właściwości komunikatu dla odpowiednich właściwości deskryptora komunikatu, jeśli deskryptor MQMD:

- Ma format MQFMT_MD_EXTENSION
- Bezpośrednio po nim znajduje się poprawna struktura MQMDE
- Jest to wersja pierwsza lub zawiera tylko domyślne pola wersji drugiej

Łańcuch wyboru może zostać przetłumaczony na wartość TRUE lub FALSE, zanim zostanie przeprowadzone dopasowanie do właściwości komunikatu. Na przykład może tak być, jeśli łańcuch wyboru ma wartość `"TRUE <> FALSE"`. Takie wczesne wartościowanie jest gwarantowane tylko wtedy, gdy w łańcuchu wyboru nie ma odwołań do właściwości komunikatu.

Jeśli łańcuch wyboru ma wartość TRUE, zanim zostaną uwzględnione jakiekolwiek właściwości komunikatu, zostaną dostarczone wszystkie komunikaty opublikowane w temacie zasubskrybowanym przez konsumenta. Jeśli łańcuch wyboru jest tłumaczony na wartość FALSE przed rozważeniem jakichkolwiek właściwości komunikatu, w wywołaniu funkcji, która przedstawiła selektor, zwracany jest kod przyczyny MQRC_SELECTOR_ALWAYS_FALSE i kod zakończenia MQCC_FAILED.

Nawet jeśli komunikat nie zawiera żadnych właściwości komunikatu (innych niż właściwości nagłówka), nadal może zostać zakwalifikowany do wyboru. Jeśli łańcuch wyboru odwołuje się do właściwości komunikatu, która nie istnieje, przyjmuje się, że ta właściwość ma wartość NULL lub 'Nieznany'.

Na przykład komunikat może nadal spełniać łańcuch wyboru, taki jak `'Color IS NULL'`, gdzie `'Color'` nie istnieje jako właściwość komunikatu w komunikacie.

Wyboru można dokonać tylko dla właściwości powiązanych z komunikatem, a nie dla samego komunikatu, chyba że dostępny jest dostawca rozszerzonego wyboru komunikatów. Wybór może być przeprowadzany na ładunku komunikatu tylko wtedy, gdy dostępny jest dostawca rozszerzonego wyboru komunikatów.

Z każdą właściwością komunikatu powiązany jest typ. Podczas dokonywania wyboru należy upewnić się, że wartości używane w wyrażeniach do testowania właściwości komunikatu mają poprawny typ. Jeśli wystąpi niezgodność typów, wyrażenie w pytaniu jest zastępowane wartością FALSE.

Do obowiązków użytkownika należy zapewnienie, że łańcuch wyboru i właściwości komunikatu używają zgodnych typów.

Kryteria wyboru są nadal stosowane w imieniu nieaktywnych stałych subskrybentów, dlatego zachowywane są tylko te komunikaty, które są zgodne z pierwotnie podanym łańcuchem wyboru.

Łańcuchy wyboru są niezmienialne, gdy trwała subskrypcja jest wznawiana ze zmianą (MQSO_ALTER). Jeśli podczas wznawiania działania przez trwały subskrybent zostanie wyświetlony inny łańcuch wyboru, do aplikacji zostanie zwrócona wartość MQRC_SELECTOR_NOT_ALTERABLE.

Aplikacje otrzymują kod powrotu MQRC_NO_MSG_AVAILABLE, jeśli w kolejce nie ma komunikatu spełniającego kryteria wyboru.

Jeśli aplikacja określiła łańcuch wyboru zawierający wartości właściwości, do wyboru kwalifikują się tylko te komunikaty, które zawierają zgodne właściwości. Na przykład subskrybent określa łańcuch wyboru o wartości "a = 3", a komunikat jest publikowany bez właściwości lub właściwości, w których wartość "a" nie istnieje lub jest różna od 3. Subskrybent nie odbiera tego komunikatu do swojej kolejki docelowej.

## Wydajność przesyłania komunikatów

Wybór komunikatów z kolejki wymaga, aby program IBM MQ sekwencyjnie sprawdzany był każdy komunikat w kolejce. Komunikaty są sprawdzane, dopóki nie zostanie znaleziony komunikat zgodny z kryteriami wyboru lub nie ma więcej komunikatów do sprawdzenia. Z tego powodu wydajność przesyłania komunikatów ulega pogorszeniem, jeśli wybór komunikatów jest używany w głębokich kolejkach.

Aby zoptymalizować wybór komunikatów w głębokich kolejkach, gdy wybór jest oparty na JMSCorrelationID lub JMSMessageID, należy użyć łańcucha wyboru w postaci:

- JMSCorrelationID = 'ID:*id_korelacji*'
- JMSMessageID= 'ID:*id_komunikatu*'

gdzie:

- *correlation_id* to łańcuch zawierający standardowy identyfikator korelacji IBM MQ .
- *id_komunikatu* to łańcuch zawierający standardowy identyfikator komunikatu IBM MQ .

**Uwaga:** Selektor powinien odwoływać się tylko do jednej z właściwości. Użycie selektora, który ma jeden z tych formatów, powoduje znaczne zwiększenie wydajności podczas wybierania atrybutu JMSCorrelationID i umożliwia niewielkie zwiększenie wydajności w przypadku atrybutu JMSMessageID. Więcej informacji na ten temat zawiera “Selektory komunikatów w produkcie JMS” na stronie 150.

## Korzystanie z selektorów złożonych

Selektory mogą zawierać wiele komponentów, na przykład:

```
a i b lub c i d lub e i f lub g i h lub i i j... lub y i z
```

Użycie takich złożonych selektorów może mieć poważny wpływ na wydajność i nadmierne wymagania dotyczące zasobów. Z tego powodu program IBM MQ chroni system, nie przetwarzając nadmiernie złożonych selektorów, które mogą spowodować niedobór zasobów systemowych. Ochrona może wystąpić w przypadku łańcuchów wyboru, które zawierają więcej niż 100 testów, lub gdy produkt IBM MQ wykryje zbliżanie się do limitu wielkości stosu systemu operacyjnego. Należy dokładnie przetestować użycie łańcuchów wyboru z wieloma komponentami na odpowiednich platformach, aby upewnić się, że limity ochrony nie zostały osiągnięte.

Wydajność i złożoność selektorów można poprawić, upraszczając je za pomocą dodatkowych nawiasów do łączenia komponentów. Na przykład:

```
( a i b lub c i d) lub ( e i f lub g i h) lub ( i i j) ...
```

**Pojęcia pokrewne**

“Reguły i ograniczenia dotyczące łańcuchów wyboru” na stronie 41
Należy zapoznać się z tymi regułami dotyczącymi sposobu interpretowania łańcuchów wyboru i ograniczeń dotyczących znaków, aby uniknąć potencjalnych problemów podczas korzystania z selektorów.

### *Składnia selektora komunikatów*

Selektor komunikatów IBM MQ to łańcuch ze składnią opartą na podzbiorze składni wyrażenia warunkowego SQL92 .

Selektor komunikatów jest wartościowany w kolejności od lewej do prawej w obrębie poziomu kolejności wykonywania. Aby zmienić tę kolejność, można użyć nawiasów. Predefiniowane literały selektorów i nazwy operatorów są tutaj pisane wielkimi literami, jednak nie jest w nich rozróżniana wielkość liter.

Jeśli selektor jest udostępniany za pośrednictwem interfejsu API, produkt IBM MQ sprawdza poprawność składniową selektora komunikatów w momencie jego prezentowania. Jeśli składnia łańcucha wyboru jest niepoprawna lub nazwa właściwości jest niepoprawna, a dostawca rozszerzonego wyboru komunikatów jest niedostępny, do aplikacji zwracany jest komunikat `MQRC_SELECTION_NOT_AVAILABLE` . Jeśli składnia łańcucha wyboru jest niepoprawna lub nazwa właściwości jest niepoprawna podczas wznawiania subskrypcji, do aplikacji zwracana jest wartość `MQRC_SELECTOR_SYNTAX_ERROR` . Jeśli sprawdzanie poprawności nazwy właściwości zostało wyłączone podczas ustawiania właściwości (przez ustawienie wartości `MQCMHO_NONE` zamiast `MQCMHO_VALIDATE`), a aplikacja następnie umieszcza komunikat z niepoprawną nazwą właściwości, ten komunikat nigdy nie jest wybierany.

Podczas wyświetlania selektora nie jest zwracany żaden błąd, jeśli program IBM MQ określi, że administracyjnie zdefiniowany selektor subskrypcji używa rozszerzonej składni komunikatów, co wskazuje parametr **DISPLAY SUB SELTYPE** o wartości EXTENDED. W takim przypadku sprawdzanie składni łańcucha wyboru jest odraczane do czasu publikacji (patrz `MQRC_SELECTION_NOT_AVAILABLE`).

Selektor może zawierać:

- Literały:
  - Literały łańcuchowe są ujęte w apostrofy. Dwa kolejne pojedyncze cudzysłowy reprezentują pojedynczy cudzysłów. Przykłady to 'literal' i 'literal''s '. Podobnie jak literały łańcuchowe Java , używają one kodowania znaków Unicode. Nie można używać podwójnych cudzysłowów do ujmowania literału łańcuchowego. Między pojedynczymi cudzysłowami można użyć dowolnej sekwencji bajtów.
  - Łańcuch bajtowy to jedna lub więcej par znaków szesnastkowych ujętych w cudzysłów i z przedrostkiem `0x`. Przykłady: `"0x2F1C"` lub `"0XD43A"`. Długość łańcucha bajtowego musi wynosić co najmniej jeden bajt. Jeśli łańcuch bajtowy selektora jest zgodny z właściwością komunikatu typu `MQTYPE_BYTE_STRING`, na początku lub na końcu zera nie jest wykonywane żadne działanie specjalne. Bajty są traktowane jako inny znak. Nie uwzględnia się również endianness. Długość łańcuchów bajtów selektora i właściwości musi być taka sama, a kolejność bajtów musi być taka sama.

    Przykłady zgodnych łańcuchów bajtowych ( *myBytes* = `0AFC23`):

    - `"myBytes = "0x0AFC23"" = TRUE`

    Następujące łańcuchy nie są zgodne:

    - `"myBytes = "0xAFC23"" = MQRC_SELECTOR_SYNTAX_ERROR` (ponieważ liczba bajtów nie jest wielokrotnością dwóch)
    - `"myBytes = "0x0AFC2300"" = FALSE` (ponieważ końcowe zero jest istotne w porównaniu)
    - `"myBytes = "0x000AFC23"" = FALSE` (ponieważ zero wiodące jest istotne w porównaniu)
    - `"myBytes = "0x23FC0A"" = FALSE` (ponieważ układ endian nie jest brany pod uwagę)
  - Liczby szesnastkowe rozpoczynają się od zera, po którym następuje wielka lub mała litera `x`. Pozostała część literału zawiera jeden lub więcej poprawnych znaków szesnastkowych. Przykłady: `0xA`, `0xAF`, `0X2020`.
  - Zero wiodące, po którym występuje co najmniej jedna cyfra z zakresu od 0 do 7, jest zawsze interpretowane jako początek liczby ósemkowej. Nie można przedstawić takiej liczby dziesiętnej z przedrostkiem zero, na przykład `09` zwraca błąd składniowy, ponieważ cyfra 9 nie jest poprawną cyfrą ósemkową. Przykładami liczb ósemkowych są `0177`, `0713`.
  - Dokładny literał liczbowy jest wartością liczbową bez separatora dziesiętnego, taką jak 57, `-957i` `+62`. Dokładny literał liczbowy może mieć na końcu wielką lub małą literę L; nie ma to wpływu na sposób zapisywania lub interpretowania liczby. IBM MQ obsługuje dokładne liczby z zakresu od `-9,223,372,036,854,775,808` do `9,223,372,036,854,775,807`.
  - Przybliżony literał liczbowy jest wartością liczbową w notacji naukowej, takiej jak `7E3` lub `-57.9E2`, lub wartością liczbową z liczbą dziesiętną, taką jak `7.`, `-95.7`lub `+6.2`. IBM MQ obsługuje liczby z zakresu od `-1.797693134862315E+308` do `1.797693134862315E+308`.

    Wartość istotności powinna następować po opcjonalnym znaku ( + lub -). Wartość istotności powinna być liczbą całkowitą lub ułamkiem. Część ułamkowa istotności nie musi mieć cyfry wiodącej.

    Wielkie lub małe litery E oznaczają początek opcjonalnego wykładnika. Wykładnik ma ułamek dziesiętny, a część liczbowa wykładnika może być poprzedzona opcjonalnym znakiem.

    Przybliżone literały liczbowe mogą być zakończone znakiem F lub D (bez rozróżniania wielkości liter). Ta składnia jest przeznaczona do obsługi międzyjęzykowych metod oznaczania liczb o pojedynczej lub podwójnej precyzji. Znaki te są opcjonalne i nie mają wpływu na sposób przechowywania lub przetwarzania przybliżonego literału liczbowego. Liczby te są zawsze zapisywane i przetwarzane z podwójną precyzją.
  - Literały boolowskie `TRUE` i `FALSE`.

  **Uwaga:** Nieskończone reprezentacje IEEE-754 , takie jak `NaN`, `+Infinity`, `-Infinity` , nie są obsługiwane w łańcuchach wyboru. Dlatego nie jest możliwe użycie tych wartości jako operandów w wyrażeniu. Zero ujemne jest traktowane tak samo jak zero dodatnie dla operacji matematycznych.
- Identyfikatory:

Identyfikator jest sekwencją znaków o zmiennej długości, która musi zaczynać się od poprawnego znaku początkowego identyfikatora, po którym następuje zero lub więcej poprawnych znaków części identyfikatora. Reguły dotyczące nazw identyfikatorów są takie same, jak reguły dotyczące nazw właściwości komunikatów. Więcej informacji na ten temat zawiera sekcja “Nazwy właściwości” na stronie 28 i “Ograniczenia dotyczące nazw właściwości” na stronie 29 .

**Uwaga:** Wybór może być przeprowadzany na ładunku komunikatu tylko wtedy, gdy dostępny jest dostawca rozszerzonego wyboru komunikatów.

Identyfikatory są odwołaniami do pól nagłówków lub odwołaniami do właściwości. Typ wartości właściwości w selektorze komunikatów musi odpowiadać typowi używanemu do ustawienia właściwości, chociaż w miarę możliwości wykonywana jest promocja liczbowa. Jeśli wystąpi niezgodność typów, wynikiem wyrażenia jest FALSE. Jeśli przywoływana jest właściwość, która nie istnieje w komunikacie, jej wartość wynosi NULL.

Konwersje typów, które mają zastosowanie do metod get dla właściwości, nie mają zastosowania, gdy właściwość jest używana w wyrażeniu selektora komunikatów. Jeśli na przykład właściwość zostanie ustawiona jako wartość łańcuchowa, a następnie zostanie użyty selektor w celu wykonania zapytania dla niej jako wartości liczbowej, wyrażenie zwróci wartość FALSE.

Nazwy pól i właściwości produktu JMS , które są odwzorowywane na nazwy właściwości lub nazwy pól MQMD , są również poprawnymi identyfikatorami w łańcuchu wyboru. IBM MQ odwzorowuje rozpoznane nazwy pól i właściwości JMS na wartości właściwości komunikatu. Więcej informacji zawiera sekcja “Selektory komunikatów w produkcie JMS” na stronie 150. Na przykład łańcuch wyboru "JMSPriority >=" wybiera właściwość Pri , która znajduje się w folderze jms bieżącego komunikatu.

- Przepełnienie/niedomiar:

  Dla liczb dziesiętnych i przybliżonych są niezdefiniowane następujące warunki:

  – Określanie liczby spoza zdefiniowanego zakresu
  – Określanie wyrażenia arytmetycznego, które spowodowałoby przepełnienie lub niedomiar

  Dla tych warunków nie są wykonywane żadne sprawdzenia.

- Białe znaki:

  Zdefiniowany jako spacja, znak nowego wiersza, znak nowego wiersza, znak powrotu karetki, tabulator poziomy lub tabulator pionowy. Następujące znaki Unicode są rozpoznawane jako białe znaki:

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

- Wyrażenia:

  – Selektor jest wyrażeniem warunkowym. Selektor, który przyjmuje wartość true, jest zgodny. Selektor, który przyjmuje wartość false lub unknown, nie jest zgodny.

  - Wyrażenia arytmetyczne składają się z samych siebie, operacji arytmetycznych, identyfikatorów (wartość identyfikatora jest traktowana jako literał liczbowy) i literałów liczbowych.
  - Wyrażenia warunkowe składają się z samych siebie, operacji porównania i operacji logicznych.
- Obsługiwana jest standardowa funkcja bracketing () do ustawiania kolejności, w jakiej wyrażenia są wartościowane.
- Operatory logiczne w kolejności wykonywania operacji: `NOT`, `AND`, `OR`.
- Operatory porównania: =, >, >=, <, <=, <> (różne od).
  - Łańcuchy dwubajtowe są równe tylko wtedy, gdy łańcuchy mają taką samą długość, a sekwencja bajtów jest taka sama.
  - Można porównywać tylko wartości tego samego typu. Jednym z wyjątków jest to, że poprawne jest porównanie dokładnych wartości liczbowych i przybliżonych wartości liczbowych (wymagana konwersja typów jest zdefiniowana przez reguły promocji numerycznej Java ). Jeśli podejmowana jest próba porównania różnych typów, selektor zawsze ma wartość false.
  - Porównanie łańcuchów i wartości boolowskich jest ograniczone do = i <>. Dwa łańcuchy są równe tylko wtedy, gdy zawierają tę samą sekwencję znaków.
- Operatory arytmetyczne w kolejności wykonywania operacji:
  - +, - jednoargumentowy.
  - * mnożenie i / dzielenie.
  - + dodawanie i - odejmowanie.
  - Operacje arytmetyczne na wartości NULL nie są obsługiwane. Jeśli zostaną podjęte próby, pełny selektor zawsze będzie mieć wartość false.
  - W operacjach arytmetycznych musi być używana promocja liczbowa Java .
- Operator porównania arithmetic-expr1 [ `NOT` ] `BETWEEN` arithmetic-expr2 i arithmetic-expr3 :
  - `Age BETWEEN 15 and 19` jest odpowiednikiem `age>= 15 AND age <= 19`.
  - `Age NOT BETWEEN 15 and 19` jest odpowiednikiem `age < 15 OR age > 19`.
  - Jeśli dowolne wyrażenie operacji `BETWEEN` ma wartość NULL, wartością tej operacji jest false. Jeśli dowolne z wyrażeń operacji `NOT BETWEEN` ma wartość `NULL`, operacja ma wartość true.
- operator porównania identifier `[NOT] IN (string-literal1, string-literal2,...)` , gdzie identyfikator ma wartość typu String lub `NULL` .
  - `Country IN ('UK', 'US', 'France')` ma wartość true dla systemu `'UK'` i false dla systemu `'Peru'`. Jest to odpowiednik wyrażenia `(Country = 'UK') OR (Country = 'US') OR (Country = 'France')`.
  - Parametr `Country NOT IN ('UK', 'US', 'France')` ma wartość false dla bazy danych `'UK'` i wartość true dla bazy danych `'Peru'`. Jest to odpowiednik wyrażenia `NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France'))`.
  - Jeśli identyfikator operacji `IN` lub `NOT IN` ma wartość NULL, wartość operacji jest nieznana.
- Operator porównania `identifier [NOT] LIKE` *`pattern-value`* `[ESCAPE` *`escape-character`* `]` , gdzie `identifier` ma wartość łańcuchową. *wartość_wzorca* jest literałem łańcuchowym, gdzie _ oznacza dowolny pojedynczy znak, a % oznacza dowolną sekwencję znaków (w tym sekwencję pustą). Wszystkie inne znaki są dla siebie. Opcjonalny *znak zmiany znaczenia* jest pojedynczym znakowym literałem łańcuchowym używanym do zmiany znaczenia specjalnego znaczenia znaków _ i % w *wartość_wzorca*. Operator `LIKE` może być używany tylko do porównywania dwóch wartości łańcuchowych.
  - Parametr `phone LIKE '12%3'` ma wartość true dla wartości 123 i 12993 oraz wartość false dla wartości 1234.
  - Parametr `word LIKE 'l_se'` ma wartość true w przypadku utraty danych i false w przypadku utraty danych.

- underscored LIKE '_%' ESCAPE '\' ma wartość true dla systemu _foo i false dla systemu bar.
- Parametr phone NOT LIKE '12%3' ma wartość false w przypadku systemów 123 i 12993 oraz wartość true w przypadku systemu 1234.
- Jeśli identyfikator operacji LIKE lub NOT LIKE ma wartość NULL, wartość operacji jest nieznana.

**Uwaga:** Operator LIKE musi być używany do porównywania dwóch wartości łańcuchowych. Wartość Root.MQMD.CorrelId jest 24-bajtową tablicą, a nie łańcuchem znaków. Łańcuch selektora Root.MQMD.CorrelId LIKE 'ABC%' jest akceptowany przez analizator składni jako poprawny pod względem składniowym, ale jego wynikiem jest wartość false. W przypadku porównywania tablicy bajtów z łańcuchem znaków nie można użyć łańcucha LIKE .

- Operator porównania identifier IS NULL testuje wartość pola nagłówka NULL lub brakującą wartość właściwości.
- Operator porównania identifier IS NOT NULL sprawdza, czy istnieje wartość pola nagłówka inna niż NULL lub wartość właściwości.
- Wartości NULL

  Wartościowanie wyrażeń selektorów, które zawierają wartości NULL , jest definiowane przez semantykę języka SQL 92 NULL w podsumowaniu:

  - SQL traktuje wartość NULL jako nieznaną.
  - Porównanie lub arytmetyka z nieznaną wartością zawsze zwraca nieznaną wartość.
  - Operatory IS NULL i IS NOT NULL przekształcają nieznaną wartość w wartości TRUE i FALSE .

  Operatory boolowskie używają trójwartościowej logiki ( T=TRUE, F=FALSE, U=UNKNOWN)

*Tabela 1. Wartość wyniku operatora boolowskiego, gdy logika ma wartość A AND B*

| **Operator A** | **Operator B** | **Wynik (A I B)** |
|---|---|---|
| T | F | F |
| T | U | U |
| T | T | T |
| F | T | F |
| F | U | F |
| F | F | F |
| U | T | U |
| U | U | U |
| U | F | F |

*Tabela 2. Wartość wyniku operatora boolowskiego, gdy logika ma wartość A OR B*

| **Operator A** | **Operator B** | **Wynik (A OR B)** |
|---|---|---|
| T | F | T |
| T | U | T |
| T | T | T |
| F | T | T |
| F | U | U |
| F | F | F |

| *Tabela 2. Wartość wyniku operatora boolowskiego, gdy logika ma wartość A OR B (kontynuacja)* | | |
|---|---|---|
| **Operator A** | **Operator B** | **Wynik (A OR B)** |
| U | T | T |
| U | U | U |
| U | F | U |

| *Tabela 3. Wartość wyniku operatora boolowskiego, gdy logika ma wartość NOT A* | |
|---|---|
| **Operator A** | **Wynik ( NOT A)** |
| T | F |
| F | T |
| U | U |

Następujący selektor komunikatów wybiera komunikaty o typie komunikatu samochód, kolorze niebieskim i wadze większej niż 2500 funtów:

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

Chociaż SQL obsługuje stałe porównania dziesiętne i arytmetyczne, selektory komunikatów nie. Dlatego dokładne literały liczbowe są ograniczone do tych, które nie mają liczby dziesiętnej. Jest to również powód, dla którego istnieją liczby z liczbą dziesiętną jako alternatywną reprezentację przybliżonej wartości liczbowej.

Komentarze SQL nie są obsługiwane.

**Pojęcia pokrewne**

“Właściwości komunikatu” na stronie 27
Właściwości komunikatu umożliwiają aplikacji wybieranie komunikatów do przetworzenia lub pobieranie informacji o komunikacie bez uzyskiwania dostępu do nagłówków MQMD lub MQRFH2 . Ułatwiają również komunikację między aplikacjami IBM MQ i JMS .

**Odsyłacze pokrewne**

MsgHandle

MQBUFMH-Przekształć bufor w uchwyt komunikatu

*Reguły i ograniczenia dotyczące łańcuchów wyboru*

Należy zapoznać się z tymi regułami dotyczącymi sposobu interpretowania łańcuchów wyboru i ograniczeń dotyczących znaków, aby uniknąć potencjalnych problemów podczas korzystania z selektorów.

- Wybór komunikatu do przesyłania komunikatów w trybie publikowania/subskrypcji występuje w komunikacie wysyłanym przez publikator. Patrz sekcja Łańcuchy wyboru.
- Równoważność jest testowana przy użyciu pojedynczego znaku równości; na przykład a = b jest poprawne, a a == b jest niepoprawne.
- Operatorem używanym przez wiele języków programowania do reprezentowania 'not equal to' jest != . Ta reprezentacja nie jest poprawnym synonimem terminu <> ; Na przykład wartość a <> b jest poprawna, a wartość a != b nie jest poprawna.
- Pojedyncze cudzysłowy są rozpoznawane tylko wtedy, gdy Używany jest znak (U+0027). Podobnie podwójne cudzysłowy, poprawne tylko wtedy, gdy są używane do ujmowania łańcuchów bajtów, muszą zawierać znak " (U+0022).
- Symbole &, &&, | i | | nie są synonimami logicznego łączenia/rozłączania; na przykład a && b musi być określone jako a AND b.
- Znaki wieloznaczne * i ? nie są synonimami % i _.

- Selektory zawierające wyrażenia złożone, takie jak `20 < b < 30` , są niepoprawne. Analizator składni wartościuje operatory, które mają taki sam priorytet od lewej do prawej. Przykładem może być `(20 < b) < 30`, co nie ma sensu. Zamiast tego wyrażenie musi być zapisane jako `(b > 20) AND (b < 30)`.
- Łańcuchy bajtowe muszą być ujęte w cudzysłów; jeśli używane są pojedyncze cudzysłowy, łańcuch bajtowy jest przyjmowany jako literał łańcuchowy. Liczba znaków (a nie liczba reprezentowana przez znaki) po znaku `0x` musi być wielokrotnością liczby dwóch.
- Słowo kluczowe `IS` nie jest synonimem znaku równości. Dlatego łańcuchy wyboru a `IS 3` i `b IS 'red'` są niepoprawne. Słowo kluczowe `IS` istnieje tylko w celu obsługi spraw `IS NULL` i `IS NOT NULL` .

**Pojęcia pokrewne**

“Zachowanie wyboru” na stronie 35
Przegląd zachowania wyboru IBM MQ .

**Odsyłacze pokrewne**

Łańcuchy wyboru

*Uwagi dotyczące kodowania UTF-8 i Unicode podczas korzystania z selektorów komunikatów*

Znaki nieujęte w apostrofy, które tworzą zastrzeżone słowa kluczowe łańcucha wyboru, muszą być wprowadzane w języku Basic Latin Unicode (od U+0000 do U+0007F). Nie można używać innych reprezentacji znaków alfanumerycznych w punktach kodowych. Na przykład, liczba 1 musi być wyrażona jako U+0031 w kodzie Unicode, nie można użyć odpowiednika w postaci cyfr pełnej szerokości U+FF11 lub odpowiednika w języku arabskim U+0661.

Nazwy właściwości komunikatu można określić za pomocą dowolnej poprawnej sekwencji znaków Unicode. Poprawność nazw właściwości komunikatu zawartych w łańcuchach wyboru, które są zakodowane w UTF-8 , zostanie sprawdzona nawet wtedy, gdy zawierają znaki wielobajtowe. Sprawdzanie poprawności wielobajtowego kodu UTF-8 jest ścisłe i należy upewnić się, że dla nazw właściwości komunikatu używane są poprawne sekwencje UTF-8 . Znaki spoza obszaru Unicode Basic Multilingual Plane (powyżej U + FFFF), reprezentowane w UTF-16 przez odpowiedniki punktów kodowych (X'D800'do X'DFFF') lub cztery bajty w UTF-8, nie są obsługiwane w nazwach właściwości komunikatów.

Podczas porównywania pod kątem równości nie jest wykonywane żadne dodatkowe przetwarzanie nazw ani wartości właściwości. Oznacza to na przykład, że nie ma miejsca przed/dekompozycja i więzadła nie mają żadnego specjalnego znaczenia. Na przykład wstępnie skomponowany znak umlaut U+00FC nie jest traktowany jako odpowiednik U+0075 + U+0308 , a sekwencja znaków ff nie jest traktowana jako odpowiednik kodu Unicode U+FB00 (LATIN SMALL LIGATURE FF).

Dane właściwości ujęte w pojedynczy cudzysłów mogą być reprezentowane przez dowolną sekwencję bajtów i nie jest sprawdzana ich poprawność.

## *Wybieranie treści komunikatu*

Istnieje możliwość subskrybowania na podstawie wyboru treści ładunku komunikatu (zwanego również filtrowaniem treści), ale decyzja o tym, które komunikaty mają być dostarczane do takiej subskrypcji, nie może być podejmowana bezpośrednio przez produkt IBM MQ. Zamiast tego do przetwarzania komunikatów wymagany jest dostawca rozszerzonego wyboru komunikatów, na przykład IBM Integration Bus.

Gdy aplikacja publikuje w łańcuchu tematu, w którym co najmniej jeden subskrybent ma wybrany łańcuch wyboru dla treści komunikatu, produkt IBM MQ zażąda, aby dostawca rozszerzonego wyboru komunikatów przeanalizował publikację i poinformował IBM MQ , czy publikacja jest zgodna z kryteriami wyboru określonymi przez każdego subskrybenta z filtrem treści.

Jeśli dostawca rozszerzonego wyboru komunikatów określi, że publikacja jest zgodna z łańcuchem wyboru subskrybenta, komunikat będzie nadal dostarczany do subskrybenta.

Jeśli dostawca rozszerzonego wyboru komunikatów stwierdzi, że publikacja nie jest zgodna, komunikat nie zostanie dostarczony do subskrybenta. Może to spowodować niepowodzenie wywołania MQPUT lub MQPUT1 z kodem przyczyny MQRC_PUBLICATION_FAILURE. Jeśli dostawca rozszerzonego wyboru

komunikatów nie może przeanalizować publikacji, zwracany jest kod przyczyny MQRC_CONTENT_ERROR, a wywołanie MQPUT lub MQPUT1 kończy się niepowodzeniem.

Jeśli dostawca wyboru rozszerzonego komunikatu jest niedostępny lub nie może określić, czy subskrybent powinien otrzymać publikację, zwracany jest kod przyczyny MQRC_SELECTION_NOT_AVAILABLE i wywołanie MQPUT lub MQPUT1 kończy się niepowodzeniem.

Gdy subskrypcja jest tworzona z filtrem treści i dostawca rozszerzonego wyboru komunikatów jest niedostępny, wywołanie MQSUB kończy się niepowodzeniem z kodem przyczyny MQRC_SELECTION_NOT_AVAILABLE. Jeśli subskrypcja z filtrem treści jest wznawiana, a dostawca rozszerzonego wyboru komunikatów jest niedostępny, wywołanie MQSUB zwraca ostrzeżenie MQRC_SELECTION_NOT_AVAILABLE, ale subskrypcja może zostać wznowiona.

**Odsyłacze pokrewne**

Łańcuchy wyboru

## Asynchroniczne korzystanie z komunikatów IBM MQ

Wykorzystanie asynchroniczne korzysta z zestawu rozszerzeń interfejsu kolejki komunikatów (Message Queue Interface-MQI), wywołania MQCB i MQCTL, które umożliwiają zapisywanie aplikacji MQI w celu odbierania komunikatów z zestawu kolejek. Komunikaty są dostarczane do aplikacji przez wywołanie jednostki kodu identyfikowanej przez aplikację przekazującą komunikat lub znacznik reprezentujący komunikat.

W najbardziej prostych środowiskach aplikacji jednostka kodu jest definiowana przez wskaźnik funkcji, jednak w innych środowiskach jednostka kodu może być definiowana przez nazwę programu lub modułu.

W przypadku asynchronicznego korzystania z komunikatów używane są następujące terminy:

**Konsument komunikatu**
: Konstrukcja programistyczna, która umożliwia zdefiniowanie programu lub funkcji wywoływanej z komunikatem, gdy jest on zgodny z wymaganiami aplikacji.

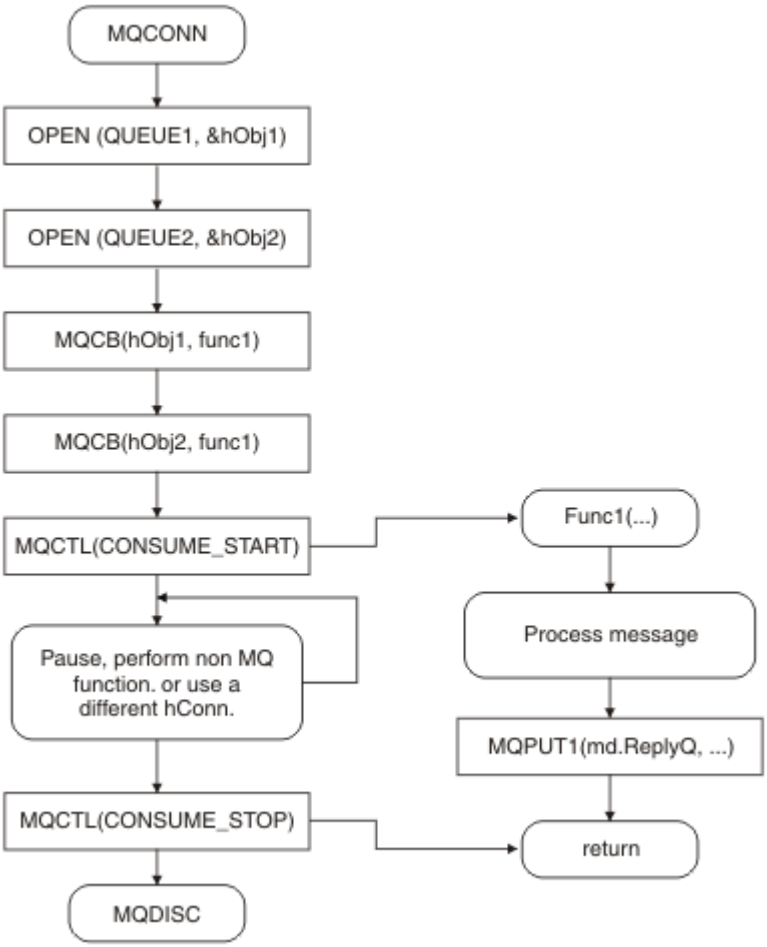
**procedura obsługi zdarzeń**
: Konstrukcja programistyczna, która umożliwia zdefiniowanie programu lub funkcji wywołanej w momencie wystąpienia zdarzenia asynchronicznego, takiego jak wyciszanie menedżera kolejek.

**Wywołanie zwrotne**
: Termin ogólny używany do odwoływania się do procedury konsumenta komunikatów lub procedury obsługi zdarzeń.

Wykorzystanie asynchroniczne może uprościć projektowanie i implementację nowych aplikacji, zwłaszcza tych, które przetwarzają wiele kolejek wejściowych lub subskrypcji. Jeśli jednak używana jest więcej niż jedna kolejka wejściowa i przetwarzane są komunikaty w kolejności priorytetów, kolejność priorytetów jest obserwowana niezależnie w każdej kolejce: komunikaty o niskim priorytecie mogą być pobierane z jednej kolejki przed komunikatami o wysokim priorytecie z innej kolejki. Kolejność komunikatów w wielu kolejkach nie jest gwarantowana. Należy również zauważyć, że jeśli używane są wyjścia funkcji API, może być konieczna ich zmiana w celu uwzględnienia wywołań MQCB i MQCTL.

Poniższe ilustracje przedstawiają przykład użycia tej funkcji.

Rysunek 5 na stronie 44 przedstawia wielowątkową aplikację odbierającą komunikaty z dwóch kolejek. Przykład przedstawia wszystkie komunikaty dostarczane do pojedynczej funkcji.

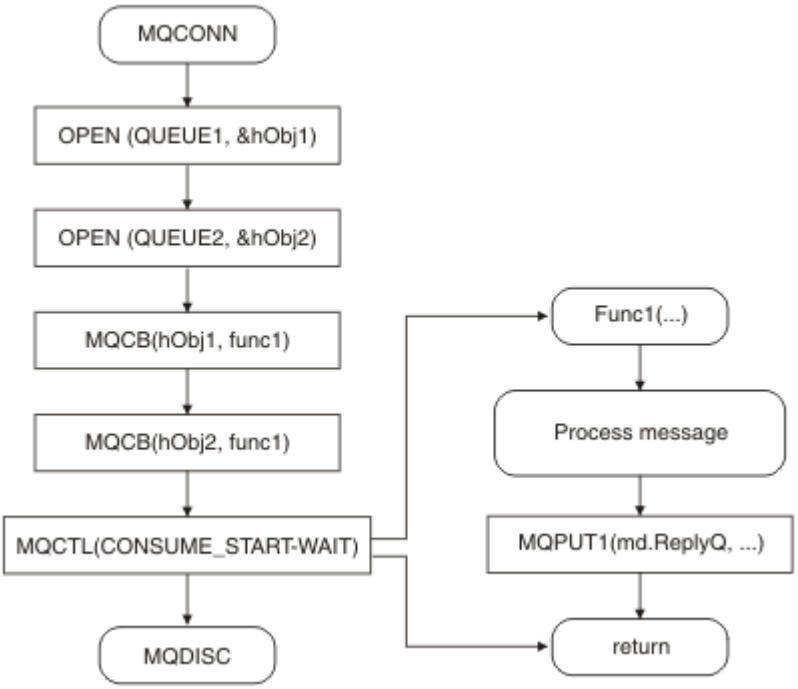



*Rysunek 5. Standardowa aplikacja sterowana komunikatami korzystająca z dwóch kolejek*

z/OS W systemie z/OSgłówny wątek sterujący musi wywołać komendę MQDISC przed zakończeniem działania. Umożliwia to kończenie i zwalnianie zasobów systemowych przez dowolny wątek wywołania zwrotnego.

Rysunek 6 na stronie 45 W tym przykładowym przepływie przedstawiono pojedynczą aplikację wielowątkową odbierającą komunikaty z dwóch kolejek. Przykład przedstawia wszystkie komunikaty dostarczane do pojedynczej funkcji.

Różnica w porównaniu z przypadkiem asynchronicznym polega na tym, że sterowanie nie powraca do wystawcy obiektu MQCTL, dopóki wszyscy konsumenci nie zdezaktywują się samodzielnie. Oznacza to, że jeden konsument wysłał żądanie MQCTL STOP lub menedżer kolejek jest wygaszony.




*Rysunek 6. Aplikacja sterowana komunikatami jednowątkowymi korzystająca z dwóch kolejek*

## Grupy komunikatów

Komunikaty mogą występować w grupach, aby umożliwić uporządkowanie komunikatów.

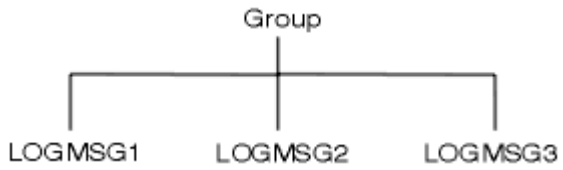
Grupy komunikatów umożliwiają oznaczanie wielu komunikatów jako powiązanych ze sobą oraz stosowanie do grupy porządku logicznego (patrz sekcja "Porządkowanie logiczne i fizyczne" na stronie 798). W systemie Wiele platform segmentacja komunikatów umożliwia podział dużych komunikatów na mniejsze segmenty. Podczas umieszczania w temacie nie można używać zgrupowanych ani posegmentowanych komunikatów.

Hierarchia w grupie jest następująca:

**Grupa**
Jest to najwyższy poziom w hierarchii i jest identyfikowany przez *GroupId*. Składa się on z jednego lub większej liczby komunikatów, które zawierają ten sam plik *GroupId*. Komunikaty te mogą być przechowywane w dowolnym miejscu w kolejce.

**Uwaga:** Termin *komunikat* jest używany w tym miejscu do oznaczenia jednego elementu w kolejce, na przykład zwróconego przez pojedynczą operację MQGET, która nie określa parametru MQGMO_COMPLETE_MSG.

Rysunek 7 na stronie 45 przedstawia grupę komunikatów logicznych:




*Rysunek 7. Grupa komunikatów logicznych*

Otwarcie kolejki i określenie opcji MQOO_BIND_ON_GROUP powoduje wymuszenie wysyłania wszystkich komunikatów w grupie, które są wysyłane do tej kolejki, do tej samej instancji kolejki. Więcej informacji na temat opcji BIND_ON_GROUP zawiera sekcja Obsługa powinowactwa komunikatów.

**Komunikat logiczny**

Komunikaty logiczne w grupie są identyfikowane za pomocą pól *GroupId* i *MsgSeqNumber* . Wartość *MsgSeqNumber* rozpoczyna się od 1 dla pierwszego komunikatu w grupie, a jeśli komunikat nie znajduje się w grupie, wartość pola wynosi 1.

Komunikaty logiczne w grupie umożliwiają:

- Należy zapewnić zamawianie (jeśli nie jest to zagwarantowane w okolicznościach, w których komunikat jest przesyłany).
- Zezwalaj aplikacjom na grupowanie podobnych komunikatów (na przykład tych, które muszą być przetworzone przez tę samą instancję serwera).

Każdy komunikat w grupie składa się z jednego komunikatu fizycznego, chyba że jest on podzielony na segmenty. Każdy komunikat jest pod względem logicznym osobnym komunikatem, a tylko pola *GroupId* i *MsgSeqNumber* w strukturze MQMD muszą zawierać wszelkie relacje z innymi komunikatami w grupie. Inne pola w strukturze MQMD są niezależne; niektóre mogą być identyczne dla wszystkich komunikatów w grupie, podczas gdy inne mogą być różne. Na przykład komunikaty w grupie mogą mieć różne nazwy formatów, identyfikatory CCSID i kodowania.

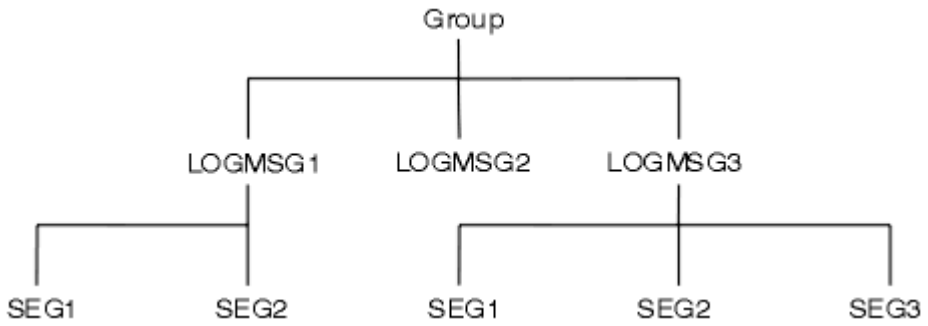
**Segment**

Segmenty są używane do obsługi komunikatów, które są zbyt duże, aby można było umieścić lub uzyskać aplikację lub menedżer kolejek (w tym menedżery kolejek, przez które komunikat jest przekazywany). Więcej informacji na ten temat zawiera sekcja "Segmentacja komunikatów" na stronie 817.

Pojedynczy komunikat jest dzielony na mniejsze komunikaty nazywane *segmentami*. Segment komunikatu jest identyfikowany przez pola *GroupId*, *MsgSeqNumber*i *Offset* . Pole *Offset* rozpoczyna się od zera dla pierwszego segmentu w komunikacie.

Każdy segment składa się z jednego komunikatu fizycznego, który może należeć do grupy ( Rysunek 8 na stronie 46 przedstawia przykład komunikatów w grupie). Segment jest logicznie częścią pojedynczego komunikatu, więc tylko pola *MsgId*, *Offset*i *MsgFlags* w deskryptorze MQMD powinny się różnić między oddzielnymi segmentami tego samego komunikatu. Jeśli odebranie segmentu nie powiedzie się, zwracany jest odpowiednio kod przyczyny MQRC_INCOMPLETE_GROUP lub MQRC_INCOMPLETE_MSG .

Rysunek 8 na stronie 46 przedstawia grupę komunikatów logicznych, z których niektóre są podzielone na segmenty:




*Rysunek 8. Segmentowane komunikaty*

z/OS Segmentacja nie jest obsługiwana w systemie IBM MQ for z/OS.

Z publikowaniem/subskrybowaniem nie można używać segmentowanych lub zgrupowanych komunikatów.

**Pojęcia pokrewne**

"Segmentacja komunikatów" na stronie 817
Ten temat zawiera informacje na temat segmentacji komunikatów. Ta funkcja nie jest obsługiwana w systemie IBM MQ for z/OS ani przez aplikacje korzystające z produktu IBM MQ classes for JMS.

**Odsyłacze pokrewne**

"Porządkowanie logiczne i fizyczne" na stronie 798
Komunikaty w kolejkach mogą występować (w obrębie każdego poziomu priorytetu) w kolejności *fizycznej* lub *logicznej* .

MQMD-deskryptor komunikatu

## Trwałość komunikatu

Komunikaty trwałe są zapisywane w dziennikach i plikach danych kolejki. Jeśli menedżer kolejek zostanie zrestartowany po awarii, odzyskuje te komunikaty trwałe, jeśli jest to konieczne, na podstawie zarejestrowanych danych. Komunikaty, które nie są trwałe, są usuwane w przypadku zatrzymania menedżera kolejek, niezależnie od tego, czy jest to spowodowane przez komendę operatora, czy przez awarię części systemu.

z/OS Wyjątkiem są nietrwałe komunikaty przechowywane w narzędziu CF w systemie z/OS . Utrzymują się one tak długo, jak długo system CF jest dostępny.

Podczas tworzenia komunikatu, jeśli deskryptor komunikatu (MQMD) jest inicjowany przy użyciu wartości domyślnych, trwałość komunikatu jest pobierana z atrybutu **DefPersistence** kolejki określonej w komendzie MQOPEN. Można również ustawić trwałość komunikatu przy użyciu pola *Persistence* struktury MQMD, aby zdefiniować komunikat jako trwały lub nietrwały.

Wydajność aplikacji ma wpływ na korzystanie z trwałych komunikatów. Zakres wpływu zależy od parametrów wydajności podsystemu we/wy komputera i sposobu użycia opcji punktu synchronizacji na każdej platformie:

- Trwały komunikat, poza bieżącą jednostką pracy, jest zapisywany na dysku przy każdej operacji umieszczania i pobierania. Patrz "Zatwierdzanie i wycofywanie jednostek pracy" na stronie 879.
- z/OS ALW W przypadku wszystkich platform z wyjątkiem platformy IBM ikomunikat trwały w bieżącej jednostce pracy jest rejestrowany tylko wtedy, gdy jednostka pracy została zatwierdzona, a jednostka pracy może zawierać wiele operacji w kolejce.

Komunikaty nietrwałe mogą być używane na potrzeby szybkiego przesyłania komunikatów. Więcej informacji na temat szybkich komunikatów zawiera sekcja Bezpieczeństwo komunikatów .

**Uwaga:** Kombinacja zapisywania trwałych komunikatów w jednostce pracy i zapisywania trwałych komunikatów poza jednostką lub pracą może spowodować poważne problemy z wydajnością aplikacji. Jest to szczególnie istotne, gdy ta sama kolejka docelowa jest używana dla obu operacji.

## Komunikaty, których dostarczenie nie powiodło się

Jeśli menedżer kolejek nie może umieścić komunikatu w kolejce, dostępne są różne opcje.

Użytkownik może:

- Spróbuj ponownie umieścić komunikat w kolejce.
- Zażądaj, aby komunikat został zwrócony do nadawcy.
- Umieść komunikat w kolejce niedostarczonych komunikatów.

Więcej informacji na ten temat zawiera sekcja"Obsługa błędów proceduralnych programu" na stronie 1069.

## Wycofane komunikaty

Podczas przetwarzania komunikatów z kolejki pod kontrolą jednostki pracy jednostka pracy może składać się z jednego lub większej liczby komunikatów. Jeśli wystąpi wycofanie, komunikaty, które zostały pobrane z kolejki, zostaną ponownie umieszczone w kolejce i mogą zostać przetworzone ponownie w innej jednostce pracy. Jeśli przetwarzanie konkretnego komunikatu powoduje problem, jednostka pracy jest wycofywana ponownie. Może to spowodować powstanie pętli przetwarzania. Komunikaty, które zostały umieszczone w kolejce, są usuwane z kolejki.

Aplikacja może wykryć komunikaty, które są wychwycone w takiej pętli, testując pole *BackoutCount* deskryptora MQMD. Aplikacja może poprawić sytuację lub wydać ostrzeżenie operatorowi.

Multi Liczba wycofań zawsze powoduje ponowne uruchomienie menedżera kolejek. Każda zmiana atrybutu **HardenGetBackout** jest ignorowana.

z/OS W przypadku kolejek współużytkowanych liczba wycofanych komunikatów zawsze jest restartowana w menedżerze kolejek. W przypadku wszystkich innych konfiguracji w systemie z/OS, aby zapewnić, że liczba wycofań dla kolejek prywatnych przetrwa restarty menedżera kolejek, należy ustawić atrybut *HardenGetBackout* na wartość MQQA_BACKOUT_HARDENED; w przeciwnym razie, jeśli menedżer kolejek musi zostać zrestartowany, nie będzie zachowywał dokładnej liczby wycofań dla każdego komunikatu. Ustawienie atrybutu w ten sposób powoduje dodanie kosztu dodatkowego przetwarzania.

Więcej informacji na temat zatwierdzania i wycofywania komunikatów zawiera sekcja "Zatwierdzanie i wycofywanie jednostek pracy" na stronie 879.

## Kolejka odpowiedzi i menedżer kolejek

Mogą wystąpić sytuacje, w których użytkownik otrzymuje komunikaty w odpowiedzi na wysłany komunikat:

- Komunikat odpowiedzi w odpowiedzi na komunikat żądania
- Komunikat raportu o nieoczekiwanym zdarzeniu lub utracie ważności
- Komunikat raportu dotyczący zdarzenia COA (potwierdzenie przybycia) lub COD (potwierdzenie dostarczenia)
- Komunikat raportu o zdarzeniu PAN (Pozytywne powiadomienie o działaniu) lub NAN (Negatywne powiadomienie o działaniu)

Za pomocą struktury MQMD w polu *ReplyToQ* określ nazwę kolejki, do której mają być wysyłane komunikaty odpowiedzi i raporty. W polu *ReplyToQMgr* podaj nazwę menedżera kolejek, który jest właścicielem kolejki odpowiedzi.

Jeśli pole *ReplyToQMgr* pozostanie puste, menedżer kolejek ustawi treść następujących pól w deskryptorze komunikatu w kolejce:

***ReplyToQ***
Jeśli *ReplyToQ* jest lokalną definicją kolejki zdalnej, pole *ReplyToQ* jest ustawione na nazwę kolejki zdalnej. W przeciwnym razie to pole nie zostanie zmienione.

***ReplyToQMgr***
Jeśli *ReplyToQ* jest lokalną definicją kolejki zdalnej, pole *ReplyToQMgr* jest ustawione na nazwę menedżera kolejek, który jest właścicielem kolejki zdalnej. W przeciwnym razie pole *ReplyToQMgr* jest ustawione na nazwę menedżera kolejek, z którym połączona jest aplikacja.

**Uwaga:** Można zażądać, aby menedżer kolejek podjął więcej niż jedną próbę dostarczenia komunikatu, a w przypadku niepowodzenia można zażądać usunięcia komunikatu. Jeśli komunikat, po niepowodzeniu dostarczenia, nie zostanie odrzucony, menedżer kolejek zdalnych umieszcza komunikat w swojej kolejce niedostarczonych komunikatów (niedostarczonych komunikatów) (patrz sekcja "Korzystanie z kolejki niedostarczonych komunikatów (niedostarczonych komunikatów)" na stronie 1072 ).

## Kontekst komunikatu

Informacje *Kontekst komunikatu* umożliwiają aplikacji, która pobiera komunikat, uzyskanie informacji o twórcy komunikatu.

Pobieranie aplikacji może wymagać:

- Sprawdź, czy aplikacja wysyłająca ma poprawny poziom uprawnień.
- Wykonaj pewną funkcję rozliczeniową, aby aplikacja wysyłająca mogła obciążać aplikację wysyłającą za każdą pracę, którą musi wykonać.

- Zachowaj zapis kontrolny wszystkich komunikatów, z którymi pracował

Jeśli do umieszczenia komunikatu w kolejce używane jest wywołanie MQPUT lub MQPUT1 , można określić, że menedżer kolejek ma dodać do deskryptora komunikatu pewne domyślne informacje o kontekście. Aplikacje, które mają odpowiedni poziom uprawnień, mogą dodawać dodatkowe informacje o kontekście. Więcej informacji na temat określania informacji o kontekście zawiera sekcja "Sterowanie informacjami o kontekście komunikatu" na stronie 784.

Kontekst użytkownika jest używany przez menedżer kolejek podczas generowania następujących typów komunikatów raportu:

- Potwierdź przy dostarczeniu
- Utrata ważności

Po wygenerowaniu tych komunikatów w kontekście użytkownika sprawdzane są uprawnienia + put i + passid w miejscu docelowym raportu. Jeśli kontekst użytkownika ma niewystarczające uprawnienia, komunikat raportu jest umieszczany w kolejce niedostarczonych komunikatów, jeśli został zdefiniowany. Jeśli nie ma kolejki niedostarczonych komunikatów, komunikat raportu jest odrzucany.

Wszystkie informacje o kontekście są przechowywane w polach kontekstu deskryptora komunikatu. Typ informacji odpowiada informacjom o tożsamości, pochodzeniu i kontekście użytkownika.

## kontekst tożsamości

Informacje w polu *Kontekst tożsamości* identyfikują użytkownika aplikacji, który pierwszy umieścił komunikat w kolejce. Odpowiednio autoryzowane aplikacje mogą ustawić następujące pola:

- Menedżer kolejek wypełnia pole *UserIdentifier* nazwą identyfikującą użytkownika. Sposób, w jaki menedżer kolejek może to zrobić, zależy od środowiska, w którym działa aplikacja.
- Menedżer kolejek wypełnia pole *AccountingToken* znacznikiem lub numerem, który został określony na podstawie aplikacji, która umieściła komunikat.
- Aplikacje mogą korzystać z pola *ApplIdentityData* w celu uzyskania dodatkowych informacji o użytkowniku (na przykład zaszyfrowanego hasła).

Identyfikator bezpieczeństwa systemu Windows (SID) jest zapisywany w polu *AccountingToken* podczas tworzenia komunikatu w katalogu IBM MQ for Windows. Identyfikatora SID można użyć do uzupełnienia pola *UserIdentifier* i do ustanowienia referencji użytkownika.

Więcej informacji na temat wypełniania pól *UserIdentifier* i *AccountingToken* przez menedżer kolejek zawierają opisy tych pól w sekcji UserIdentifier i AccountingToken.

Aplikacje, które przekazują komunikaty z jednego menedżera kolejek do innego, powinny również przekazywać informacje o kontekście tożsamości, aby inne aplikacje znały tożsamość nadawcy komunikatu.

## Kontekst źródła

Informacje o *kontekście źródła* opisują aplikację, która umieściła komunikat w kolejce, w której komunikat jest obecnie przechowywany. Deskryptor komunikatu zawiera następujące pola dla informacji o kontekście źródłowym:

- *PutApplType* definiuje typ aplikacji, która umieściła komunikat (na przykład transakcję CICS ).
- *PutApplName* definiuje nazwę aplikacji, która umieściła komunikat (na przykład nazwę zadania lub transakcji).
- *PutDate* : data umieszczenia komunikatu w kolejce.
- *PutTime* definiuje godzinę umieszczenia komunikatu w kolejce.
- *ApplOriginData* definiuje dodatkowe informacje, które aplikacja chce dołączyć na temat pochodzenia komunikatu. Na przykład może zostać ustawiona przez odpowiednio autoryzowane aplikacje w celu wskazania, czy dane tożsamości są zaufane.

Informacje o kontekście źródłowym są zwykle dostarczane przez menedżer kolejek. Czas Greenwich (GMT) jest używany w polach *PutDate* i *PutTime* . Patrz opisy tych pól w sekcji PutDate i PutTime.

Aplikacja z wystarczającymi uprawnieniami może udostępnić własny kontekst. Umożliwia to zachowanie informacji rozliczeniowych, gdy jeden użytkownik ma inny identyfikator użytkownika w każdym systemie, który przetwarza komunikat, z którego pochodzi.

### Obiekty IBM MQ

Te informacje zawierają szczegółowe informacje na temat obiektów IBM MQ , które obejmują: menedżery kolejek, grupy współużytkowania kolejek, kolejki, obiekty tematów administracyjnych, listy nazw, definicje procesów, obiekty informacji uwierzytelniającej, kanały, klasy pamięci masowej, programy nasłuchujące i usługi.

Menedżery kolejek definiują właściwości (nazywane atrybutami) tych obiektów. Wartości tych atrybutów wpływają na sposób przetwarzania tych obiektów przez program IBM MQ . W aplikacjach do sterowania tymi obiektami używany jest interfejs kolejki komunikatów (Message Queue Interface-MQI). Obiekty są identyfikowane przez *deskryptor obiektu* (MQOD), gdy są adresowane z programu.

Jeśli na przykład do definiowania, modyfikowania lub usuwania obiektów używane są komendy systemu IBM MQ , menedżer kolejek sprawdza, czy użytkownik ma poziom uprawnień wymagany do wykonania tych operacji. Podobnie, gdy aplikacja używa wywołania MQOPEN do otwarcia obiektu, menedżer kolejek sprawdza, czy aplikacja ma wymagany poziom uprawnień, zanim zezwoli na dostęp do tego obiektu. Sprawdzana jest nazwa otwieranego obiektu.

**Pojęcia pokrewne**

“Sterowanie informacjami o kontekście komunikatu” na stronie 784
Jeśli do umieszczenia komunikatu w kolejce używane jest wywołanie MQPUT lub MQPUT1 , można określić, że menedżer kolejek ma dodać do deskryptora komunikatu pewne domyślne informacje o kontekście. Aplikacje, które mają odpowiedni poziom uprawnień, mogą dodawać dodatkowe informacje o kontekście. Do sterowania informacjami o kontekście można użyć pola opcji w strukturze MQPMO.

**Odsyłacze pokrewne**

“Opcje MQOPEN dotyczące kontekstu komunikatu” na stronie 774
Aby można było powiązać informacje o kontekście z komunikatem umieszczonym w kolejce, podczas otwierania kolejki należy użyć jednej z opcji kontekstu komunikatu.

## Windows Przygotowywanie i uruchamianie aplikacji produktu Microsoft Transaction Server

Aby przygotować aplikację MTS do działania jako aplikacja IBM MQ MQI client , należy postępować zgodnie z instrukcjami odpowiednimi dla danego środowiska.

Ogólne informacje na temat tworzenia aplikacji Microsoft Transaction Server (MTS) uzyskujących dostęp do zasobów IBM MQ zawiera sekcja dotycząca MTS w Centrum pomocy produktu IBM MQ .

Aby przygotować aplikację MTS do działania jako aplikacja IBM MQ MQI client , wykonaj jedną z następujących czynności dla każdego komponentu aplikacji:

- Jeśli komponent używa powiązań języka C dla interfejsu MQI, należy postępować zgodnie z instrukcjami zawartymi w sekcji “Przygotowywanie programów w języku C w systemie Windows” na stronie 1046 , ale powiązać komponent z biblioteką mqicxa.lib zamiast z biblioteką mqic.lib.
- Jeśli komponent używa klas języka C++ IBM MQ , należy postępować zgodnie z instrukcjami zawartymi w sekcji “Budowanie programów w języku C++ w systemie Windows” na stronie 569 , ale połączyć komponent z biblioteką imqx23vn.lib zamiast z biblioteką imqc23vn.lib.
- Jeśli komponent używa powiązań języka Visual Basic dla interfejsu MQI, należy postępować zgodnie z instrukcjami zawartymi w sekcji “Przygotowywanie programów Visual Basic w systemie Windows” na stronie 1050 , ale podczas definiowania projektu Visual Basic należy wpisać wartość MqType=3 w polu **Argumenty kompilacji warunkowej** .

# Uwagi dotyczące projektowania dla aplikacji IBM MQ

Po podjęciu decyzji, w jaki sposób aplikacje mogą korzystać z dostępnych platform i środowisk, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

Podczas projektowania aplikacji IBM MQ należy wziąć pod uwagę następujące pytania i opcje:

**Rodzaj zastosowania**

Jaki jest cel Twojej aplikacji? Aby uzyskać informacje na temat różnych typów aplikacji, które można tworzyć, należy skorzystać z następujących odsyłaczy:

- Serwer
- Klient
- Publikowanie / subskrypcja
- Usługi WWW
- Wyjścia użytkownika, wyjścia funkcji API i usługi instalowalne

Ponadto można również napisać własne aplikacje, aby zautomatyzować administrowanie produktem IBM MQ. Więcej informacji na ten temat zawierają sekcje Interfejs administracyjny IBM MQ (MQAI) i Automatyzacja zadań administracyjnych.

**Język programowania**

Produkt IBM MQ obsługuje wiele różnych języków programowania na potrzeby pisania aplikacji. Więcej informacji na ten temat zawiera sekcja “Tworzenie aplikacji dla składnika IBM MQ” na stronie 5.

**Aplikacje dla więcej niż jednej platformy**

Czy aplikacja będzie działać na więcej niż jednej platformie? Czy masz strategię przejścia do innej platformy niż ta, z której korzystasz dzisiaj? Jeśli odpowiedź na którekolwiek z tych pytań jest twierdząca, należy upewnić się, że programy zostały napisane w celu zapewnienia niezależności platformy.

Na przykład, jeśli używany jest kod C, kod w standardzie ANSI C. Należy użyć standardowej funkcji biblioteki w języku C, a nie równoważnej funkcji specyficznej dla platformy, nawet jeśli funkcja specyficzna dla platformy jest szybsza lub bardziej wydajna. Wyjątkiem jest sytuacja, w której efektywność w kodzie jest nadrzędna, a kod dla obu sytuacji należy zakodować przy użyciu kodu #ifdef. Na przykład:

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

**Typy kolejek**

Czy chcesz utworzyć kolejkę za każdym razem, gdy jest potrzebna, czy chcesz użyć kolejek, które zostały już skonfigurowane? Czy chcesz usunąć kolejkę po zakończeniu jej używania, czy też będzie ona używana ponownie? Czy chcesz użyć kolejek aliasowych w celu uzyskania niezależności aplikacji? Aby sprawdzić, jakie typy kolejek są obsługiwane, należy zapoznać się z sekcją Kolejki.

z/OS **Korzystanie z kolejek współużytkowanych, grup współużytkowania kolejek i klastrów grup współużytkowania kolejek (tylko system IBM MQ for z/OS )**

Użytkownik może skorzystać ze zwiększonej dostępności, skalowalności i równoważenia obciążenia, które są możliwe, gdy używane są kolejki współużytkowane z grupami współużytkowania kolejek. Więcej informacji na ten temat zawiera sekcja Kolejki współużytkowane i grupy współużytkowania kolejek .

Można również oszacować średnie i szczytowe przepływy komunikatów i rozważyć użycie klastrów grup współużytkowania kolejek w celu rozłożenia obciążenia. Więcej informacji na ten temat zawiera sekcja Kolejki współużytkowane i grupy współużytkowania kolejek .

**Korzystanie z klastrów menedżera kolejek**
Można skorzystać z uproszczonego administrowania systemem oraz większej dostępności, skalowalności i równoważenia obciążenia, które są możliwe w przypadku korzystania z klastrów.

**Typy komunikatów**
Można użyć datagramów dla prostych komunikatów, ale komunikatów żądań (dla których oczekuje się odpowiedzi) dla innych sytuacji. Do niektórych komunikatów można przypisać różne priorytety. Więcej informacji na temat projektowania komunikatów zawiera sekcja "Techniki projektowania komunikatów" na stronie 60.

**Korzystanie z funkcji przesyłania komunikatów w trybie publikowania/subskrypcji lub w trybie punkt z punktem**
Za pomocą przesyłania komunikatów w trybie publikowania/subskrypcji aplikacja wysyłająca wysyła informacje, które mają być współużytkowane w komunikacie IBM MQ , do standardowego miejsca docelowego zarządzanego przez funkcję IBM MQ publish? subscribe i umożliwia IBM MQ obsługę dystrybucji tych informacji. Aplikacja docelowa nie musi wiedzieć nic o źródle informacji, które otrzymuje, po prostu rejestruje zainteresowanie jednym lub większą liczbą tematów i otrzymuje te informacje, gdy są one dostępne. Więcej informacji na temat przesyłania komunikatów w trybie publikowania/subskrypcji zawiera sekcja Przesyłanie komunikatów w trybie publikowania/subskrypcji.

Za pomocą przesyłania komunikatów w trybie punkt z punktem aplikacja wysyłająca wysyła komunikat do konkretnej kolejki, skąd wie, że aplikacja odbierająca go pobierze. Aplikacja odbierająca pobiera komunikaty z konkretnej kolejki i działa na ich zawartości. Aplikacja często działa zarówno jako nadawca, jak i odbiorca, wysyłając zapytanie do innej aplikacji i odbierając odpowiedź.

**Sterowanie programami IBM MQ**
Użytkownik może chcieć uruchomić niektóre programy automatycznie lub spowodować, że programy będą oczekiwać do momentu pojawienia się określonego komunikatu w kolejce (korzystając z opcji IBM MQ *wyzwalania* , patrz sekcja "Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy" na stronie 891 ). Alternatywnie można uruchomić inną instancję aplikacji, gdy komunikaty w kolejce nie są wystarczająco szybko przetwarzane (za pomocą funkcji IBM MQ *instrumentacji zdarzeń* opisanej w sekcji Instrumentacja zdarzeń ).

**Uruchamianie aplikacji na kliencie IBM MQ**
Pełna wersja interfejsu MQI jest obsługiwana w środowisku klienta, a prawie każda aplikacja IBM MQ napisana w języku proceduralnym może zostać ponownie włączona do uruchamiania w systemie IBM MQ MQI client. Powiąż aplikację na serwerze IBM MQ MQI client z biblioteką MQIC, a nie z biblioteką MQI. z/OS Funkcja get (sygnał) w systemie z/OS nie jest obsługiwana.

**Uwaga:** Aplikacja działająca na kliencie IBM MQ może jednocześnie nawiązać połączenie z więcej niż jednym menedżerem kolejek lub użyć nazwy menedżera kolejek z gwiazdką (*) w wywołaniu MQCONN lub MQCONNX. Zmień aplikację, aby utworzyć dowiązanie do bibliotek menedżera kolejek zamiast do bibliotek klienta, ponieważ ta funkcja nie będzie dostępna.

Więcej informacji zawiera sekcja "Uruchamianie aplikacji w środowisku IBM MQ MQI client" na stronie 949.

**Zarządzanie wydajnością**
Decyzje dotyczące projektowania mogą mieć wpływ na wydajność aplikacji, a sugestie dotyczące zwiększania wydajności aplikacji IBM MQ można znaleźć w sekcji "Uwagi dotyczące projektowania aplikacji i wydajności" na stronie 61 IBM i i "Uwagi dotyczące projektowania i wydajności aplikacji IBM i" na stronie 65 .

**Zaawansowane techniki IBM MQ**
W przypadku bardziej zaawansowanych aplikacji można użyć niektórych zaawansowanych technik produktu IBM MQ , takich jak korelowanie odpowiedzi oraz generowanie i wysyłanie informacji o kontekście IBM MQ . Więcej informacji na ten temat zawiera sekcja "Techniki projektowania dla zaawansowanych aplikacji" na stronie 63.

**Zabezpieczanie danych i zachowanie ich integralności**
Informacji o kontekście, które zostały przekazane wraz z komunikatem, można użyć do sprawdzenia, czy komunikat został wysłany z akceptowalnego źródła. Aby upewnić się, że dane pozostają spójne

z innymi zasobami, można użyć narzędzi do synchronizacji udostępnianych przez produkt IBM MQ lub system operacyjny (więcej informacji na ten temat zawiera sekcja “Zatwierdzanie i wycofywanie jednostek pracy” na stronie 879 ). Aby zapewnić dostarczanie ważnych komunikatów, można użyć funkcji *trwałości* komunikatów IBM MQ .

**Testowanie aplikacji IBM MQ**

Środowisko programistyczne aplikacji dla programów IBM MQ nie różni się od innych aplikacji, dlatego można używać tych samych narzędzi programistycznych oraz narzędzi śledzenia IBM MQ .

z/OS Podczas testowania aplikacji CICS za pomocą produktu IBM MQ for z/OSmożna użyć narzędzia CICS Execution Diagnostic Facility (CEDF). CEDF przechwytuje wejście i wyjście każdego wywołania MQI, a także wywołania do wszystkich usług CICS . Ponadto w środowisku CICS można napisać program obsługi wyjścia dla komunikacji między interfejsami API, aby udostępnić informacje diagnostyczne przed każdym wywołaniem MQI i po nim. Więcej informacji na ten temat zawiera sekcja “Używanie i pisanie aplikacji w systemie IBM MQ for z/OS” na stronie 916.

IBM i Podczas testowania aplikacji IBM i można użyć standardowego debugera. W tym celu należy użyć komendy STRDBG.

**Obsługa wyjątków i błędów**

Należy rozważyć sposób przetwarzania komunikatów, których nie można dostarczyć, oraz sposób rozwiązywania problemów zgłoszonych przez menedżer kolejek. W przypadku niektórych raportów należy ustawić opcje raportu w MQPUT.

**Pojęcia pokrewne**

IBM MQ Przegląd techniczny

“Uwagi dotyczące projektowania i wydajności aplikacji z/OS” na stronie 67
Projektowanie aplikacji jest jednym z najważniejszych czynników wpływających na wydajność. Ten temat zawiera informacje o niektórych czynnikach projektowych związanych z wydajnością.

“Tworzenie aplikacji dla składnika IBM MQ” na stronie 5
Użytkownik może tworzyć aplikacje do wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i zasobami pokrewnymi. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i środowiskach.

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć języka proceduralnego lub obiektowego. Przed rozpoczęciem projektowania i pisania aplikacji IBM MQ należy zapoznać się z podstawowymi pojęciami związanymi z produktem IBM MQ .

“Pisanie aplikacji proceduralnej dotyczącej kolejkowania” na stronie 744
Ten temat zawiera informacje o pisaniu aplikacji kolejkowania, nawiązywaniu i rozłączaniu połączeń z menedżerem kolejek, publikowaniu i subskrybowaniu oraz otwieraniu i zamykaniu obiektów.

“Pisanie aplikacji proceduralnych klienta” na stronie 941
Informacje potrzebne do pisania aplikacji klienckich w systemie IBM MQ przy użyciu języka proceduralnego.

“Tworzenie aplikacji .NET” na stronie 573
IBM MQ classes for .NET zezwól aplikacjom .NET na nawiązywanie połączeń z produktem IBM MQ jako IBM MQ MQI client lub nawiązywanie połączeń bezpośrednio z serwerem IBM MQ .

“Tworzenie aplikacji w języku C++” na stronie 545
IBM MQ udostępnia klasy C++ równoważne obiektom IBM MQ i pewne dodatkowe klasy równoważne tablicom typów danych. Udostępnia on wiele funkcji, które nie są dostępne za pośrednictwem interfejsu MQI.

“Korzystanie z IBM MQ classes for JMS/Jakarta Messaging” na stronie 84
IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging są dostawcami przesyłania komunikatów produktu Java dostarczanymi wraz z produktem IBM MQ. Oprócz implementowania interfejsów zdefiniowanych w specyfikacjach JMS i Jakarta Messaging dostawcy przesyłania komunikatów dodają dwa zestawy rozszerzeń do interfejsu API przesyłania komunikatów produktu Java .

“użycieIBM MQ classes for Java” na stronie 357
Użyj IBM MQ w środowisku Java . IBM MQ classes for Java Zezwalaj aplikacji Java na nawiązywanie połączenia z produktem IBM MQ jako klientem IBM MQ lub nawiązywanie połączenia bezpośrednio z menedżerem kolejek produktu IBM MQ .

# Określanie nazwy aplikacji w obsługiwanych językach programowania

Przed wprowadzeniem IBM MQ 9.2.0można było określić nazwę aplikacji w aplikacjach klienckich w systemie Java lub JMS . Począwszy od wersji IBM MQ 9.2.0 ta funkcja jest rozszerzona o inne języki programowania w systemie IBM MQ for Multiplatforms.

## Sposób użycia nazwy aplikacji

Nazwa aplikacji jest wyprowadzana z:

- Komenda runmqsc DISPLAY CONN APPLTAG
- runmqsc DISPLAY QSTATUS TYPE (HANDLE) APPLTAG
- runmqsc DISPLAY CHSTATUS RAPPLTAG
- MQMD.PutApplName
- Śledzenie aktywności aplikacji

Nazwa aplikacji jest również używana podczas konfigurowania śledzenia aktywności aplikacji. Domyślna nazwa aplikacji dla aplikacji innych niż aplikacjeJava jest obciętą nazwą pliku wykonywalnego, z wyjątkiem systemów Windows i IBM i.

Windows W systemie Windowsnazwą domyślną jest pełna nazwa pliku wykonywalnego obcięta do 28 znaków po lewej stronie.

IBM i W systemie IBM inazwą domyślną jest nazwa zadania.

W przypadku aplikacji Java jest to nazwa klasy poprzedzona nazwą pakietu obciętą z lewej strony do 28 znaków.

Więcej informacji na ten temat zawiera sekcja PutApplNazwa.

W systemie IBM MQ 9.2.0aplikacje w systemie IBM MQ for Multiplatforms mogą ustawiać nazwy swoich aplikacji zarówno administracyjnie, jak i przy użyciu różnych metod programowania. Dzięki temu aplikacje mogą udostępniać bardziej znaczącą nazwę niezależną od platformy podczas konfigurowania śledzenia aktywności aplikacji lub podczas wyprowadzania danych z różnych komend **runmqsc** .

W produkcie IBM MQ 9.2.0można zrównoważyć aplikacje w jednolitym klastrze. W tym celu używane są zrozumiałe nazwy aplikacji.

## Obsługiwane znaki

Więcej informacji na temat określania nazwy aplikacji zawiera sekcja “Zalecane znaki nazwy aplikacji” na stronie 55 .

## Języki programowania

Więcej informacji na temat sposobu, w jaki aplikacje tłumaczone na biblioteki IBM MQ w języku C i w innych językach programowania mogą udostępniać nazwę aplikacji, zawiera sekcja “Połączenia języka programowania” na stronie 57 .

## Zarządzane aplikacje .NET

Informacje na temat sposobu, w jaki zarządzane aplikacje .NET mogą udostępniać nazwę aplikacji, zawiera sekcja “Zarządzane aplikacje .NET” na stronie 58 .

### Aplikacje XMS

Informacje na temat sposobu, w jaki aplikacje XMS mogą udostępniać nazwę aplikacji, zawiera sekcja “Aplikacje XMS” na stronie 59 .

### Aplikacje powiązań Java i JMS

ALW

Więcej informacji na temat sposobu, w jaki aplikacje Java i JMS mogą udostępniać nazwę aplikacji, zawiera sekcja “Aplikacje powiązań Java i JMS” na stronie 59 .

**Pojęcia pokrewne**

Śledzenie aktywności aplikacji

Informacje o jednolitych klastrach

**Odsyłacze pokrewne**

MQCNO

MQCNO w systemie IBM i

## Używanie nazwy aplikacji w obsługiwanych językach programowania

Te informacje umożliwiają poznanie sposobu wybierania nazwy aplikacji w różnych językach obsługiwanych przez produkt IBM MQ .

### Zalecane znaki nazwy aplikacji

Nazwy aplikacji muszą znajdować się w zestawie znaków określonym przez atrybut **CodedCharSetId** pola menedżera kolejek. Więcej informacji na temat tego atrybutu zawiera sekcja Atrybuty menedżera kolejek.

Jeśli jednak aplikacja działa jako aplikacja IBM MQ MQI client, nazwa aplikacji musi być w zestawie znaków i kodowaniu klienta.

Aby zapewnić płynne przejście nazwy aplikacji między menedżerami kolejek i umożliwić monitorowanie zasobów aplikacji za pomocą tematów monitorowania zasobów, nazwy aplikacji powinny zawierać tylko drukowalne znaki jednobajtowe.

**Uwagi:**

- Należy również unikać używania ukośników i znaków ampersand w nazwach aplikacji.
- V 9.3.0 Należy unikać używania znaku ampersand w nazwach aplikacji. Wielkości mierzone STATAPP w temacie systemowym dla nazw aplikacji zawierających znak ampersand nie zostaną utworzone.

Ogranicza to nazwę do:

- Znaki alfanumeryczne: `A-Z`, `a-z` i `0-9`

  **Uwaga:** W nazwach aplikacji w systemach używających kodu EBCDIC Katakana nie należy używać małych liter `a-z` .
- Znak spacji
- Drukowalne znaki, które są niezmienne w kodzie EBCDIC: `+ < = > % * ' ( ) , _ - . : ; ?`
- V 9.3.0 Znak/. Podczas subskrybowania pomiarów śledzenia aktywności lub pomiarów tematów systemu STATAPP dla aplikacji, której nazwa zawiera ukośnik, należy zastąpić wszystkie znaki ukośnika znakiem ampersand. Na przykład, aby otrzymywać metryki STATAPP dla aplikacji o nazwie "DEPT1/APPS/STOCKQUOTE", należy zasubskrybować łańcuch tematu "$SYS/MQ/INFO/QMGR/QMBASIC/Monitor/STATAPP/DEPT1&APPS&STOCKQUOTE/INSTANCE". Przykładowe aplikacje amqsact i amqsrua automatycznie przekształcają ukośniki w znaki ampersand podczas tworzenia ich subskrypcji.

## Sposób ustawiania znaków

W poniższej tabeli przedstawiono sposób, w jaki nazwa aplikacji jest wybierana w różnych językach obsługiwanych przez program IBM MQ . Sposób, w jaki nazwa jest wybierana, jest w kolejności pierwszeństwa, najpierw najwyższy.

| | **Powiązania C i klient** | **Powiązania i klient Java** | **Powiązania i klient JMS** | **Zarządzany klient .NET** | **Niezarządzane powiązania i klient .NET** | **Zarządzany klient XMS** | **Niezarządzane powiązania i klient XMS** |
|---|---|---|---|---|---|---|---|
| Przesłonięcie właściwości połączenia | | Java przesłonięcie właściwości połączenia | | .Przesłonięcie właściwości połączenia NET | .Przesłonięcie właściwości połączenia NET | | |
| nadpisana właściwość | | Java przesłonięta właściwość | | .Przesłonięta właściwość NET | .Przesłonięta właściwość NET | | |
| Środowisko MQ | | Java Środowisko MQ | | .NET MQEnvironment | .NET MQEnvironment | | |
| Właściwość fabryki połączeń | | | Właściwość fabryki połączeń | | | Właściwość fabryki połączeń | Właściwość fabryki połączeń |
| JMSAdmin | | | JMSAdmin | | | JMSAdmin | JMSAdmin |
| MQCNO | Opcje nawiązywania połączeń | | | | | | |
| Zmienna środowiskowa | Zmienne środowiskowe | | | | Zmienne środowiskowe | | Zmienne środowiskowe |
| mqclient.ini<br>(Dotyczy tylko połączeń klienckich) | Połączenia kliencki e | | | | Połączenia kliencki e | | Połączenia kliencki e |
| Java nazwa klasy | | Nazwa klasyJava | Nazwa klasyJava | | | | |

| | Powiązania C i klient | Powiązania i klient Java | Powiązania i klient JMS | Zarządzany klient .NET | Niezarządzane powiązania i klient .NET | Zarządzany klient XMS | Niezarządzane powiązania i klient XMS |
|---|---|---|---|---|---|---|---|
| Nazwa domyślna | Nazwa domyślna | | | .Domyślna nazwa NET | .Domyślna nazwa NET | .Domyślna nazwa NET | .Domyślna nazwa NET |

**Uwaga:** Powiązania języka C i kolumna klienta mają również zastosowanie do następujących języków programowania:

- kompilatory
- Assembler
- Visual Basic
- RPG

## Połączenia języka programowania

Aplikacje tłumaczone na biblioteki IBM MQ w języku C i innych językach programowania mogą udostępniać nazwę aplikacji w następujący sposób.

Metody połączenia są wymienione w kolejności wykonywania, począwszy od najwyższego.

Multi **Opcje nawiązywania połączeń**

- ALW MQCNO

  **Uwaga:** z/OS Podczas nawiązywania połączenia z menedżerem kolejek produktu IBM MQ for z/OS nazwę aplikacji można ustawić tylko za pomocą połączeń w trybie klienta lub za pomocą aplikacji IBM MQ classes for JMS lub IBM MQ classes for Java .

- IBM i MQCNO w systemie IBM i

ALW **Zmienne środowiskowe**

Jeśli nie wybrano jeszcze nazwy aplikacji, można użyć zmiennej środowiskowej **MQAPPLNAME** , aby zidentyfikować połączenie z menedżerem kolejek. Na przykład:

```
export MQAPPLNAME=ExampleApplName
```

Należy zauważyć, że używanych jest tylko pierwszych 28 znaków, a znaki te nie mogą być odstępami ani znakami o kodzie zero.

**Uwaga:** Atrybut dotyczy tylko obsługiwanych języków programowania, niezarządzanych połączeń .NETi niezarządzanych połączeń XMS .

ALW **plik konfiguracyjny klienta**

Jeśli nie wybrano jeszcze nazwy aplikacji, a połączenie jest połączeniem klienta, w pliku konfiguracyjnym klienta (na przykład `mqclient.ini`) można podać następujące informacje, aby zidentyfikować połączenie z menedżerem kolejek.

```
Connection:
    ApplName=ExampleApplName
```

**Uwagi:**

1. Używanych jest tylko 28 pierwszych znaków, a znaki te nie mogą być odstępami ani znakami pustymi.
2. Atrybut dotyczy tylko połączeń klienckich w obsługiwanych językach programowania, niezarządzanych połączeniach .NETi niezarządzanych połączeniach XMS .

Więcej informacji na ten temat zawiera plik konfiguracyjny produktu IBM MQ MQI client , `mqclient.ini`.

#### Nazwa domyślna

Jeśli nadal nie wybrano nazwy aplikacji, nadal będzie używana nazwa domyślna, która zawiera tyle ścieżek i nazw plików wykonywalnych, ile jest wyświetlane w systemie operacyjnym. Więcej informacji na ten temat zawiera sekcja PutApplNazwa.

## Zarządzane aplikacje .NET

Zarządzane aplikacje .NET mogą udostępniać nazwę aplikacji w następujący sposób.

Metody połączenia są wymienione w kolejności wykonywania, począwszy od najwyższego.

#### Nadanie właściwości połączenia

Plik przesłaniania szczegółów połączenia można udostępnić aplikacjom w następujący sposób:

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

Plik określony przez `overrideConnectionDetailsFile` zawiera listę właściwości z przedrostkiem `mqj`. Aplikacje muszą zdefiniować właściwość **`mqj.APPNAME`** , w której wartość właściwości **`mqj.APPNAME`** określa nazwę używaną do identyfikowania połączenia z menedżerem kolejek.

Używanych jest tylko 28 pierwszych znaków nazwy. Na przykład:

```
mqj.APPNAME=ExampleApplName
```

#### nadpisana właściwość

Stała **`MQC.APPNAME_PROPERTY`** została zdefiniowana z wartością *APPNAME*. Teraz można przekazać tę właściwość do konstruktora **`MQQueueManager`** , używając tylko pierwszych 28 znaków nazwy. Na przykład:

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

Więcej informacji na ten temat zawiera “Operacje zarządzane i niezarządzane w produkcie .NET” na stronie 673.

#### Środowisko MQ

Właściwość *AppName* jest dodawana do klasy **`MQEnvironment`**i używanych jest tylko pierwszych 28 znaków. Na przykład:

```
MQEnvironment.AppName = "ExampleApplName";
```

#### Nazwa domyślna

Jeśli nazwa aplikacji nie została podana w żaden sposób opisany w poprzednim tekście, nazwa aplikacji jest automatycznie ustawiana jako nazwa pliku wykonywalnego (i w takiej części ścieżki, która będzie pasować).

## Aplikacje XMS

Metody połączenia są wymienione w kolejności wykonywania, począwszy od najwyższego.

### Właściwość fabryki połączeń

Aplikacje XMS mogą udostępniać nazwę aplikacji w fabryce połączeń przy użyciu właściwości **XMSC.WMQ_APPLICATIONNAME** (*XMSC_WMQ_APPNAME*) w podobny sposób jak w przypadku usługi JMS. Można podać do 28 znaków.

Więcej informacji na ten temat zawierają sekcje “XMS .NET tworzenie obiektów administrowanych” na stronie 682 i “Właściwości komunikatu XMS” na stronie 690.

### JMSAdmin

W narzędziach administracyjnych właściwość jest w skrócie nazywana "**APPLICATIONNAME**" lub "**APPNAME**".

## Aplikacje powiązań Java i JMS

Metody połączenia są wymienione w kolejności wykonywania, począwszy od najwyższego.

ALW Aplikacje klienckie Java i JMS mogą już określać nazwę aplikacji, która została rozszerzona w systemie IBM MQ for Multiplatforms o aplikacje powiązań przy użyciu pola MQCNO **ApplName** .

### Nadanie właściwości połączenia

Właściwość **Application name** została dodana do listy właściwości połączenia, które można nadpisać. Więcej informacji na ten temat zawiera sekcja Używanie przesłonięcia właściwości połączenia IBM MQ.

**Ostrzeżenie:** Właściwości połączenia i sposób użycia pliku przesłonięcia właściwości połączenia są takie same dla parametrów IBM MQ classes for Java i .NET.

### nadpisana właściwość

Stała **MQC.APPNAME_PROPERTY** została zdefiniowana z wartością *APPNAME*. Teraz można przekazać tę właściwość do konstruktora **MQQueueManager** , używając tylko pierwszych 28 znaków nazwy. Więcej informacji na ten temat zawiera sekcja Korzystanie z nadpisania właściwości połączenia w produkcie IBM MQ classes for Java.

### Środowisko MQ

Właściwość *AppName* jest dodawana do klasy **MQEnvironment** i używanych jest tylko pierwszych 28 znaków.

Więcej informacji na ten temat zawiera “Konfigurowanie środowiska IBM MQ dla systemu IBM MQ classes for Java” na stronie 386.

### Nazwa klasyJava

Jeśli nazwa aplikacji nie została podana w żaden sposób w poprzedzającym tekście, nazwa aplikacji jest określana na podstawie nazwy klasy głównej.

Więcej informacji na ten temat zawiera “Konfigurowanie środowiska IBM MQ dla systemu IBM MQ classes for Java” na stronie 386.

**Ostrzeżenie:** IBM i W systemie IBM i nie można wysłać zapytania do głównej nazwy klasy, dlatego zamiast niej używany jest parametr IBM MQ client for Java .

### Pojęcia pokrewne

“Konfigurowanie środowiska IBM MQ dla systemu IBM MQ classes for Java” na stronie 386

Aby aplikacja mogła nawiązać połączenie z menedżerem kolejek w trybie klienta, musi ona określić nazwę kanału, nazwę hosta i numer portu.

**Odsyłacze pokrewne**
MQCNO
MQCNO w systemie IBM i

# Techniki projektowania komunikatów

Uwagi dotyczące projektowania komunikatów, w tym uwagi dotyczące selektorów i właściwości komunikatów.

## Zagadnienia do rozważenia na etapie projektowania

Komunikat jest tworzony, gdy do umieszczenia komunikatu w kolejce używane jest wywołanie MQI. Jako dane wejściowe wywołania należy podać pewne informacje sterujące w *deskryptorze komunikatu* (MQMD) oraz dane, które mają zostać wysłane do innego programu. Jednak na etapie projektowania należy wziąć pod uwagę następujące kwestie, ponieważ mają one wpływ na sposób tworzenia wiadomości:

**Typ komunikatu do użycia**
Czy projektujesz prostą aplikację, w której można wysłać komunikat, a następnie nie podejmować dalszych działań? A może pytasz o odpowiedź na pytanie? Jeśli zadasz pytanie, możesz dołączyć do deskryptora komunikatu nazwę kolejki, w której chcesz otrzymać odpowiedź.

Czy chcesz, aby komunikaty żądania i odpowiedzi były synchroniczne? Oznacza to, że ustawiono limit czasu dla odpowiedzi na żądanie, a jeśli odpowiedź nie zostanie otrzymana w tym okresie, zostanie ona potraktowana jako błąd.

Czy też wolisz pracować asynchronicznie, aby procesy nie musiały zależeć od występowania konkretnych zdarzeń, takich jak wspólne sygnały czasowe?

Innym zagadnieniem jest to, czy wszystkie wiadomości znajdują się w jednostce pracy.

**Przypisywanie różnych priorytetów do komunikatów**
Do każdego komunikatu można przypisać wartość priorytetu i zdefiniować kolejkę, tak aby zachowywała ona komunikaty w kolejności ich priorytetu. W takim przypadku, gdy inny program pobiera komunikat z kolejki, zawsze otrzymuje komunikat o najwyższym priorytecie. Jeśli kolejka nie zachowuje swoich komunikatów w kolejności priorytetów, program pobierający komunikaty z kolejki pobierze je w kolejności, w jakiej zostały dodane do kolejki.

Programy mogą również wybrać komunikat przy użyciu identyfikatora, który został przypisany do menedżera kolejek podczas umieszczania komunikatu w kolejce. Można również wygenerować własne identyfikatory dla każdego komunikatu.

**Wpływ restartowania menedżera kolejek na komunikaty**
Menedżer kolejek zachowuje wszystkie komunikaty trwałe, odtwarzając je w razie potrzeby z plików dziennika produktu IBM MQ po zrestartowaniu. Nietrwałe komunikaty i tymczasowe kolejki dynamiczne nie są zachowywane. Wszystkie komunikaty, które nie mają być usuwane, muszą być zdefiniowane jako trwałe podczas ich tworzenia. Podczas zapisywania aplikacji dla systemu IBM MQ for Windows lub IBM MQ w systemach AIX and Linux należy się upewnić, że wiadomo, w jaki sposób system został skonfigurowany pod kątem przydzielania plików dziennika, aby zmniejszyć ryzyko związane z projektowaniem aplikacji, która będzie działać zgodnie z limitami plików dziennika.

z/OS Ponieważ komunikaty w kolejkach współużytkowanych (dostępne tylko w systemie IBM MQ for z/OS ) są przechowywane w narzędziu CF, komunikaty nietrwałe są zachowywane po restarcie menedżera kolejek, dopóki system CF pozostaje dostępny. Jeśli działanie systemu CF nie powiedzie się, komunikaty nietrwałe zostaną utracone.

**Przekazywanie informacji o sobie odbiorcy wiadomości**
Zwykle menedżer kolejek ustawia identyfikator użytkownika, ale odpowiednie autoryzowane aplikacje mogą również ustawić to pole, aby można było dołączyć własny identyfikator użytkownika i inne informacje, których program odbierający może użyć do rozliczania lub zabezpieczania.

**Liczba kolejek odbiorczych**

Multi Jeśli komunikat może wymagać umieszczenia w kilku kolejkach, można go opublikować w temacie lub na liście dystrybucyjnej.

z/OS Jeśli komunikat może wymagać umieszczenia w kilku kolejkach, można go opublikować w temacie.

### Selektory i właściwości komunikatów

Z komunikatami mogą być powiązane metadane obok głównego ładunku komunikatu. Te właściwości komunikatu mogą być przydatne przy podawaniu dodatkowych danych.

Istnieją dwa aspekty tych dodatkowych danych, o których należy wiedzieć:

- Właściwości nie podlegają ochronie AMS ( Advanced Message Security ). Jeśli do ochrony danych mają być używane zabezpieczenia AMS, należy umieścić je w ładunku, a nie we właściwościach komunikatu.
- Właściwości mogą być używane do wybierania komunikatów.

Należy zauważyć, że użycie selektorów powoduje złamanie standardowej konwencji komunikatów metody "pierwszy przyszedł-pierwszy wyszedł". Ponieważ menedżer kolejek jest zoptymalizowany dla tego obciążenia, udostępnianie złożonych selektorów nie jest zalecane ze względu na wydajność. Menedżer kolejek nie przechowuje indeksów właściwości komunikatu, dlatego wyszukiwanie komunikatu musi być wyszukiwaniem liniowym. Im głębsza kolejka, tym bardziej złożony selektor i mniejsze prawdopodobieństwo, że selektor zgodny z komunikatem będzie miał negatywny wpływ na wydajność.

Jeśli wymagany jest wybór złożony, zaleca się filtrowanie komunikatów przy użyciu dowolnej aplikacji lub mechanizmu przetwarzania, takiego jak IBM Integration Bus, do różnych miejsc docelowych. Alternatywnie przydatne może być użycie hierarchii tematów.

**Uwaga:** Produkt IBM MQ classes for Java nie obsługuje użycia selektorów, jeśli mają być używane selektory, należy to zrobić za pośrednictwem interfejsu API języka JMS .

## Uwagi dotyczące projektowania aplikacji i wydajności

Istnieje wiele sposobów, w jaki zły projekt programu może wpłynąć na wydajność. Może to być trudne do wykrycia, ponieważ program może wydawać się dobrze sprawny, ale wpływa na wydajność innych zadań. W tym temacie opisano kilka problemów specyficznych dla programów, które wywołują wywołania IBM MQ .

Oto kilka pomysłów, które pomogą Ci w projektowaniu wydajnych aplikacji:

- Zaprojektuj aplikację w taki sposób, aby przetwarzanie było wykonywane równolegle z czasem myślenia użytkownika:
  - Wyświetl panel i zezwól użytkownikowi na rozpoczęcie wpisywania, gdy aplikacja jest nadal inicjowana.
  - Uzyskaj potrzebne dane równolegle z różnych serwerów.
- Pozostaw otwarte połączenia i kolejki, jeśli mają być ponownie wykorzystywane zamiast wielokrotnego otwierania i zamykania, łączenia i rozłączania.
- Jednak aplikacja serwera, która umieszcza tylko jeden komunikat, powinna używać wywołania MQPUT1.
- Menedżery kolejek są zoptymalizowane pod kątem komunikatów o wielkości od 4 kB do 100 kB. Bardzo duże komunikaty są nieefektywne; prawdopodobnie lepiej jest wysłać 100 komunikatów po 1 MB każdy niż pojedynczy komunikat o wielkości 100 MB. Bardzo małe wiadomości są również nieefektywne. Menedżer kolejek wykonuje taką samą ilość pracy dla komunikatu jednobajtowego, jak dla komunikatu o wielkości 4 kB.
- Komunikaty należy przechowywać w jednostce pracy, tak aby mogły być jednocześnie zatwierdzone lub wycofane.
- Opcji nietrwałej należy użyć dla komunikatów, które nie muszą być odtwarzalne.

- Jeśli konieczne jest wysłanie komunikatu do pewnej liczby kolejek docelowych, należy rozważyć użycie listy dystrybucyjnej.

## Wpływ długości komunikatu

Ilość danych w komunikacie może mieć wpływ na wydajność aplikacji, która przetwarza komunikat. Aby uzyskać najlepszą wydajność aplikacji, należy wysłać tylko podstawowe dane w komunikacie. Na przykład w przypadku żądania obciążenia konta bankowego jedynymi informacjami, które mogą wymagać przekazania z klienta do aplikacji serwera, są numer konta i kwota obciążenia.

## Wpływ trwałości komunikatu

Komunikaty trwałe są zazwyczaj rejestrowane. Rejestrowanie komunikatów zmniejsza wydajność aplikacji, dlatego należy używać trwałych komunikatów tylko dla podstawowych danych. Jeśli dane w komunikacie mogą zostać usunięte w przypadku zatrzymania lub niepowodzenia menedżera kolejek, należy użyć komunikatu nietrwałego.

z/OS Operacje MQPUT i MQGET dla komunikatów trwałych będą blokowane, gdy ilość miejsca w dzienniku odtwarzania będzie niewystarczająca do zarejestrowania operacji. Taki warunek jest wskazywany w protokole zadania menedżera kolejek przez komunikaty CSQJ110E i CSQJ111A. Upewnij się, że procesy monitorowania są uruchomione, aby można było zarządzać takimi warunkami i unikać ich.

## Wyszukiwanie konkretnej wiadomości

Wywołanie MQGET zwykle pobiera pierwszy komunikat z kolejki. Jeśli w deskryptorze komunikatu używane są identyfikatory komunikatu i korelacji (*MsgId* i *CorrelId*), aby określić konkretny komunikat, menedżer kolejek musi przeszukać kolejkę, dopóki nie znajdzie tego komunikatu. Użycie wywołania MQGET w ten sposób wpływa na wydajność aplikacji.

## Kolejki zawierające komunikaty o różnych długościach

Jeśli aplikacja nie może używać komunikatów o stałej długości, należy dynamicznie powiększać i zmniejszać bufory, aby dostosować je do typowej wielkości komunikatu. Jeśli aplikacja wyśle wywołanie MQGET, które nie powiedzie się z powodu zbyt małego buforu, zostanie zwrócona wielkość danych komunikatu. Dodaj kod do aplikacji, aby odpowiednio zmienić wielkość buforu i ponownie wydać wywołanie MQGET.

**Uwaga:** Jeśli atrybut **MaxMsgLength** nie zostanie jawnie ustawiony, jego wartością domyślną jest 4 MB, co może być bardzo nieefektywne, jeśli ma to wpływ na wielkość buforu aplikacji.

## Częstotliwość punktów synchronizacji

Programy, które wywołują bardzo dużą liczbę wywołań MQPUT lub MQGET w punkcie synchronizacji, bez ich zatwierdzania, mogą powodować problemy z wydajnością. Kolejki, których to dotyczy, mogą zapełniać się komunikatami, które są obecnie niedostępne, podczas gdy inne zadania mogą oczekiwać na pobranie tych komunikatów. Ma to wpływ na pamięć masową oraz na wątki powiązane z zadaniami, które próbują pobrać komunikaty.

## Użycie wywołania MQPUT1

Wywołania MQPUT1 należy używać tylko wtedy, gdy istnieje pojedynczy komunikat do umieszczenia w kolejce. Aby umieścić więcej niż jeden komunikat, należy użyć wywołania MQOPEN, po którym następuje seria wywołań MQPUT i pojedyncze wywołanie MQCLOSE.

## Liczba używanych wątków

Windows W systemie IBM MQ for Windowsaplikacja może wymagać dużej liczby wątków. Każdemu procesowi menedżera kolejek przydzielana jest maksymalna dozwolona liczba wątków aplikacji.

Aplikacje mogą używać zbyt wielu wątków. Należy rozważyć, czy aplikacja uwzględnia tę możliwość i czy podejmuje działania w celu zatrzymania lub zgłoszenia tego typu wystąpienia.

### Umieść trwałe komunikaty w punkcie synchronizacji

Komunikaty trwałe powinny zostać umieszczone i umieszczone w punkcie synchronizacji. Jest to spowodowane tym, że podczas pobierania komunikatu trwałego poza punktem synchronizacji, jeśli operacja pobierania nie powiedzie się, nie ma możliwości, aby aplikacja wiedziała, czy komunikat został odebrany z kolejki, czy też nie, a jeśli komunikat został odebrany, to również został utracony. Jeśli podczas pobierania trwałych komunikatów w punkcie synchronizacji wystąpi niepowodzenie, transakcja zostanie wycofana, a komunikat trwały nie zostanie utracony, ponieważ nadal znajduje się w kolejce.

Podobnie podczas umieszczania komunikatów trwałych należy umieścić je w punkcie synchronizacji. Inną przyczyną umieszczania i pobierania komunikatów trwałych w punkcie synchronizacji jest to, że kod komunikatu trwałego w produkcie IBM MQ jest mocno zoptymalizowany pod kątem punktu synchronizacji. Dlatego umieszczanie i pobieranie trwałych komunikatów w punkcie synchronizacji jest szybsze niż umieszczanie i pobieranie trwałych komunikatów poza punktem synchronizacji.

Jeśli aplikacja umieściła trwałe komunikaty poza punktem synchronizacji, menedżer kolejek sprawdza, czy może utworzyć niejawny punkt synchronizacji w imieniu aplikacji. Jeśli menedżer kolejek może to zrobić, włącza umieszczanie w tym punkcie synchronizacji i automatycznie zatwierdza ten punkt. Bardziej szczegółowy opis znajduje się w sekcji “Niejawny punkt synchronizacji na wielu platformach” na stronie 888 .

Jednak umieszczanie i pobieranie nietrwałych komunikatów poza punktem synchronizacji jest szybsze, ponieważ kod nietrwały w produkcie IBM MQ jest zoptymalizowany pod kątem pozostawania poza punktem synchronizacji. Umieszczanie i pobieranie trwałych komunikatów odbywa się z szybkością dysku, ponieważ trwały komunikat jest utrwalany na dysku. Jednak umieszczanie i pobieranie nietrwałych komunikatów odbywa się z szybkością procesora, ponieważ nie ma konieczności zapisu na dysku, nawet jeśli używany jest punkt synchronizacji.

Jeśli aplikacja otrzymuje komunikaty i nie wie z góry, czy są one trwałe, można użyć opcji GMO MQGMO_SYNCPOINT_IF_PERSISTENT.

## Techniki projektowania dla zaawansowanych aplikacji

Podczas projektowania bardziej zaawansowanych aplikacji można rozważyć pewne techniki, takie jak oczekiwanie na komunikaty, korelowanie odpowiedzi, ustawianie i używanie informacji kontekstowych, automatyczne uruchamianie aplikacji, generowanie raportów i usuwanie powinowactwa komunikatów podczas korzystania z technologii klastrowej.

W przypadku prostej aplikacji IBM MQ należy zdecydować, które obiekty IBM MQ mają być używane w aplikacji, oraz jakie typy komunikatów mają być używane. W przypadku bardziej zaawansowanej aplikacji można użyć niektórych technik wprowadzonych w poniższych sekcjach.

### Oczekiwanie na komunikaty

Program obsługujący kolejkę może oczekiwać na komunikaty przez:

- Oczekiwanie na odebrany komunikat lub upłynięcie określonego przedziału czasu (patrz sekcja “Oczekiwanie na komunikaty” na stronie 822 ).
- z/OS Tylko w systemie IBM MQ for z/OS : ustawienie sygnału informującego program o nadejściu komunikatu. Więcej informacji na ten temat zawiera sekcja “Sygnalizacja” na stronie 823.
- Ustanawianie wyjścia wywołania zwrotnego, które ma być sterowane po nadejściu komunikatu; patrz sekcja “Asynchroniczne korzystanie z komunikatów IBM MQ” na stronie 43.
- Wykonywanie okresowych wywołań w kolejce w celu sprawdzenia, czy komunikat został odebrany (*odpytywanie*). Zazwyczaj nie jest to zalecane, ponieważ może mieć wpływ na wydajność.

## Korelowanie odpowiedzi

W aplikacjach IBM MQ , gdy program otrzymuje komunikat, który żąda wykonania pewnej pracy, zwykle wysyła jeden lub więcej komunikatów odpowiedzi do requestera.

Aby ułatwić requesterowi powiązanie tych odpowiedzi z oryginalnym żądaniem, aplikacja może ustawić pole *identyfikatora korelacji* w deskryptorze każdego komunikatu. Następnie programy kopiują identyfikator komunikatu żądania do pola identyfikatora korelacji w swoich komunikatach odpowiedzi.

## Ustawianie i używanie informacji o kontekście

*Informacje kontekstowe* są używane do wiązania komunikatów z użytkownikiem, który je wygenerował, oraz do identyfikowania aplikacji, która wygenerowała komunikat. Takie informacje są przydatne do ochrony, rozliczania, kontroli i określania problemów.

Podczas tworzenia komunikatu można określić opcję, która żąda od menedżera kolejek powiązania domyślnych informacji o kontekście z komunikatem.

Więcej informacji na temat używania i ustawiania informacji o kontekście zawiera sekcja “Kontekst komunikatu” na stronie 48.

## Automatyczne uruchamianie programów IBM MQ

Użyj opcji IBM MQ *wyzwalanie* , aby uruchomić program automatycznie po nadejściu komunikatu do kolejki.

Można ustawić warunki wyzwalacza w kolejce, aby program rozpoczął przetwarzanie tej kolejki:

- Za każdym razem, gdy komunikat pojawia się w kolejce
- Po nadejściu pierwszego komunikatu do kolejki
- Gdy liczba komunikatów w kolejce osiągnie predefiniowaną liczbę

Więcej informacji na temat wyzwalania zawiera sekcja “Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy” na stronie 891. Wyzwalanie jest tylko jednym ze sposobów automatycznego uruchamiania programu. Na przykład można automatycznie uruchomić program na liczniku czasu za pomocą narzędzi innych niżIBM MQ .

Multi W systemie Wiele platformprogram IBM MQ może definiować obiekty usług w celu uruchamiania programów IBM MQ podczas uruchamiania menedżera kolejek. Więcej informacji na ten temat zawiera sekcja Obiekty usług.

## Generowanie raportów IBM MQ

W aplikacji można zażądać następujących raportów:

- Raporty o wyjątkach
- Raporty dotyczące utraty ważności
- Raporty potwierdzenia przy odbiorze (COA)
- Raporty dotyczące potwierdzania dostarczenia (COD)
- Sprawozdania z powiadomień o działaniach pozytywnych (PAN)
- Raporty powiadomień o negatywnym działaniu (NAN)

Zostały one opisane w sekcji “Komunikaty raportu” na stronie 21.

## Klastry i powinowactwa komunikatów

Przed rozpoczęciem używania klastrów z wieloma definicjami dla tej samej kolejki należy sprawdzić, czy istnieją aplikacje, które wymagają wymiany pokrewnych komunikatów.

W obrębie klastra komunikat może być kierowany do dowolnego menedżera kolejek, który udostępnia instancję odpowiedniej kolejki. Dlatego logika aplikacji z powinowactwa komunikatów może być niezadowolona.

Na przykład mogą istnieć dwie aplikacje, które polegają na serii komunikatów przepływających między nimi w postaci pytań i odpowiedzi. Ważne może być, aby wszystkie pytania były wysyłane do tego samego menedżera kolejek i aby wszystkie odpowiedzi były wysyłane z powrotem do innego menedżera kolejek. W takiej sytuacji ważne jest, aby procedura zarządzania obciążeniem nie wysyłała komunikatów do żadnego menedżera kolejek, który właśnie udostępniał instancję odpowiedniej kolejki.

Jeśli to możliwe, usuń powinowactwa. Usunięcie powinowactwa komunikatów zwiększa dostępność i skalowalność aplikacji.

Więcej informacji na ten temat zawiera sekcja Obsługa powinowactwa komunikatów.

# Uwagi dotyczące projektowania i wydajności aplikacji IBM i

Te informacje umożliwiają zrozumienie wpływu projektu aplikacji, wątków i pamięci masowej na wydajność.

Te informacje są podzielone na dwie sekcje:


- "Uwagi dotyczące projektowania aplikacji" na stronie 65
- "Konkretne problemy z wydajnością" na stronie 66


## Uwagi dotyczące projektowania aplikacji

Istnieje wiele sposobów, w jaki zły projekt programu może wpłynąć na wydajność. Problemy te mogą być trudne do wykrycia, ponieważ program może wydawać się dobrze wykonywać, wpływając jednocześnie na wydajność innych zadań. W poniższych sekcjach opisano kilka problemów specyficznych dla programów wywołujące wywołania IBM MQ for IBM i .

Więcej informacji na temat projektowania aplikacji zawiera sekcja "Uwagi dotyczące projektowania dla aplikacji IBM MQ" na stronie 51.

**Wpływ długości komunikatu**
Chociaż produkt IBM MQ for IBM i zezwala, aby komunikaty przechowywał do 100 MB danych, ilość danych w komunikacie wpływa na wydajność aplikacji, która przetwarza komunikat. Aby uzyskać najlepszą wydajność aplikacji, należy wysłać tylko podstawowe dane w wiadomości; na przykład w żądaniu obciążenia konta bankowego jedynymi informacjami, które mogą być przekazywane z klienta do aplikacji serwera, są numer konta i kwota obciążenia.

**Wpływ trwałości komunikatu**
Komunikaty trwałe są kronikowane. Kronikowanie komunikatów zmniejsza wydajność aplikacji, dlatego należy używać trwałych komunikatów tylko dla podstawowych danych. Jeśli dane w komunikacie mogą zostać usunięte w przypadku zatrzymania lub niepowodzenia menedżera kolejek, należy użyć komunikatu nietrwałego.

**Wyszukiwanie konkretnej wiadomości**
Wywołanie MQGET zwykle pobiera pierwszy komunikat z kolejki. Jeśli w deskryptorze komunikatu używane są identyfikatory komunikatu i korelacji (*MsgId* i *CorrelId*), aby określić konkretny komunikat, menedżer kolejek musi przeszukać kolejkę, dopóki nie znajdzie tego komunikatu. Użycie wywołania MQGET w ten sposób wpływa na wydajność aplikacji.

**Kolejki zawierające komunikaty o różnych długościach**
Jeśli komunikaty w kolejce mają różne długości, w celu określenia wielkości komunikatu aplikacja może użyć wywołania MQGET z polem *BufferLength* ustawionym na zero, tak aby mimo niepowodzenia wywołania zwracane były dane komunikatu. Aplikacja może następnie powtórzyć wywołanie, określając identyfikator komunikatu, który zmierzył w pierwszym wywołaniu, oraz bufor o poprawnej wielkości. Jeśli jednak istnieją inne aplikacje obsługujące tę samą kolejkę, może się okazać, że wydajność aplikacji jest zmniejszona, ponieważ drugie wywołanie MQGET spędza czas na wyszukiwanie komunikatu, który inna aplikacja pobrała w czasie między dwoma wywołaniami.

Jeśli aplikacja nie może używać komunikatów o stałej długości, innym rozwiązaniem tego problemu jest użycie wywołania MQINQ w celu znalezienia maksymalnej wielkości komunikatów, które może zaakceptować kolejka, a następnie użycie tej wartości w wywołaniu MQGET. Maksymalna wielkość komunikatów dla kolejki jest przechowywana w atrybucie **MaxMsgLen** kolejki. Ta metoda może używać dużej ilości pamięci masowej, ponieważ wartość tego atrybutu kolejki może być wartością maksymalną dozwoloną przez IBM MQ for IBM i, która może być większa niż 2 GB.

**Częstotliwość punktów synchronizacji**
Programy, które wywołują wiele wywołań MQPUT w punkcie synchronizacji, bez ich zatwierdzania, mogą powodować problemy z wydajnością. Kolejki, których to dotyczy, mogą zapełniać się komunikatami, które są obecnie nieużyteczne, podczas gdy inne zadania mogą oczekiwać na pobranie tych komunikatów. Ten problem ma wpływ na pamięć masową, a także na wątki powiązane z zadaniami, które próbują pobrać komunikaty.

**Użycie wywołania MQPUT1**
Wywołania MQPUT1 należy używać tylko wtedy, gdy istnieje pojedynczy komunikat do umieszczenia w kolejce. Aby umieścić więcej niż jeden komunikat, należy użyć wywołania MQOPEN, po którym następuje seria wywołań MQPUT i pojedyncze wywołanie MQCLOSE.

**Liczba używanych wątków**
Aplikacja może wymagać wielu wątków. Każdemu procesowi menedżera kolejek przydzielana jest maksymalna dozwolona liczba wątków. Jeśli niektóre aplikacje są kłopotliwe, może to być spowodowane tym, że używają zbyt wielu wątków. Należy rozważyć, czy aplikacja uwzględnia tę możliwość i czy podejmuje działania w celu zatrzymania lub zgłoszenia tego typu wystąpienia. Maksymalna liczba wątków dozwolonych przez program IBM i wynosi 4 095. Jednak wartością domyślną jest 64. Program IBM MQ udostępnia swoim procesom maksymalnie 63 wątki.

## Konkretne problemy z wydajnością

W tej sekcji opisano problemy związane z pamięcią masową i niską wydajnością.

**Problemy z pamięcią**
Jeśli zostanie wyświetlony komunikat systemowy `CPF0907. Serious storage condition may exist` , oznacza to, że możliwe jest zapełnienie miejsca powiązanego z menedżerami kolejek systemu IBM MQ for IBM i .

**Czy aplikacja lub IBM MQ for IBM i działa powoli?**
Jeśli aplikacja działa powoli, może to oznaczać, że jest w pętli lub oczekuje na zasób, który nie jest dostępny. To powolne działanie może być również spowodowane przez problem z wydajnością. Być może dlatego, że system działa w pobliżu ograniczeń swojej pojemności. Ten typ problemu jest prawdopodobnie najgorszy w godzinach szczytowego obciążenia systemu, zwykle w połowie rano i w połowie popołudnia. (Jeśli sieć obejmuje więcej niż jedną strefę czasową, może się wydawać, że szczytowe obciążenie systemu występuje w innym czasie).

Jeśli okaże się, że obniżenie wydajności nie zależy od ładowania systemu, ale zdarza się czasami, gdy system jest lekko załadowany, prawdopodobnie wina spadnie na źle zaprojektowany program użytkowy. Ten problem może objawiać się jako problem, który występuje tylko wtedy, gdy uzyskiwany jest dostęp do określonych kolejek.

Wartości QTOTJOB i QADLTOTJ są wartościami systemowymi, które warto zbadać.

Następujące objawy mogą wskazywać, że produkt IBM MQ for IBM i działa powoli:

- Jeśli system wolno odpowiada na komendy MQSC.
- Jeśli powtarzające się ekrany głębokości kolejki wskazują, że kolejka jest przetwarzana powoli dla aplikacji, z którą można oczekiwać dużej aktywności kolejki.
- Czy śledzenie IBM MQ jest uruchomione?

## Linux Uwagi dotyczące projektowania dla aplikacji Linux on Power Systems - Little Endian

Ponieważ produkt Linux on Power Systems - Little Endian obsługuje tylko aplikacje 64-bitowe, produkt IBM MQ nie obsługuje aplikacji 32-bitowych.

**Pojęcia pokrewne**

“Uwagi dotyczące projektowania dla aplikacji IBM MQ” na stronie 51

Po podjęciu decyzji, w jaki sposób aplikacje mogą korzystać z dostępnych platform i środowisk, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

## z/OS Uwagi dotyczące projektowania i wydajności aplikacji z/OS

Projektowanie aplikacji jest jednym z najważniejszych czynników wpływających na wydajność. Ten temat zawiera informacje o niektórych czynnikach projektowych związanych z wydajnością.

Istnieje wiele sposobów, w jaki zły projekt programu może wpłynąć na wydajność. Problemy te mogą być trudne do wykrycia, ponieważ program może wydawać się dobrze wykonywać, wpływając jednocześnie na wydajność innych zadań. W poniższych sekcjach przedstawiono kilka problemów specyficznych dla programów realizujących wywołania MQI.

Więcej informacji na temat projektowania aplikacji zawiera sekcja “Uwagi dotyczące projektowania dla aplikacji IBM MQ” na stronie 51.

### Wpływ długości komunikatu

Chociaż produkt IBM MQ for z/OS zezwala, aby komunikaty przechowywał do 100 MB danych, ilość danych w komunikacie wpływa na wydajność aplikacji, która przetwarza komunikat. Aby uzyskać najlepszą wydajność aplikacji, należy wysłać tylko podstawowe dane w komunikacie. Na przykład w żądaniu obciążenia konta bankowego jedynymi informacjami, które mogą wymagać przekazania z klienta do aplikacji serwera, są numer konta i kwota obciążenia.

### Wpływ trwałości komunikatu

Komunikaty trwałe są rejestrowane. Rejestrowanie komunikatów zmniejsza wydajność aplikacji, dlatego należy używać trwałych komunikatów tylko dla podstawowych danych. Jeśli dane w komunikacie mogą zostać usunięte w przypadku zatrzymania lub niepowodzenia menedżera kolejek, należy użyć komunikatu nietrwałego.

Dane dla trwałych komunikatów są zapisywane w buforach dziennika. Te bufory są zapisywane w zestawach danych dziennika, gdy:

- Następuje zatwierdzenie
- Komunikat został odebrany lub umieszczony poza punktem synchronizacji
- Bufory WRTHRSH są zapełnione

Przetwarzanie wielu komunikatów w jednej jednostce pracy może spowodować mniejszą liczbę operacji wejścia/wyjścia niż w przypadku przetwarzania komunikatów po jednym dla każdej jednostki pracy lub poza punktem synchronizacji.

### Wyszukiwanie konkretnej wiadomości

Wywołanie MQGET zwykle pobiera pierwszy komunikat z kolejki. Jeśli używane są identyfikatory komunikatu i korelacji ( **MsgId** i **CorrelId** ) w deskryptorze komunikatu w celu określenia konkretnego komunikatu, menedżer kolejek przeszukuje kolejkę, dopóki nie znajdzie tego komunikatu. Użycie komendy MQGET w ten sposób wpływa na wydajność aplikacji, ponieważ w celu znalezienia konkretnego komunikatu może być konieczne skanowanie całej kolejki przez program IBM MQ .

Za pomocą atrybutu kolejki **IndexType** można określić, że menedżer kolejek ma obsługiwać indeks, który może być używany w celu zwiększenia szybkości operacji MQGET w kolejce. Istnieje jednak niewielka redukcja wydajności na potrzeby obsługi indeksu, dlatego należy wygenerować indeks tylko wtedy, gdy jest on potrzebny. Można wybrać opcję budowania indeksu identyfikatorów komunikatów lub identyfikatorów korelacji lub opcję niebudowania indeksu dla kolejek, w których komunikaty są pobierane sekwencyjnie. Spróbuj mieć wiele różnych wartości klucza, a nie wiele o tej samej wartości. Na przykład Balance1, Balance2i Balance3, a nie trzy z Saldem. W przypadku kolejek współużytkowanych wymagany jest poprawny plik **IndexType**. Szczegółowe informacje na temat atrybutu kolejki **IndexType** zawiera sekcja IndexType.

Aby uniknąć wpływu na czas restartu menedżera kolejek za pomocą kolejek poindeksowanych, należy użyć parametru QINDXBLD (NOWAIT) w makrze CSQ6SYSP . Pozwala to na zakończenie restartu menedżera kolejek bez oczekiwania na zakończenie budowania indeksu kolejki.

Pełny opis atrybutu **IndexType** i innych atrybutów obiektu zawiera sekcja Atrybuty obiektów.

## Kolejki zawierające komunikaty o różnych długościach

Pobierz komunikat, używając wielkości buforu zgodnej z oczekiwaną wielkością komunikatu. Jeśli zostanie wyświetlony kod powrotu wskazujący, że komunikat jest zbyt długi, należy uzyskać większy bufor. Jeśli operacja pobierania nie powiedzie się w ten sposób, zwracana długość danych jest wielkością nieprzekształconych danych komunikatu. Jeśli w wywołaniu MQGET zostanie określona opcja MQGMO_CONVERT, a dane rozszerzają się podczas konwersji, mogą one nadal nie zmieścić się w buforze. W takim przypadku konieczne jest dalsze zwiększenie wielkości buforu.

Jeśli komenda MQGET zostanie wydana z długością buforu równą zero, zwróci ona wielkość komunikatu, a aplikacja będzie mogła uzyskać bufor o tej wielkości i ponownie wydać komendę get. Jeśli kolejka jest przetwarzana przez wiele aplikacji, inna aplikacja mogła już przetworzyć komunikat, gdy oryginalna aplikacja ponownie wysłała komendę pobierania. W przypadku sporadycznie dużych komunikatów może być konieczne uzyskanie dużego buforu tylko dla tych komunikatów i zwolnienie go po przetworzeniu komunikatu. Pomoże to zmniejszyć problemy z pamięcią wirtualną, jeśli wszystkie aplikacje mają duże bufory.

Jeśli aplikacja nie może używać komunikatów o stałej długości, innym rozwiązaniem tego problemu jest użycie wywołania MQINQ w celu znalezienia maksymalnej wielkości komunikatów, które może zaakceptować kolejka, a następnie użycie tej wartości w wywołaniu MQGET . Maksymalna wielkość komunikatów dla kolejki jest przechowywana w atrybucie **MaxMsgL** kolejki. Ta metoda może użyć dużej ilości pamięci masowej, ponieważ wartość parametru **MaxMsgL** może wynosić nawet 100 MB, co jest wartością maksymalną dozwoloną przez system IBM MQ for z/OS.

**Uwaga:** Parametr **MaxMsgL** można obniżyć po umieszczeniu w kolejce dużych komunikatów. Na przykład można umieścić komunikat o wielkości 100 MB, a następnie ustawić wartość **MaxMsgL** na 50 bajtów. Oznacza to, że nadal możliwe jest uzyskanie komunikatów większych niż oczekiwane przez aplikację.

## Częstotliwość punktów synchronizacji

Programy wywołujące wiele wywołań MQPUT w punkcie synchronizacji bez ich zatwierdzania mogą powodować problemy z wydajnością. Kolejki, których to dotyczy, mogą zapełniać się komunikatami, które są obecnie nieużyteczne, podczas gdy inne zadania mogą oczekiwać na pobranie tych komunikatów. Ma to wpływ na pamięć masową oraz na wątki powiązane z zadaniami, które próbują pobrać komunikaty.

Co do zasady, jeśli kolejka jest przetwarzana przez wiele aplikacji, zwykle osiągane są najlepsze wyniki, gdy:

- 100 krótkich wiadomości (mniej niż 1 kB), lub
- Jeden komunikat dla większych komunikatów (100 kB)

dla każdego punktu synchronizacji. Jeśli kolejka jest przetwarzana tylko przez jedną aplikację, dla każdej jednostki pracy musi istnieć więcej komunikatów.

Za pomocą atrybutu menedżera kolejek **MAXUMSGS** można ograniczyć liczbę komunikatów, które zadanie może pobrać lub umieścić w pojedynczej jednostce odzyskiwania. Więcej informacji na temat tego atrybutu zawiera opis komendy **ALTER QMGR** w sekcji Komendy MQSC.

## Zalety wywołania MQPUT1

Wywołania MQPUT1 należy używać tylko wtedy, gdy istnieje pojedynczy komunikat do umieszczenia w kolejce. Aby umieścić więcej niż jeden komunikat, należy użyć wywołania MQOPEN , po którym następuje seria wywołań MQPUT i pojedyncze wywołanie MQCLOSE .

## Liczba komunikatów, które może zawierać menedżer kolejek

### Kolejki lokalne

Liczba lokalnych komunikatów, które menedżer kolejek może przechowywać, jest w zasadzie wielkością zestawów stron. Można mieć do 100 zestawów stron (chociaż zalecany jest zestaw stron 0, a zestaw stron 1 jest przeznaczony dla obiektów i kolejek związanych z systemem). Można użyć zestawu stron w formacie rozszerzonym i zwiększyć wielkość zestawu stron.

### Kolejki współużytkowane

Pojemność kolejek współużytkowanych zależy od wielkości narzędzia CF. W produkcie IBM MQ używane są struktury list systemów CF, w których podstawowymi jednostkami pamięci są pozycje i elementy. Każdy komunikat jest przechowywany jako 1 pozycja i wiele elementów zawierających powiązane dane MQMD i inne dane komunikatu. Liczba elementów używanych przez pojedynczy komunikat zależy od wielkości komunikatu, a w przypadku CFLEVEL (5)-od reguł przenoszenia obowiązujących w czasie wykonywania operacji MQPUT. Gdy dane komunikatu są przenoszone do systemu Db2 lub SMDS, wymagana jest mniejsza liczba elementów. Dostęp do danych komunikatu jest wolniejszy, gdy komunikat jest przesyłany. Więcej informacji na temat porównania wydajności i narzutu procesora związanego z przenoszeniem komunikatów można znaleźć w podręczniku Performance Supportpac MP1H .

## Co wpływa na wydajność

Wydajność może oznaczać, jak szybko mogą być przetwarzane komunikaty, a także ile czasu procesora potrzeba na jeden komunikat.

### Wpływ na szybkość przetwarzania komunikatów

W przypadku trwałych komunikatów największy wpływ ma szybkość zestawów danych dziennika. Szybkość zestawów danych dziennika zależy od DASD, na którym są one włączone. Dlatego należy zachować ostrożność, aby umieścić zestaw danych dziennika na woluminach o niskim użyciu w celu zmniejszenia rywalizacji. Rozsiewanie dzienników produktu MQ poprawia wydajność dziennika, gdy na jeden we/wy zapisywanych jest wiele stron. Z Połączenie światłowodowe o wysokiej wydajności (zHPF) ma również znaczącą wydajność w zakresie czasu odpowiedzi operacji we/wy, gdy podsystem we/wy jest zajęty.

Jeśli istnieje żądanie pobrania i umieszczenia komunikatu, dostęp do kolejki jest blokowany podczas żądania w celu zachowania integralności kolejki. Na potrzeby planowania należy rozważyć użycie kolejki zablokowanej dla całego żądania. Jeśli więc czas wykonania operacji put wynosi 100 mikrosekund, a liczba żądań na sekundę przekracza 10 000, mogą wystąpić opóźnienia. Można osiągnąć lepsze niż to w praktyce, ale jest to dobra zasada ogólna. W celu zwiększenia wydajności można użyć różnych kolejek.

Możliwe przyczyny tego mogą być następujące:

- używać wspólnej kolejki odpowiedzi, z której korzysta każda transakcja CICS
- każda transakcja CICS otrzymuje unikalną odpowiedź do kolejki

- Ta kolejka jest używana przez odpowiedź do kolejki dla regionu CICS i wszystkich transakcji w regionie CICS .

Odpowiedź zależy od liczby żądań na sekundę i czasu odpowiedzi żądań.

Jeśli komunikaty muszą być odczytywane z zestawu stron, będą wolniejsze w porównaniu do sytuacji, gdy komunikaty znajdują się w puli buforów. Jeśli istnieje więcej komunikatów niż mieści się w puli buforów, zostaną one wysłane na dysk. Dlatego należy upewnić się, że pula buforów jest wystarczająco duża dla komunikatów o krótkim czasie życia. Jeśli istnieją komunikaty, które są przetwarzane wiele godzin później, prawdopodobnie zostaną one umieszczone na dysku, dlatego należy oczekiwać, że operacja get dla tych komunikatów będzie wolniejsza niż w przypadku, gdy znajdują się one w puli buforów.

W przypadku kolejki współużytkowanej szybkość komunikatów zależy od szybkości narzędzia CF. System CF w procesorze fizycznym jest prawdopodobnie szybszy niż zewnętrzny system CF. Czas odpowiedzi systemu CF zależy od zajętości systemu CF. Na przykład w systemach Hursley, gdy system CF był 17% zajęty, czas odpowiedzi wynosił 14 mikrosekund. Gdy system CF był zajęty w 95%, czas odpowiedzi wynosił 45 mikrosekund.

Jeśli żądania produktu MQ zużywają dużo procesora, może to wpłynąć na szybkość przetwarzania komunikatów. Ponieważ jeśli partycja logiczna (LPAR) jest ograniczona dla procesora, aplikacje będą opóźniać oczekiwanie na procesor.

**Ile procesora na komunikat**

Na ogół większe komunikaty zużywają więcej procesora, dlatego należy unikać dużych (x MB) komunikatów, jeśli to możliwe.

Podczas pobierania konkretnych komunikatów z kolejek należy poindeksować kolejkę, aby menedżer kolejek mógł przejść bezpośrednio do komunikatu (co pozwala uniknąć potencjalnie całego skanowania kolejki). Jeśli kolejka nie jest poindeksowana, to jest skanowana od początku w poszukiwaniu komunikatu. Jeśli w kolejce znajduje się 1000 komunikatów, może być konieczne skanowanie wszystkich 1000 komunikatów. W wyniku tego wykorzystanie procesora jest dużo niepotrzebne.

Kanały używające protokołu TLS mają dodatkowy koszt związany z szyfrowaniem komunikatu.

W produkcie MQ V7 można wybierać komunikaty za pomocą łańcucha selektora oprócz **CORRELID** lub **MSGID**. Każdy komunikat musi zostać sprawdzony, więc jeśli w kolejce znajduje się wiele komunikatów, jest to kosztowne.

Bardziej wydajne jest, aby aplikacja wykonała OPEN PUT PUT CLOSE niż PUT1 PUT1.

**Wyzwalanie w programie CICS**

Jeśli szybkość odbierania komunikatów dla wyzwalanej kolejki jest niska, efektywne jest użycie wyzwalacza jako pierwszego. Jeśli szybkość odbierania komunikatów jest większa niż 10 komunikatów na sekundę, bardziej wydajne jest wyzwolenie pierwszej transakcji, następnie przetworzenie komunikatu przez transakcję i pobranie następnego komunikatu itd. Jeśli komunikat nie nadeszła w krótkim czasie (np. w okresie od 0.1 do 1 sekundy), transakcja zostanie zakończona. W przypadku dużej przepustowości może być konieczne uruchomienie wielu transakcji w celu przetworzenia komunikatów i zapobieżenia tworzeniu się komunikatów. Dla każdego utworzonego komunikatu wyzwalacza wymagane jest umieszczenie i pobranie komunikatu wyzwalacza, co w efekcie podwaja koszt komunikatu.

**Liczba obsługiwanych połączeń lub jednocześnie pracujących użytkowników**

Każde połączenie używa pamięci wirtualnej w obrębie menedżera kolejek, dlatego im więcej jednocześnie pracujących użytkowników, tym większa ilość używanej pamięci. Jeśli potrzebna jest bardzo duża pula buforów i duża liczba użytkowników, może być konieczne ograniczenie wirtualnej pamięci masowej i zmniejszenie wielkości pul buforów.

Jeśli używane są zabezpieczenia, menedżer kolejek buforuje informacje w menedżerze kolejek przez długi czas. Ma to wpływ na ilość pamięci wirtualnej używanej w menedżerze kolejek.

**CHINIT** może obsłużyć do około 10 000 połączeń. Jest to ograniczone przez wirtualną pamięć masową. Jeśli połączenie używa więcej pamięci masowej, na przykład za pomocą protokołu TLS, pamięć masowa na połączenie zwiększa się, co oznacza, że **CHINIT** może obsługiwać mniej połączeń. W przypadku przetwarzania dużych komunikatów będą one wymagać większej ilości pamięci masowej dla buforów w **CHINIT**, aby serwer **CHINIT** mógł obsłużyć mniejszą liczbę komunikatów.

Połączenia ze zdalnym menedżerem kolejek są bardziej wydajne niż połączenia klienta. Na przykład każde żądanie klienta MQ wymaga dwóch przepływów sieciowych (jednego dla żądania i jednego dla odpowiedzi). W przypadku kanału do zdalnego menedżera kolejek może istnieć 50 operacji wysyłania przez sieć, zanim zostanie odebrana odpowiedź. Jeśli rozważana jest duża sieć kliencka, bardziej wydajne może być użycie menedżera kolejek koncentratora w rozproszonym pudełku i umieszczenie jednego kanału w koncentratorze i z koncentratora.

### Inne kwestie wpływające na wydajność

Wielkość zestawu danych dziennika powinna wynosić co najmniej 1000 cylindrów. Jeśli dzienniki są mniejsze od tego, działanie punktu kontrolnego może być zbyt częste. W przypadku zajętego systemu punkt kontrolny zwykle powinien być co 15 minut lub dłuższy, przy bardzo dużej przepustowości może być mniejszy niż ten. Gdy wystąpi punkt kontrolny, pule buforów są skanowane, a na dysk zapisywane są "stare" komunikaty i zmienione strony. Jeśli punkty kontrolne są zbyt częste, może to mieć wpływ na wydajność. Wartość parametru LOGLOAD może również wpływać na częstotliwość punktów kontrolnych. W przypadku nieprawidłowego zakończenia działania menedżera kolejek podczas restartu może być konieczne odczytanie do 3 punktów kontrolnych. Najlepszym odstępem czasu punktu kontrolnego jest równowaga między działaniami podejmowanymi podczas wykonywania punktu kontrolnego i ilością danych dziennika, które mogą wymagać odczytu podczas restartowania menedżera kolejek.

Podczas uruchamiania kanału występuje znaczny narzut. Zwykle lepiej jest uruchomić kanał i pozostawić go podłączony, niż często uruchamiać i zatrzymywać kanał.

**Informacje pokrewne**

MP1K: IBM MQ for z/OS 9.0 Raport dotyczący wydajności

## z/OS Aplikacje mostu IMS i IMS w systemie IBM MQ for z/OS

Informacje te są pomocne podczas pisania aplikacji IMS przy użyciu produktu IBM MQ.

- Informacje na temat używania punktów synchronizacji i wywołań MQI w aplikacjach IMS zawiera sekcja “Pisanie aplikacji IMS przy użyciu programu IBM MQ” na stronie 72.
- Informacje na temat pisania aplikacji używających mostu IBM MQ - IMS zawiera sekcja “Pisanie aplikacji mostu IMS” na stronie 76.

Aby uzyskać więcej informacji na temat aplikacji mostu IMS i IMS w systemie IBM MQ for z/OS, należy skorzystać z następujących odsyłaczy:

- “Pisanie aplikacji IMS przy użyciu programu IBM MQ” na stronie 72
- “Pisanie aplikacji mostu IMS” na stronie 76

**Pojęcia pokrewne**

“Przegląd interfejsu kolejki komunikatów” na stronie 745
Sekcja zawiera informacje na temat komponentów interfejsu kolejki komunikatów (Message Queue Interface-MQI).

“Nawiązywanie i rozłączanie połączenia z menedżerem kolejek” na stronie 758
Aby można było używać usług programistycznych IBM MQ , program musi mieć połączenie z menedżerem kolejek. Ten temat zawiera informacje o nawiązywaniu połączenia z menedżerem kolejek i rozłączaniu się z nim.

“Otwieranie i zamykanie obiektów” na stronie 766
Informacje te udostępniają wgląd w otwieranie i zamykanie obiektów IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 777

Ta sekcja zawiera informacje na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 792
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Uzyskiwanie informacji o atrybutach obiektu i ustawianie ich” na stronie 876
Atrybuty są właściwościami definiującymi charakterystykę obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 879
W tej sekcji opisano sposób zatwierdzania i wycofywania wszelkich odtwarzalnych operacji pobierania i umieszczania, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy” na stronie 891
Informacje o wyzwalaczach i sposobie uruchamiania aplikacji IBM MQ za pomocą wyzwalaczy.

“Praca z funkcją MQI i klastrami” na stronie 911
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do łączenia w klastry.

“Używanie i pisanie aplikacji w systemie IBM MQ for z/OS” na stronie 916
Aplikacje IBM MQ for z/OS mogą być tworzone z programów działających w wielu różnych środowiskach. Oznacza to, że mogą korzystać z udogodnień dostępnych w więcej niż jednym środowisku.

## z/OS Pisanie aplikacji IMS przy użyciu programu IBM MQ

Podczas korzystania z produktu IBM MQ w aplikacjach systemu IMS należy wziąć pod uwagę, które wywołania interfejsu API produktu MQ mogą być używane, a które mechanizm jest używany na potrzeby punktu synchronizacji.

Aby uzyskać więcej informacji na temat pisania aplikacji IMS w systemie IBM MQ for z/OS, należy skorzystać z następujących odsyłaczy:

- “Punkty synchronizacji w aplikacjach IMS” na stronie 72
- “Wywołania MQI w aplikacjach IMS” na stronie 73

### Ograniczenia

Istnieją ograniczenia dotyczące wywołań funkcji API języka IBM MQ , które mogą być używane przez aplikację używającą adaptera IMS .

Następujące wywołania API IBM MQ nie są obsługiwane w aplikacji używającej adaptera IMS :

- Baza MQCB
- MQCB_FUNKCJA
- Komenda MQCTL

**Pojęcia pokrewne**

“Pisanie aplikacji mostu IMS” na stronie 76
Ten temat zawiera informacje dotyczące pisania aplikacji korzystających z mostu IBM MQ - IMS .

## z/OS *Punkty synchronizacji w aplikacjach IMS*

W aplikacji IMS punkt synchronizacji jest ustanawiany za pomocą wywołań IMS , takich jak GU (get unique) do IOPCB i CHKP (checkpoint).

Aby wycofać wszystkie zmiany od poprzedniego punktu kontrolnego, można użyć wywołania IMS ROLB (wycofanie zmian). Więcej informacji na ten temat zawiera sekcja Wywołanie ROLB w dokumentacji systemu IMS .

Menedżer kolejek jest uczestnikiem protokołu zatwierdzania dwufazowego, a menedżer punktów synchronizacji systemu IMS jest koordynatorem.

Wszystkie otwarte uchwyty są zamykane przez adapter IMS w punkcie synchronizacji (z wyjątkiem środowiska BMP sterowanego wsadowo lub niesterowanego komunikatami). Jest to spowodowane tym, że inny użytkownik może zainicjować następną jednostkę pracy, a sprawdzanie zabezpieczeń produktu

IBM MQ jest wykonywane podczas wykonywania wywołań MQCONN, MQCONNX i MQOPEN, a nie podczas wykonywania wywołań MQPUT lub MQGET.

Jednak w środowisku typu Wait-for-Input (WFI) lub pseudo-Wait-for-Input (PWFI) IMS nie powiadamia programu IBM MQ o zamknięciu uchwytów do momentu pojawienia się następnego komunikatu lub zwrócenia do aplikacji kodu statusu QC. Jeśli aplikacja oczekuje w regionie IMS i dowolny z tych uchwytów należy do kolejek wyzwalanych, wyzwalanie nie nastąpi, ponieważ kolejki są otwarte. Z tego powodu aplikacje działające w środowisku WFI lub PWFI powinny jawnie MQCLOSE uchwyty kolejki przed wykonaniem operacji GU na IOPCB dla następnego komunikatu.

Jeśli aplikacja IMS (BMP lub MPP) wysyła wywołanie MQDISC, otwarte kolejki są zamykane, ale nie jest pobierany niejawny punkt synchronizacji. Jeśli aplikacja zakończy się normalnie, wszystkie otwarte kolejki zostaną zamknięte i nastąpi niejawne zatwierdzenie. Jeśli aplikacja zakończy działanie nieprawidłowo, wszystkie otwarte kolejki zostaną zamknięte i nastąpi niejawne wycofanie.

## z/OS *Wywołania MQI w aplikacjach IMS*

Te informacje umożliwiają zapoznanie się z użyciem wywołań MQI w aplikacjach serwera i aplikacjach Enquiry.

W tej sekcji opisano użycie wywołań MQI w następujących typach aplikacji IMS :

- “Aplikacje serwera” na stronie 73
- “Aplikacje do uzyskiwania informacji” na stronie 75

### Aplikacje serwera

Poniżej przedstawiono schemat modelu aplikacji serwera MQI:

```
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
```

Przykładowy program CSQ4ICB3 przedstawia implementację, w systemie C/370, BMP wykorzystującego ten model. Program najpierw nawiązuje komunikację z programem IMS , a następnie z programem IBM MQ:

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

```
Return
```

Proces inicjowania produktu IMS określa, czy program został wywołany jako BMP sterowany komunikatami, czy jako BMP zorientowany na zadanie wsadowe, oraz steruje odpowiednio połączeniami i uchwytami kolejek menedżera kolejek produktu IBM MQ :

```
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
```

Proces inicjowania programu IBM MQ łączy się z menedżerem kolejek i otwiera kolejki. W komponencie BMP sterowanym komunikatami jest on wywoływany po wykonaniu każdego punktu synchronizacji systemu IMS . W komponencie BMP zorientowanym na zadanie wsadowe jest on wywoływany tylko podczas uruchamiania programu:

```
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
```

Na implementację modelu serwera w MPP ma wpływ fakt, że MPP przetwarza pojedynczą jednostkę pracy na wywołanie. Jest to spowodowane tym, że po wykonaniu punktu synchronizacji (GU) uchwyty połączenia i kolejki są zamykane i dostarczany jest następny komunikat IMS . Ograniczenie to może zostać częściowo usunięte przez jedną z następujących sytuacji:

- **Przetwarzanie wielu komunikatów w pojedynczej jednostce pracy**

Obejmuje to:

- Odczytywanie komunikatu
- Przetwarzanie wymaganych aktualizacji
- Umieszczanie odpowiedzi

w pętli do momentu przetworzenia wszystkich komunikatów lub do momentu przetworzenia ustawionej maksymalnej liczby komunikatów. W tym momencie pobierany jest punkt synchronizacji.

W ten sposób można podejść do tylko niektórych typów aplikacji (na przykład prostej aktualizacji bazy danych lub zapytania). Mimo że komunikaty odpowiedzi MQI mogą być umieszczane z uprawnieniami inicjatora obsługiwanego komunikatu MQI, należy uważnie uwzględnić wpływ aktualizacji zasobów IMS na bezpieczeństwo.

- **Przetwarzanie jednego komunikatu na wywołanie MPP i zapewnienie wielu harmonogramów MPP w celu przetworzenia wszystkich dostępnych komunikatów.**

  Użyj programu monitora wyzwalacza IBM MQ IMS (CSQQTRMN), aby zaplanować transakcję MPP, gdy w kolejce IBM MQ znajdują się komunikaty i nie są obsługiwane żadne aplikacje.

  Jeśli monitor wyzwalacza uruchamia proces MPP, nazwa menedżera kolejek i nazwa kolejki są przekazywane do programu, jak pokazano w następującym wyodrębnieniu kodu COBOL:

```
* 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 CMQTMC2L.
*
* 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 '='.
```

Model serwera, który powinien być długotrwałym zadaniem, jest lepiej obsługiwany w regionie przetwarzania wsadowego, chociaż BMP nie może być wyzwalany przy użyciu CSQQTRMN.

## Aplikacje do uzyskiwania informacji

Typowa aplikacja IBM MQ inicjująca zapytanie lub aktualizację działa w następujący sposób:

- Zgromadź dane od użytkownika
- Umieść jeden lub więcej komunikatów IBM MQ
- Pobierz komunikaty odpowiedzi (może być konieczne oczekiwanie na nie)
- Podaj odpowiedź dla użytkownika

Ponieważ komunikaty umieszczane w kolejkach IBM MQ nie są dostępne dla innych aplikacji IBM MQ , dopóki nie zostaną zatwierdzone, muszą być umieszczane poza punktem synchronizacji lub aplikacja IMS musi być podzielona na dwie transakcje.

Jeśli zapytanie obejmuje umieszczenie pojedynczego komunikatu, można użyć opcji *no syncpoint* . Jeśli jednak zapytanie jest bardziej złożone lub dotyczy aktualizacji zasobów, mogą wystąpić problemy ze spójnością w przypadku wystąpienia awarii i braku użycia punktu synchronizacji.

Aby rozwiązać ten problem, można podzielić transakcje IMS MPP za pomocą wywołań MQI za pomocą przełącznika komunikatów typu program-program (program-to-program message switch). Więcej informacji na ten temat zawiera sekcja *IMS Komunikacja międzysystemowa (ISC)* . Pozwala to na zaimplementowanie programu zapytania w MPP:

```
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 Pisanie aplikacji mostu IMS

Ten temat zawiera informacje dotyczące pisania aplikacji korzystających z mostu IBM MQ - IMS .

Informacje na temat mostu IBM MQ - IMS zawiera sekcja Most IMS.

Aby uzyskać więcej informacji na temat pisania aplikacji mostu IMS w systemie IBM MQ for z/OS, należy skorzystać z następujących odsyłaczy:

- “Sposób postępowania mostu IMS z komunikatami” na stronie 76
- “Pisanie programów transakcyjnych IMS za pośrednictwem programu IBM MQ” na stronie 939

**Pojęcia pokrewne**

“Pisanie aplikacji IMS przy użyciu programu IBM MQ” na stronie 72
Podczas korzystania z produktu IBM MQ w aplikacjach systemu IMS należy wziąć pod uwagę, które wywołania interfejsu API produktu MQ mogą być używane, a które mechanizm jest używany na potrzeby punktu synchronizacji.

### z/OS *Sposób postępowania mostu IMS z komunikatami*

Jeśli do wysyłania komunikatów do aplikacji IMS używany jest most IBM MQ - IMS , należy utworzyć komunikaty w specjalnym formacie.

Należy również umieścić komunikaty w kolejkach systemu IBM MQ , które zostały zdefiniowane za pomocą klasy pamięci określającej grupę XCF i nazwę elementu docelowego systemu IMS . Są one nazywane kolejkami mostów MQ-IMS lub po prostu kolejkami **mostów** .

Most IBM MQ-IMS wymaga wyłącznego dostępu wejścia (MQOO_INPUT_EXCLUSIVE) do kolejki mostu, jeśli jest zdefiniowany z użyciem QSGDISP (QMGR) lub jeśli jest zdefiniowany z użyciem QSGDISP (SHARED) razem z opcją NOSHARE.

Użytkownik nie musi wpisywać się do IMS przed wysłaniem komunikatów do aplikacji IMS . Identyfikator użytkownika w polu *UserIdentifier* struktury MQMD jest używany do sprawdzania zabezpieczeń. Poziom sprawdzania jest określany, gdy program IBM MQ łączy się z systemem IMSi jest opisany w sekcji Kontrola dostępu do aplikacji dla mostu IMS. Umożliwia to zaimplementowanie pseudo-logowania.

Most IBM MQ - IMS akceptuje następujące typy komunikatów:

- Komunikaty zawierające dane transakcji IMS i strukturę MQIIH (opisane w sekcji MQIIH ):

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

**Uwaga:**

1. Nawiasy kwadratowe [] reprezentują opcjonalne wiele segmentów.
2. Ustaw pole *Format* struktury MQMD na wartość MQFMT_IMS, aby użyć struktury MQIIH.

- Komunikaty zawierające dane transakcji IMS , ale bez struktury MQIIH:

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

Program IBM MQ sprawdza poprawność danych komunikatu, aby upewnić się, że suma bajtów LL i długości komunikatu MQIIH (jeśli istnieje) jest równa długości komunikatu.

Gdy most IBM MQ - IMS pobiera komunikaty z kolejek mostu, przetwarza je w następujący sposób:

- Jeśli komunikat zawiera strukturę MQIIH, most sprawdza MQIIH (patrz sekcja MQIIH ), buduje nagłówki OTMA i wysyła komunikat do produktu IMS. Kod transakcji jest określony w komunikacie wejściowym. Jeśli jest to LTERM, IMS odpowiada komunikatem DFS1288E . Jeśli kod transakcji reprezentuje komendę, program IMS wykonuje komendę; w przeciwnym razie komunikat jest umieszczany w kolejce IMS dla transakcji.
- Jeśli komunikat zawiera dane transakcji IMS , ale nie zawiera struktury MQIIH, most IMS przyjmuje następujące założenia:
  - Kod transakcji jest w bajtach od 5 do 12 danych użytkownika
  - Transakcja jest w trybie niekonwersacyjnym
  - Transakcja jest w trybie kontroli transakcji 0 (commit-then-send)
  - Zmienna *Format* w strukturze MQMD jest używana jako zmienna *MFSMapName* (na wejściu)
  - Tryb zabezpieczeń to MQISS_CHECK

  Komunikat odpowiedzi jest również budowany bez struktury MQIIH, pobierając wartość *Format* dla deskryptora MQMD z pliku *MFSMapName* wyjścia IMS .

Most IBM MQ - IMS używa jednego lub dwóch potoków dla każdej kolejki produktu IBM MQ :

- Zsynchronizowany potok Tpipe jest używany dla wszystkich komunikatów korzystających z trybu kontroli transakcji 0 (COMMIT_THEN_SEND) (wyświetlane z wartością SYN w polu statusu komendy TPIPE xxxx klienta IMS /DIS TMEMBER).
- Niezsynchronizowany potok Tpipe jest używany dla wszystkich komunikatów korzystających z trybu kontroli transakcji 1 (SEND_THEN_COMMIT)

Potoki Tpotokowe są tworzone przez produkt IBM MQ , gdy są używane po raz pierwszy. Niezsynchronizowany potok Tpipe istnieje do momentu zrestartowania systemu IMS . Zsynchronizowane potoki istnieją do momentu zimnego uruchomienia serwera IMS . Nie można samodzielnie usunąć tych potoków.

Więcej informacji na temat obsługi komunikatów przez most IBM MQ - IMS zawierają następujące tematy:


- “Odwzorowanie komunikatów IBM MQ na typy transakcji IMS” na stronie 78
- “Jeśli nie można umieścić komunikatu w kolejce IMS” na stronie 78
- “Kody sprzężenia zwrotnego mostu IMS” na stronie 79
- “Pola MQMD w komunikatach z mostu IMS” na stronie 79
- “Pola MQIIH w komunikatach z mostu IMS” na stronie 80
- “Komunikaty odpowiedzi od IMS” na stronie 81
- “Używanie alternatywnych bloków PCB odpowiedzi w transakcjach IMS” na stronie 81

- “Wysyłanie niezamówionych komunikatów z serwisu IMS” na stronie 82
- “Segmentacja komunikatów” na stronie 82
- “Konwersja danych dla komunikatów do i z mostu IMS” na stronie 82

**Pojęcia pokrewne**

“Pisanie programów transakcyjnych IMS za pośrednictwem programu IBM MQ” na stronie 939
Kod wymagany do obsługi transakcji IMS za pośrednictwem produktu IBM MQ zależy od formatu komunikatu wymaganego przez transakcję IMS oraz zakresu zwracanych przez nią odpowiedzi. Istnieje jednak kilka punktów, które należy wziąć pod uwagę, gdy aplikacja obsługuje informacje o formatowaniu ekranu w systemie IMS .

z/OS *Odwzorowanie komunikatów IBM MQ na typy transakcji IMS*

Tabela opisująca odwzorowanie komunikatów IBM MQ na typy transakcji IMS .

*Tabela 4. W jaki sposób komunikaty IBM MQ są odwzorowywane na typy transakcji IMS*

| IBM MQ typ komunikatu | Commit-then-send (tryb 0)-używa zsynchronizowanych potoków IMS | Send-then-commit (tryb 1)-używa niezsynchronizowanych potoków IMS |
|---|---|---|
| Trwałe komunikaty IBM MQ | • Odtwarzalne transakcje o pełnej funkcjonalności<br>• Transakcje nienaprawialne są odrzucane przez IMS | • Transakcje krótkiej ścieżki<br>• Transakcje konwersacyjne<br>• W pełni funkcjonalne transakcje |
| Nietrwałe komunikaty IBM MQ | • Nienaprawialne transakcje z pełnymi funkcjami<br>• Transakcje odtwarzalne są dozwolone w poprawkach IMS V8 i APAR PQ61404 oraz we wszystkich późniejszych wersjach systemu IMS . | • Transakcje krótkiej ścieżki<br>• Transakcje konwersacyjne<br>• W pełni funkcjonalne transakcje |

**Uwaga:** Komendy IMS nie mogą używać trwałych komunikatów IBM MQ w trybie kontroli transakcji 0. Więcej informacji na ten temat zawiera sekcja Tryb zatwierdzania (commitMode) .

z/OS *Jeśli nie można umieścić komunikatu w kolejce IMS*

Sekcja zawiera informacje na temat działań, które należy wykonać, jeśli nie można umieścić komunikatu w kolejce IMS .

Jeśli komunikat nie może zostać umieszczony w kolejce IMS , program IBM MQpodejmuje następujące działanie:

- Jeśli komunikat nie może zostać umieszczony w katalogu IMS z powodu niepoprawnego komunikatu, jest on umieszczany w kolejce niedostarczonych komunikatów, a komunikat jest wysyłany do konsoli systemowej.
- Jeśli komunikat jest poprawny, ale został odrzucony przez program IMS, program IBM MQ wysyła komunikat o błędzie do konsoli systemowej, komunikat zawiera kod rozpoznania IMS , a komunikat IBM MQ jest umieszczany w kolejce niedostarczonych komunikatów. Jeśli kod rozpoznania IMS to 001A, program IMS wysyła do kolejki odpowiedzi komunikat IBM MQ zawierający przyczynę niepowodzenia.

  **Uwaga:** W wymienionych wcześniej okolicznościach, jeśli program IBM MQ nie może umieścić komunikatu w kolejce niedostarczonych komunikatów z jakiegokolwiek powodu, komunikat jest zwracany do źródłowej kolejki IBM MQ . Do konsoli systemowej wysyłany jest komunikat o błędzie, a z tej kolejki nie są wysyłane żadne dalsze komunikaty.

  Aby ponownie wysłać komunikaty, wykonaj ***jedną*** z następujących czynności:

- Zatrzymaj i zrestartuj potoki Tpotoków w programie IMS odpowiadające kolejce.
- Zmień kolejkę na GET (DISABLED) i ponownie na GET (ENABLED)
- Zatrzymaj i zrestartuj produkt IMS lub komponent OTMA.
- Zatrzymaj i zrestartuj podsystem IBM MQ .

- Jeśli komunikat został odrzucony przez program IMS z powodu błędu innego niż komunikat, komunikat IBM MQ jest zwracany do kolejki źródłowej, program IBM MQ zatrzymuje przetwarzanie kolejki, a do konsoli systemowej wysyłany jest komunikat o błędzie.

  Jeśli wymagany jest komunikat raportu o wyjątku, most umieszcza go w kolejce odpowiedzi z uprawnieniami nadawcy. Jeśli nie można umieścić komunikatu w kolejce, komunikat raportu jest umieszczany w kolejce niedostarczonych komunikatów z uprawnieniami mostu. Jeśli nie można go umieścić w DLQ, jest on odrzucany.

z/OS *Kody sprzężenia zwrotnego mostu IMS*

Kody rozpoznania IMS są zwykle wyprowadzane w formacie szesnastkowym w komunikatach konsoli IBM MQ , takich jak CSQ2001I (na przykład kod rozpoznania 0x001F). Kody informacji zwrotnych IBM MQ widoczne w nagłówku niedostarczonych komunikatów w kolejce niedostarczonych komunikatów są liczbami dziesiętnymi.

Kody sprzężenia zwrotnego mostu IMS należą do zakresu od 301 do 399 lub od 600 do 855 dla kodu rozpoznania NACK 0x001A. Są one odwzorowywane z kodów rozpoznania IMS-OTMA w następujący sposób:

1. Kod rozpoznania IMS-OTMA jest przekształcany z liczby szesnastkowej na liczbę dziesiętną.
2. 300 jest dodawana do liczby uzyskanej w wyniku obliczenia w 1, co daje kod IBM MQ *Feedback* .
3. Specjalny przypadek to IMS-OTMA sense code 0x001A, dziesiętne 26. Generowany jest kod *Feedback* w zakresie od 600 do 855.
   a. Kod przyczyny IMS-OTMA jest przekształcany z liczby szesnastkowej na liczbę dziesiętną.
   b. Wartość 600 jest dodawana do liczby uzyskanej w wyniku obliczenia w a, co daje kod IBM MQ *Feedback* (Opinia).

Informacje na temat kodów rozpoznania IMS-OTMA zawiera sekcja Kody rozpoznania OTMA dla komunikatów NAK.

z/OS *Pola MQMD w komunikatach z mostu IMS*

Informacje o polach MQMD w komunikatach z mostu IMS .

Deskryptor MQMD komunikatu źródłowego jest przenoszony przez IMS w sekcji User Data (Dane użytkownika) nagłówków OTMA. Jeśli komunikat pochodzi z pliku IMS, jest on budowany przez wyjście rozstrzygania miejsca docelowego produktu IMS . Struktura MQMD komunikatu odebranego z IMS jest zbudowana w następujący sposób:

**StrucID**
"MD"

**Wersja**
MQMD_VERSION_1

**Raport**
MQRO_BRAK

**MsgType**
MQMT_REPLY

**Utrata ważności**
Jeśli opcja MQIIH_PASS_EXPIRATION jest ustawiona w polu Flags nagłówka MQIIH, to pole zawiera pozostały czas utraty ważności, w przeciwnym razie jest ustawione na wartość MQEI_UNLIMITED.

**Opinie**
MQFB_BRAK

**Kodowanie**
MQENC.Native (kodowanie systemu z/OS )

**CodedCharSetId**
MQCCSI_Q_MGR ( CodedCharSetID systemu z/OS )

**Format**
MQFMT_IMS, jeśli MQMD.Format komunikatu wejściowego to MQFMT_IMS, w przeciwnym razie IOPCB.MODNAME

**Priorytet**
MQMD.Priority komunikatu wejściowego

**Trwałość**
Zależy od trybu zatwierdzania: MQMD.Persistence komunikatu wejściowego, jeśli trwałość CM-1; jest zgodna z odtwarzalnością komunikatu IMS , jeśli CM-0

**MsgId**
MQMD.MsgId jeśli MQRO_PASS_MSG_ID, w przeciwnym razie New MsgId (wartość domyślna)

**CorrelId**
MQMD.CorrelId z komunikatu wejściowego, jeśli MQRO_PASS_CORREL_ID, w przeciwnym razie MQMD.MsgId z komunikatu wejściowego (wartość domyślna)

**BackoutCount**
0

**ReplyToQ**
Puste

**ReplyToQMgr**
Odstępy (ustawione na nazwę lokalnego menedżera kolejek przez menedżer kolejek podczas operacji MQPUT)

**UserIdentifier**
MQMD.UserIdentifier komunikatu wejściowego

**AccountingToken**
MQMD.AccountingToken komunikatu wejściowego

**Dane_tożsamości_aplikacji**
MQMD.ApplIdentityData komunikatu wejściowego

**Typ_aplikacji_wstawiającej**
MQAT_XCF, jeśli nie ma błędu, w przeciwnym razie MQAT_BRIDGE

**Nazwa_aplikacji_wstawiającej**
<XCFgroupName> <XCFmemberName>, jeśli nie wystąpił błąd, w przeciwnym razie nazwa QMGR

**PutDate**
Data umieszczenia komunikatu

**PutTime**
Czas umieszczenia komunikatu

**Dane_pochodzenia_aplikacji**
Puste

z/OS *Pola MQIIH w komunikatach z mostu IMS*

Informacje o polach MQIIH w komunikatach z mostu IMS .

Komunikat MQIIH odebrany z produktu IMS jest budowany w następujący sposób:

**StrucId**
"IIH"

**Wersja**
1

**StrucLength**
84

**Kodowanie**
RODZIMA MQENC

**CodedCharSetId**
MQCCSI_Q_MGR (menedżer kolejek MQ)

**Format**
MQIIH.ReplyToFormat komunikatu wejściowego, jeśli MQIIH.ReplyToFormat nie jest pusty, w przeciwnym razie IOPCB.MODNAME

**Flagi**
0

**LTermOverride**
Nazwa LTERM (Tpipe) z nagłówka OTMA

**MFSMapName**
Nazwa mapy z nagłówka OTMA

**Format ReplyTo**
Puste

**Authenticator**
MQIIH.Authenticator komunikatu wejściowego, jeśli komunikat odpowiedzi jest umieszczany w kolejce mostu MQ-IMS , w przeciwnym razie jest pusty.

**Identyfikator TranInstance**
Identyfikator konwersacji/znacznik serwera z nagłówka OTMA, jeśli w konwersacji. W wersjach systemu IMS wcześniejszych niż V14to pole ma zawsze wartość null, jeśli nie jest używane w konwersacji. Począwszy od wersji IMS V14 , to pole może być ustawiane przez system IMS , nawet jeśli nie jest używane w konwersacji.

**TranState**
"C", jeśli w konwersacji, w przeciwnym razie puste

**CommitMode**
Tryb kontroli transakcji z nagłówka OTMA ("0" lub "1")

**SecurityScope**
Wartość pusta

**Zarezerwowane**
Wartość pusta

z/OS *Komunikaty odpowiedzi od IMS*

Gdy transakcja IMS kieruje dane ISRT do swojego IOPCB, komunikat jest kierowany z powrotem do źródłowego LTERM lub TPIPE.

Są one wyświetlane w pliku IBM MQ jako komunikaty odpowiedzi. Komunikaty odpowiedzi z produktu IMS są umieszczane w kolejce odpowiedzi określonej w pierwotnym komunikacie. Jeśli komunikatu nie można umieścić w kolejce odpowiedzi, jest on umieszczany w kolejce niedostarczonych komunikatów przy użyciu uprawnień mostu. Jeśli komunikat nie może zostać umieszczony w kolejce niedostarczonych komunikatów, do produktu IMS wysyłane jest potwierdzenie negatywne informujące, że komunikat nie może zostać odebrany. Odpowiedzialność za komunikat jest następnie zwracana do IMS. Jeśli używany jest tryb kontroli transakcji 0, komunikaty z tego potoku Tpipe nie są wysyłane do mostu i pozostają w kolejce IMS . Oznacza to, że do czasu restartu nie są wysyłane żadne dalsze komunikaty. Jeśli używany jest tryb kontroli transakcji 1, inne prace mogą być kontynuowane.

Jeśli odpowiedź ma strukturę MQIIH, jej typ formatu to MQFMT_IMS; jeśli nie, typ formatu jest określany przez nazwę IMS MOD używaną podczas wstawiania komunikatu.

z/OS *Używanie alternatywnych bloków PCB odpowiedzi w transakcjach IMS*

Gdy transakcja systemu IMS używa alternatywnych bloków PCB odpowiedzi (ISRTs do ALTPCB lub wysyła wywołanie CHNG do modyfikowalnego bloku PCB), wywoływane jest wyjście przed routingiem (DFSYPRX0) w celu określenia, czy komunikat powinien zostać przekierowany.

Jeśli komunikat ma zostać przekierowany, wywoływane jest wyjście rozstrzygania miejsca docelowego (DFSYDRU0) w celu potwierdzenia miejsca docelowego i przygotowania informacji nagłówkowych. Informacje na temat tych programów obsługi wyjścia zawiera sekcja Używanie wyjść OTMA w IMS oraz sekcja Wyjście routingu wstępnego DFSYPRX0 .

Jeśli nie zostanie wykonane działanie w wyjściach, wszystkie dane wyjściowe z transakcji IMS zainicjowanych z menedżera kolejek IBM MQ , zarówno do IOPCB, jak i do ALTPCB, zostaną zwrócone do tego samego menedżera kolejek.

z/OS *Wysyłanie niezamówionych komunikatów z serwisu IMS*

Aby wysłać komunikaty z produktu IMS do kolejki IBM MQ , należy wywołać transakcję IMS , która ISRTs do ALTPCB.

Konieczne jest napisanie programów zewnętrznych routingu wstępnego i rozstrzygania miejsc docelowych w celu kierowania niezamówionych komunikatów z produktu IMS i budowania danych użytkownika OTMA, aby możliwe było poprawne zbudowanie deskryptora MQMD komunikatu. Informacje na temat tych programów obsługi wyjścia znajdują się w sekcji Program obsługi wyjścia routingu wstępnego DFSYPRX0 i Program obsługi wyjścia o docelowej rozdzielczości .

**Uwaga:** Most IBM MQ - IMS nie wie, czy odbierany komunikat jest odpowiedzią, czy niezamówionym komunikatem. Obsługuje on komunikat w taki sam sposób w każdym przypadku, budując deskryptorze MQMD i MQIIH odpowiedzi na podstawie UserData z OTMA, który został odebrany wraz z komunikatem.

Niezamówione komunikaty mogą tworzyć nowe potoki Tpotoków. Na przykład, jeśli istniejąca transakcja IMS została przełączona na nowy terminal LTERM (na przykład PRINT01), ale implementacja wymaga, aby dane wyjściowe były dostarczane za pośrednictwem OTMA, zostanie utworzony nowy potok Tpipe (w tym przykładzie nazywany PRINT01 ). Domyślnie jest to niezsynchronizowany Tpipe. Jeśli implementacja wymaga, aby komunikat był odtwarzalny, należy ustawić flagę wyjścia wyjścia rozstrzygnięcia miejsca docelowego. Więcej informacji na ten temat zawiera publikacja *IMS Customization Guide* .

z/OS *Segmentacja komunikatów*

Transakcje IMS można zdefiniować jako oczekiwane dane wejściowe jedno-lub wielosegmentowe.

Źródłowa aplikacja IBM MQ musi tworzyć dane wejściowe użytkownika następujące po strukturze MQIIH jako jeden lub więcej segmentów danych LLZZ. Wszystkie segmenty komunikatu IMS muszą być zawarte w pojedynczym komunikacie IBM MQ wysłanym za pomocą pojedynczego wywołania MQPUT.

Maksymalna długość segmentu danych LLZZ jest definiowana przez IMS/OTMA (32767 bajtów). Łączna długość komunikatu IBM MQ jest sumą bajtów LL plus długość struktury MQIIH.

Wszystkie segmenty odpowiedzi są zawarte w pojedynczym komunikacie IBM MQ .

Istnieje dodatkowe ograniczenie 32 kB dotyczące komunikatów w formacie MQFMT_IMS_VAR_STRING. Gdy dane w komunikacie ASCII-mixed CCSID są konwertowane na komunikat EBCDIC-mixed CCSID, bajt shift-in lub bajt shift-out jest dodawany za każdym razem, gdy występuje przejście między znakami SBCS i DBCS. Ograniczenie 32 kB dotyczy maksymalnej wielkości komunikatu. Oznacza to, że ponieważ pole LL w komunikacie nie może być większe niż 32 kB, komunikat nie może być dłuższy niż 32 kB, w tym wszystkie znaki shift-in i shift-out. Aplikacja budująca komunikat musi na to zezwolić.

z/OS *Konwersja danych dla komunikatów do i z mostu IMS*

Konwersja danych jest wykonywana przez rozproszoną funkcję kolejkowania (która może wywoływać wszystkie niezbędne wyjścia) lub przez wewnątrzgrupowy agent kolejkowania (który nie obsługuje użycia wyjść), gdy umieszcza komunikat w kolejce docelowej, która ma informacje XCF zdefiniowane dla swojej klasy pamięci masowej. Konwersja danych nie jest wykonywana, gdy komunikat jest dostarczany do kolejki przez publikowanie/subskrypcję.

Wszystkie wymagane wyjścia muszą być dostępne dla rozproszonego narzędzia kolejkowania w zestawie danych, do którego odwołuje się instrukcja CSQXLIB DD. Oznacza to, że można wysyłać komunikaty do aplikacji IMS przy użyciu mostu IBM MQ - IMS z dowolnej platformy IBM MQ .

W przypadku wystąpienia błędów konwersji komunikat jest umieszczany w kolejce bez konwersji. W rezultacie most IBM MQ - IMS traktuje go jako błąd, ponieważ most nie rozpoznaje formatu nagłówka. Jeśli wystąpi błąd konwersji, do konsoli z/OS zostanie wysłany komunikat o błędzie.

Szczegółowe informacje na temat konwersji danych znajdują się w sekcji “Zapisywanie wyjść konwersji danych” na stronie 1013 .

### Wysyłanie komunikatów do mostu IBM MQ - IMS

Aby zapewnić poprawne wykonanie konwersji, należy poinformować menedżer kolejek o formacie komunikatu.

Jeśli komunikat ma strukturę MQIIH, parametr *Format* w strukturze MQMD musi być ustawiony na wbudowany format MQFMT_IMS, a parametr *Format* w strukturze MQIIH musi być ustawiony na nazwę formatu opisującego dane komunikatu. Jeśli nie ma MQIIH, ustaw wartość *Format* w deskryptorze MQMD na nazwę formatu.

Jeśli wszystkie dane (inne niż LLZZs) są danymi znakowymi (MQCHAR), należy użyć jako nazwy formatu (odpowiednio w MQIIH lub MQMD) wbudowanego formatu MQFMT_IMS_VAR_STRING. W przeciwnym razie należy użyć własnej nazwy formatu. W takim przypadku należy również podać wyjście konwersji danych dla formatu. Wyjście musi obsługiwać konwersję LLZZs w komunikacie, oprócz samych danych (ale nie musi obsługiwać żadnych MQIIH na początku komunikatu).

Jeśli aplikacja używa produktu *MFSMapName*, można użyć komunikatów z systemem MQFMT_IMS i zdefiniować nazwę odwzorowania przekazaną do transakcji IMS w polu MFSMapName MQIIH.

### Odbieranie komunikatów z mostu IBM MQ - IMS

Jeśli w oryginalnym komunikacie wysyłanym do produktu IMSznajduje się struktura MQIIH, taka struktura jest również obecna w komunikacie odpowiedzi.

Aby upewnić się, że odpowiedź została poprawnie przekształcona:

- Jeśli w oryginalnym komunikacie istnieje struktura MQIIH, określ format komunikatu odpowiedzi w polu MQIIH *ReplytoFormat* komunikatu oryginalnego. Ta wartość jest umieszczana w polu MQIIH *Format* komunikatu odpowiedzi. Jest to szczególnie przydatne, gdy wszystkie dane wyjściowe mają postać LLZZ < dane znakowe >.
- Jeśli w oryginalnym komunikacie nie ma struktury MQIIH, należy określić format komunikatu odpowiedzi jako nazwę MFS MOD w narzędziu ISRT aplikacji IMS do IOPCB.

## Tworzenie aplikacji JMS/Jakarta Messaging i Java

IBM MQ udostępnia trzy interfejsy języka Java : IBM MQ classes for Jakarta Messaging, IBM MQ classes for JMSi IBM MQ classes for Java.

### O tym zadaniu

V 9.3.0 JM 3.0 V 9.3.0 **IBM MQ classes for Jakarta Messaging**

IBM MQ classes for Jakarta Messaging jest dostawcą Jakarta Messaging , który implementuje interfejsy Jakarta Messaging dla produktu IBM MQ jako system przesyłania komunikatów. Produkt Jakarta Connectors Architecture udostępnia standardowy sposób łączenia aplikacji działających w środowisku Jakarta EE z systemem informacyjnym przedsiębiorstwa (Enterprise Information System-EIS), takim jak IBM MQ lub Db2.

Więcej informacji na ten temat zawierają sekcje “Dlaczego należy używać IBM MQ classes for Jakarta Messaging?” na stronie 86 i “Uzyskiwanie dostępu do produktu IBM MQ z poziomu programu Java -wybór interfejsu API” na stronie 89.

JMS 2.0 **IBM MQ classes for JMS**

IBM MQ classes for JMS jest dostawcą JMS , który implementuje interfejsy JMS dla produktu IBM MQ jako system przesyłania komunikatów. Java Platform, Enterprise Edition Connector Architecture (JCA)

udostępnia standardowy sposób łączenia aplikacji działających w środowisku Java EE z systemem EIS (Enterprise Information System), takim jak IBM MQ lub Db2.

Więcej informacji na ten temat zawierają sekcje "Dlaczego należy używać IBM MQ classes for JMS?" na stronie 87 i "Uzyskiwanie dostępu do produktu IBM MQ z poziomu programu Java -wybór interfejsu API" na stronie 89.

**IBM MQ classes for Java**

IBM MQ classes for Java umożliwia korzystanie z produktu IBM MQ w środowisku Java . IBM MQ classes for Java Zezwalaj aplikacji Java na nawiązywanie połączenia z produktem IBM MQ jako klientem IBM MQ lub nawiązywanie połączenia bezpośrednio z menedżerem kolejek produktu IBM MQ .

Produkt IBM MQ classes for Java hermetyzuje interfejs kolejki komunikatów (Message Queue Interface-MQI), rodzimy interfejs API języka IBM MQ , i używa tego samego modelu obiektowego co inne interfejsy obiektowe, podczas gdy produkty IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging implementują interfejsy przesyłania komunikatów Java odpowiednio z baz danych Oracle i Java Community Process .

Więcej informacji na ten temat zawierają sekcje "Dlaczego należy używać IBM MQ classes for Java?" na stronie 359 i "Uzyskiwanie dostępu do produktu IBM MQ z poziomu programu Java -wybór interfejsu API" na stronie 89.

**Uwaga:**

Stabilized Produkt IBM nie zawiera żadnych dodatkowych udoskonaleń w produkcie IBM MQ classes for Java i są one funkcjonalnie ustabilizowane na poziomie dostarczanym wraz z produktem IBM MQ 8.0. Istniejące aplikacje korzystające z produktu IBM MQ classes for Java nadal będą w pełni obsługiwane, ale nowe funkcje nie zostaną dodane, a żądania rozszerzeń zostaną odrzucone. W pełni obsługiwane oznacza, że defekty zostaną usunięte wraz ze wszystkimi zmianami wymaganymi przez zmiany w wymaganiach systemowych produktu IBM MQ .

IBM MQ classes for Java nie są obsługiwane w systemie IMS.

IBM MQ classes for Java nie są obsługiwane w systemie WebSphere Liberty. Nie można ich używać z funkcją przesyłania komunikatów produktu IBM MQ Liberty ani z ogólną obsługą języka JCA . Więcej informacji na ten temat zawiera sekcja Używanie interfejsów produktu WebSphere MQ Java w środowiskach J2EE/JEE.

## Korzystanie z IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging są dostawcami przesyłania komunikatów produktu Java dostarczanymi wraz z produktem IBM MQ. Oprócz implementowania interfejsów zdefiniowanych w specyfikacjach JMS i Jakarta Messaging dostawcy przesyłania komunikatów dodają dwa zestawy rozszerzeń do interfejsu API przesyłania komunikatów produktu Java .

JM 3.0 V 9.3.0 V 9.3.0 W produkcie IBM MQ 9.3.0produkt Jakarta Messaging 3.0 jest obsługiwany na potrzeby tworzenia nowych aplikacji. Produkt IBM MQ 9.3.0 nadal obsługuje produkt JMS 2.0 dla istniejących aplikacji. Nie jest obsługiwane używanie zarówno interfejsu API JMS 2.0 , jak i interfejsu API Jakarta Messaging 3.0 w tej samej aplikacji.

**Uwaga:** W przypadku systemu Jakarta Messaging 3.0sterowanie specyfikacją JMS jest przenoszone z Oracle do Java Community Process. Jednak środowisko Oracle zachowuje kontrolę nad nazwą javax, która jest używana w innych technologiach Java , które nie zostały przeniesione do procesu społeczności Java . Tak więc Jakarta Messaging 3.0 jest funkcjonalnie równoważne JMS 2.0 istnieją pewne różnice w nazewnictwie:

- Oficjalna nazwa wersji 3.0 to Jakarta Messaging , a nie Java Message Service.
- Nazwy pakietów i stałych są poprzedzone przedrostkiem `jakarta` , a nie `javax`. Na przykład w systemie JMS 2.0 początkowe połączenie z dostawcą przesyłania komunikatów to obiekt `javax.jms.Connection` , a w systemie Jakarta Messaging 3.0 jest to obiekt `jakarta.jms.Connection` .

JMS 2.0 Pakiety javax.jms definiują interfejsy JMS , a dostawca JMS implementuje te interfejsy dla konkretnego produktu przesyłania komunikatów. IBM MQ classes for JMS jest dostawcą JMS , który implementuje interfejsy JMS dla IBM MQ.

JM 3.0 Pakiety jakarta.jms definiują interfejsy języka Jakarta Messaging , a dostawca Jakarta Messaging implementuje te interfejsy dla konkretnego produktu przesyłania komunikatów. IBM MQ classes for Jakarta Messaging jest dostawcą Jakarta Messaging , który implementuje interfejsy Jakarta Messaging dla IBM MQ.

Specyfikacje JMS i Jakarta Messaging oczekują, że obiekty ConnectionFactory i Destination będą administrowane. Administrator tworzy i obsługuje obiekty administrowane w centralnym repozytorium, a aplikacja JMS lub Jakarta Messaging pobiera te obiekty za pomocą programu Java Naming Directory Interface (JNDI).

JMS 2.0 W przypadku systemu JMS 2.0administrator może użyć narzędzia administracyjnego IBM MQ JMS **JMSAdmin**lub IBM MQ Explorer, aby utworzyć i obsługiwać obiekty administrowane w centralnym repozytorium.

JM 3.0 W przypadku systemu Jakarta Messaging 3.0nie można administrować interfejsem JNDI przy użyciu programu IBM MQ Explorer. Administrowanie interfejsem JNDI jest obsługiwane przez Jakarta Messaging 3.0 wariant **JMSAdmin**, który ma wartość **JMS30Admin**.

Ponieważ JMS i Jakarta Messaging współużytkują wiele wspólnych elementów, dalsze odwołania do JMS w tym temacie mogą być traktowane jako odwołania do obu tych elementów. Wszelkie różnice są podświetlane w razie potrzeby.

IBM MQ classes for JMS udostępnia również dwa zestawy rozszerzeń dla interfejsu API języka JMS . Głównym celem tych rozszerzeń jest dynamiczne tworzenie i konfigurowanie fabryk połączeń i miejsc docelowych w czasie wykonywania, ale udostępniają one również funkcje, które nie są bezpośrednio związane z przesyłaniem komunikatów, takie jak funkcje służące do określania problemów.

**Rozszerzenia IBM MQ JMS**

IBM MQ classes for JMS zawiera rozszerzenia zaimplementowane w obiektach, takich jak MQConnectionFactory, MQQueue i MQTopic. Te obiekty mają właściwości i metody specyficzne dla produktu IBM MQ. Obiekty mogą być obiektami administrowanymi lub aplikacja może tworzyć obiekty dynamicznie w czasie wykonywania. Te rozszerzenia są określane jako rozszerzenia produktu IBM MQ JMS .

**Rozszerzenia IBM JMS**

Produkt IBM MQ classes for JMS udostępnia również bardziej ogólny zestaw rozszerzeń interfejsu API języka JMS , które nie są specyficzne dla produktu IBM MQ jako systemu przesyłania komunikatów lub Java jako używanego języka programowania. Te rozszerzenia są określane jako rozszerzenia produktu IBM JMS i mają następujące ogólne cele:

- Zapewnienie wyższego poziomu spójności między dostawcami IBM JMS .
- Ułatwienie pisania aplikacji pomostowej między dwoma systemami przesyłania komunikatów IBM .
- Ułatwia przeniesienie aplikacji z jednego dostawcy IBM JMS do innego.

Rozszerzenia udostępniają funkcje podobne do tych, które są dostępne w systemach IBM MQ Message Service Client (XMS) for C/C++ i IBM MQ Message Service Client (XMS) for .NET.

**Pojęcia pokrewne**

Interfejsy języka IBM MQ Java

**Zadania pokrewne**

“Pisanie aplikacji IBM MQ classes for JMS/Jakarta Messaging” na stronie 145
Po krótkim wprowadzeniu do modelu JMS ta sekcja zawiera szczegółowe wskazówki dotyczące pisania aplikacji IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging .

## V 9.3.0 JM 3.0 V 9.3.0 Dlaczego należy używać IBM MQ classes for Jakarta Messaging?

Korzystanie z produktu IBM MQ classes for Jakarta Messaging ma wiele zalet, w tym możliwość ponownego wykorzystania istniejących umiejętności Jakarta Messaging w organizacji, a aplikacje są bardziej niezależne od dostawcy Jakarta Messaging i bazowej konfiguracji produktu IBM MQ .

### Podsumowanie korzyści wynikających z używania produktu IBM MQ classes for Jakarta Messaging

Użycie IBM MQ classes for Jakarta Messaging pozwala na ponowne wykorzystanie istniejących umiejętności Jakarta Messaging i zapewnia niezależność od aplikacji.

- Możesz ponownie wykorzystać Jakarta Messaging umiejętności.

  IBM MQ classes for Jakarta Messaging jest dostawcą Jakarta Messaging , który implementuje interfejsy Jakarta Messaging dla produktu IBM MQ jako system przesyłania komunikatów. Jeśli organizacja jest nowa w produkcie IBM MQ, ale dysponuje już umiejętnościami w zakresie tworzenia aplikacji w systemie Jakarta Messaging (lub JMS), wygodniejsze może być użycie znanego interfejsu API Jakarta Messaging do uzyskiwania dostępu do zasobów IBM MQ , a nie jednego z innych interfejsów API dostarczonych z produktem IBM MQ.

- Jakarta Messaging jest integralną częścią systemu Jakarta EE.

  Jakarta Messaging to naturalny interfejs API używany na potrzeby przesyłania komunikatów na platformie Jakarta EE . Każdy serwer aplikacji zgodny z systemem Jakarta EE musi zawierać dostawcę Jakarta Messaging . Produktu Jakarta Messaging można używać w klientach aplikacji, serwletach, stronach JSP ( Java Server Pages), komponentach EJB (Enterprise Java Bean) i komponentach MDB (Message Driven Bean). W szczególności należy zauważyć, że aplikacje Jakarta EE używają komponentów MDB do asynchronicznego przetwarzania komunikatów, a wszystkie komunikaty są dostarczane do komponentów MDB jako komunikaty Jakarta Messaging .

- Fabryki połączeń i miejsca docelowe mogą być przechowywane jako obiekty administrowane przez Jakarta Messaging w centralnym repozytorium, zamiast być zakodowane na stałe w aplikacji.

  Administrator może tworzyć i obsługiwać obiekty administrowane przez Jakarta Messaging w centralnym repozytorium, a aplikacje IBM MQ classes for Jakarta Messaging mogą pobierać te obiekty za pomocą programu Java Naming Directory Interface (JNDI). Fabryki połączeń i miejsca docelowe Jakarta Messaging hermetyzują informacje specyficzne dla produktu IBM MQ, takie jak nazwy menedżerów kolejek, nazwy kanałów, opcje połączeń, nazwy kolejek i nazwy tematów. Jeśli fabryki połączeń i miejsca docelowe są przechowywane jako obiekty administrowane, te informacje nie są zakodowane na stałe w aplikacji. Dzięki temu aplikacja będzie w pewnym stopniu niezależna od bazowej konfiguracji produktu IBM MQ .

- Jakarta Messaging to standardowy interfejs API, który zapewnia przenośność aplikacji.

  Aplikacja Jakarta Messaging może używać języka JNDI do pobierania fabryk połączeń i miejsc docelowych, które są przechowywane jako obiekty administrowane, a do wykonywania operacji przesyłania komunikatów może używać tylko interfejsów zdefiniowanych w pakiecie `jakarta.jms` (Jakarta Messaging 3.0). Aplikacja jest wtedy całkowicie niezależna od dowolnego dostawcy Jakarta Messaging , takiego jak IBM MQ classes for Jakarta Messaging, i może być przeniesiona z jednego dostawcy Jakarta Messaging do innego bez wprowadzania żadnych zmian w aplikacji.

  Jeśli produkt JNDI nie jest dostępny w konkretnym środowisku aplikacji, aplikacja IBM MQ classes for Jakarta Messaging może używać rozszerzeń interfejsu API języka Jakarta Messaging do dynamicznego tworzenia i konfigurowania fabryk połączeń i miejsc docelowych w czasie wykonywania. Aplikacja jest wtedy całkowicie samodzielna, ale jest powiązana z produktem IBM MQ classes for Jakarta Messaging jako dostawca Jakarta Messaging .

- Pisanie aplikacji mostu może być łatwiejsze, jeśli zostanie użyty produkt Jakarta Messaging.

  Aplikacja pomostowa to aplikacja, która odbiera komunikaty z jednego systemu przesyłania komunikatów i wysyła je do innego systemu przesyłania komunikatów. Pisanie aplikacji pomostowej

może być skomplikowane, jeśli używane są specyficzne dla produktu interfejsy API i formaty komunikatów. Zamiast tego można napisać aplikację pomostową przy użyciu dwóch dostawców Jakarta Messaging , po jednym dla każdego systemu przesyłania komunikatów. Aplikacja następnie używa tylko jednego interfejsu API, interfejsu API Jakarta Messaging i przetwarza tylko komunikaty Jakarta Messaging .

## Środowiska możliwe do wdrożenia

Aby zapewnić integrację z serwerem aplikacji Jakarta EE , standardy Jakarta EE wymagają od dostawców przesyłania komunikatów dostarczenia adaptera zasobów. Zgodnie ze specyfikacją Jakarta Connectors Architecture IBM MQ udostępnia adapter zasobów, który używa produktu Jakarta Messaging do udostępniania funkcji przesyłania komunikatów w dowolnym certyfikowanym środowisku Jakarta EE . Więcej informacji na ten temat zawiera sekcja “Liberty i adapter zasobów IBM MQ” na stronie 456.

**Uwaga:** WebSphere Application Server traditional obecnie nie obsługuje Jakarta EE.

Poza środowiskiem Jakarta EE udostępniono pliki OSGi i JAR, dzięki czemu łatwiej jest uzyskać tylko plik IBM MQ classes for Jakarta Messaging. Te pliki JAR są łatwiej dostępne do wdrożenia zarówno w środowisku autonomicznym, jak i w środowisku zarządzania oprogramowaniem, takim jak Maven. Więcej informacji na ten temat zawiera sekcja “Uzyskiwanie osobno produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging” na stronie 131.

**Pojęcia pokrewne**

IBM MQ classes for Jakarta Messaging: przegląd

“Uzyskiwanie dostępu do produktu IBM MQ z poziomu programu Java -wybór interfejsu API” na stronie 89

IBM MQ udostępnia trzy interfejsy języka Java .

## JMS 2.0 Dlaczego należy używać IBM MQ classes for JMS?

Korzystanie z produktu IBM MQ classes for JMS ma wiele zalet, w tym możliwość ponownego wykorzystania istniejących umiejętności JMS w organizacji, a aplikacje są bardziej niezależne od dostawcy JMS i bazowej konfiguracji produktu IBM MQ .

## Podsumowanie korzyści wynikających z używania produktu IBM MQ classes for JMS

Użycie IBM MQ classes for JMS pozwala na ponowne wykorzystanie istniejących umiejętności JMS i zapewnia niezależność od aplikacji.

V 9.3.0 V 9.3.0

**Uwaga:** JMS 2.0 został zastąpiony przez Jakarta Messaging. Produkt IBM MQ classes for JMS nadal obsługuje standard JMS 2.0 , ale przyszłe rozszerzenia przesyłania komunikatów produktu Java będą pojawiać się tylko w produkcie Jakarta Messaging, a więc w produkcie IBM MQ classes for Jakarta Messaging. IBM MQ classes for JMS są zalecane tylko w przypadku konserwacji i rozszerzania istniejących aplikacji JMS 2.0 . IBM MQ classes for Jakarta Messaging powinna być preferowaną technologią dla nowego programowania.

- Możesz ponownie wykorzystać JMS umiejętności.

  IBM MQ classes for JMS jest dostawcą JMS , który implementuje interfejsy JMS dla produktu IBM MQ jako system przesyłania komunikatów. Jeśli organizacja jest nowa w produkcie IBM MQ, ale ma już umiejętności programowania aplikacji w systemie JMS , wygodniejsze może być użycie znanego interfejsu API JMS do uzyskiwania dostępu do zasobów IBM MQ , a nie jednego z innych interfejsów API dostarczanych z produktem IBM MQ.
- JMS jest integralną częścią systemu Java Platform, Enterprise Edition (Java EE).

  JMS to naturalny interfejs API używany na potrzeby przesyłania komunikatów na platformie Java EE . Każdy serwer aplikacji zgodny z systemem Java EE musi zawierać dostawcę JMS . Produktu JMS można używać w klientach aplikacji, serwletach, stronach JSP ( Java Server Pages), komponentach EJB (Enterprise Java Bean) i komponentach MDB (Message Driven Bean). W szczególności należy zauważyć,

że aplikacje Java EE używają komponentów MDB do asynchronicznego przetwarzania komunikatów, a wszystkie komunikaty są dostarczane do komponentów MDB jako komunikaty JMS .

- Fabryki połączeń i miejsca docelowe mogą być przechowywane jako obiekty administrowane przez JMS w centralnym repozytorium, zamiast być zakodowane na stałe w aplikacji.

  Administrator może tworzyć i obsługiwać obiekty administrowane przez JMS w centralnym repozytorium, a aplikacje IBM MQ classes for JMS mogą pobierać te obiekty za pomocą programu Java Naming Directory Interface (JNDI). Fabryki połączeń i miejsca docelowe JMS hermetyzują informacje specyficzne dla produktu IBM MQ, takie jak nazwy menedżerów kolejek, nazwy kanałów, opcje połączeń, nazwy kolejek i nazwy tematów. Jeśli fabryki połączeń i miejsca docelowe są przechowywane jako obiekty administrowane, te informacje nie są zakodowane na stałe w aplikacji. Dzięki temu aplikacja będzie w pewnym stopniu niezależna od bazowej konfiguracji produktu IBM MQ .

- JMS to standardowy interfejs API, który zapewnia przenośność aplikacji.

  Aplikacja JMS może używać języka JNDI do pobierania fabryk połączeń i miejsc docelowych, które są przechowywane jako obiekty administrowane, a do wykonywania operacji przesyłania komunikatów może używać tylko interfejsów zdefiniowanych w pakiecie `javax.jms` . Aplikacja jest wtedy całkowicie niezależna od dowolnego dostawcy JMS , takiego jak IBM MQ classes for JMS, i może być przeniesiona z jednego dostawcy JMS do innego bez wprowadzania żadnych zmian w aplikacji.

  Jeśli produkt JNDI nie jest dostępny w konkretnym środowisku aplikacji, aplikacja IBM MQ classes for JMS może używać rozszerzeń interfejsu API języka JMS do dynamicznego tworzenia i konfigurowania fabryk połączeń i miejsc docelowych w czasie wykonywania. Aplikacja jest wtedy całkowicie samodzielna, ale jest powiązana z produktem IBM MQ classes for JMS jako dostawca JMS .

- Pisanie aplikacji mostu może być łatwiejsze, jeśli zostanie użyty produkt JMS.

  Aplikacja pomostowa to aplikacja, która odbiera komunikaty z jednego systemu przesyłania komunikatów i wysyła je do innego systemu przesyłania komunikatów. Pisanie aplikacji pomostowej może być skomplikowane, jeśli używane są specyficzne dla produktu interfejsy API i formaty komunikatów. Zamiast tego można napisać aplikację pomostową przy użyciu dwóch dostawców JMS , po jednym dla każdego systemu przesyłania komunikatów. Aplikacja następnie używa tylko jednego interfejsu API, interfejsu API JMS i przetwarza tylko komunikaty JMS .

## Środowiska możliwe do wdrożenia

Aby zapewnić integrację z serwerem aplikacji Java EE , standardy Java EE wymagają od dostawców przesyłania komunikatów dostarczenia adaptera zasobów. Zgodnie ze specyfikacją Java EE Connector Architecture (JCA), IBM MQ udostępnia adapter zasobów, który używa JMS do udostępniania funkcji przesyłania komunikatów w dowolnym certyfikowanym środowisku Java EE .

Chociaż możliwe było użycie IBM MQ classes for Java wewnątrz Java EE, ten interfejs API nie został zaprojektowany ani zoptymalizowany do tego celu. Więcej informacji na temat uwag dotyczących produktu IBM MQ classes for Java w produkcie Java EEzawiera sekcja “Uruchamianie aplikacji IBM MQ classes for Java w produkcie Java EE” na stronie 360.

Poza środowiskiem Java EE udostępniono pliki OSGi i JAR, dzięki czemu łatwiej jest uzyskać tylko plik IBM MQ classes for JMS. Te pliki JAR są łatwiej dostępne do wdrożenia zarówno w środowisku autonomicznym, jak i w środowisku zarządzania oprogramowaniem, takim jak Maven. Więcej informacji na ten temat zawiera sekcja “Uzyskiwanie osobno produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging” na stronie 131.

**Pojęcia pokrewne**

V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging: przegląd

“Dlaczego należy używać IBM MQ classes for Jakarta Messaging?” na stronie 86
Korzystanie z produktu IBM MQ classes for Jakarta Messaging ma wiele zalet, w tym możliwość ponownego wykorzystania istniejących umiejętności Jakarta Messaging w organizacji, a aplikacje są bardziej niezależne od dostawcy Jakarta Messaging i bazowej konfiguracji produktu IBM MQ .

“Uzyskiwanie dostępu do produktu IBM MQ z poziomu programu Java -wybór interfejsu API” na stronie 89

IBM MQ udostępnia trzy interfejsy języka Java .

## Uzyskiwanie dostępu do produktu IBM MQ z poziomu programu Java -wybór interfejsu API

IBM MQ udostępnia trzy interfejsy języka Java .

- V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging
- IBM MQ classes for JMS
- IBM MQ classes for Java

### IBM MQ classes for Jakarta Messaging

V 9.3.0 V 9.3.0 Produkt IBM MQ classes for Jakarta Messaging umożliwia aplikacjom napisanym przy użyciu interfejsów API języka Jakarta Messaging 3.0 korzystanie z produktu IBM MQ jako dostawcy przesyłania komunikatów.

Jakarta Messaging to strategiczny kierunek przesyłania komunikatów w aplikacjach Java .

Produkt Jakarta Messaging 3.0 jest funkcjonalnie równoważny produktowi JMS 2.0, dlatego więcej informacji na ten temat zawiera sekcja “Korzystanie z IBM MQ classes for JMS/Jakarta Messaging” na stronie 84.

### IBM MQ classes for JMS

Produkt IBM MQ classes for JMS umożliwia aplikacjom napisanym przy użyciu interfejsów API języka JMS 2.0 korzystanie z produktu IBM MQ jako dostawcy przesyłania komunikatów.

Jako Jakarta Messaging zastępuje JMS, zaleca się użycie IBM MQ classes for JMS w istniejących aplikacjach lub w środowiskach (na przykład WebSphere Application Server), które nie obsługują Jakarta Messaging.

Nie jest obsługiwane używanie zarówno języka IBM MQ classes for Jakarta Messaging , jak i języka IBM MQ classes for JMS w tej samej aplikacji.

Więcej informacji na ten temat zawiera sekcja “Korzystanie z IBM MQ classes for JMS/Jakarta Messaging” na stronie 84.

### IBM MQ classes for Java

Stabilized Innym interfejsem API, którego aplikacje Java mogą używać w celu uzyskiwania dostępu do zasobów IBM MQ , jest interfejs IBM MQ classes for Java, który udostępnia zorientowany na produkt IBM MQinterfejs API dla programów korzystających z produktu IBM MQ jako dostawcy przesyłania komunikatów. Jednak produkt IBM MQ classes for Java jest stabilny funkcjonalnie na poziomie dostarczonym z produktem IBM MQ 8.0. Więcej informacji na ten temat zawiera sekcja “Dlaczego należy używać IBM MQ classes for Java?” na stronie 359. Chociaż istniejące aplikacje, które korzystają z produktu IBM MQ classes for Java , nadal są w pełni obsługiwane, nowe aplikacje powinny używać produktu IBM MQ classes for Jakarta Messaging.

## Wspólne funkcje produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging

IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging zapewniają dostęp zarówno do funkcji przesyłania komunikatów w trybie punkt z punktem, jak i publikowania/subskrypcji produktu IBM MQ. Oprócz wysyłania komunikatów JMS , które zapewniają obsługę standardowego modelu przesyłania komunikatów produktu JMS , aplikacje mogą również wysyłać i odbierać komunikaty bez dodatkowych nagłówków, co pozwala na współdziałanie z innymi aplikacjami produktu IBM MQ , na przykład aplikacjami MQI języka C. Dostępna jest pełna kontrola ładunków komunikatów MQMD i MQ .

Dostępne są także dodatkowe funkcje produktu IBM MQ , takie jak przetwarzanie strumieniowe komunikatów, asynchroniczne umieszczanie komunikatów i raportowanie.

Przy użyciu dostarczonych klas pomocniczych PCF, komunikaty administracyjne PCF IBM MQ mogą być wysyłane i odbierane za pośrednictwem interfejsu API języka JMS i mogą być używane do administrowania menedżerami kolejek.

Funkcje, które zostały ostatnio dodane do produktu IBM MQ, takie jak wykorzystanie asynchroniczne i automatyczne ponowne połączenie, nie są dostępne w produkcie IBM MQ classes for Java, ale są dostępne w produkcie IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging.

## Żądania udoskonaleń

Jeśli potrzebny jest dostęp do funkcji programu IBM MQ , które nie są dostępne za pośrednictwem produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging, można zgłosić pomysł.

IBM może następnie poinformować, czy implementacja jest możliwa w implementacji IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging , lub czy istnieje sprawdzona procedura, która może być stosowana.

W przypadku dodatkowych funkcji przesyłania komunikatów, ponieważ produkt IBM jest kontrybutorem standardu otwartego, te funkcje mogą zostać zgłoszone jako część procesu JCP. Mają one zastosowanie tylko do Jakarta Messaging.

**Informacje pokrewne**

Witamy w portalu IBM Ideas Portal

Proces przeglądu specyfikacji JMS Java

Używanie usługi JMS do wysyłania komunikatów PCF

## V 9.3.0 JM 3.0 V 9.3.0 Wymagania wstępne dla produktu IBM MQ classes for Jakarta Messaging

W tym temacie opisano, co należy wiedzieć przed użyciem produktu IBM MQ classes for Jakarta Messaging. Do tworzenia i uruchamiania aplikacji IBM MQ classes for Jakarta Messaging niezbędne są pewne komponenty oprogramowania, które są wstępnie wymagane.

Informacje na temat wymagań wstępnych dla produktu IBM MQ classes for Jakarta Messagingzawiera sekcja Wymagania systemowe produktu IBM MQ.

Do tworzenia aplikacji IBM MQ classes for Jakarta Messaging potrzebny jest pakiet Java SE Software Development Kit (SDK). Szczegółowe informacje na temat pakietów JDK obsługiwanych przez system operacyjny zawiera sekcja Wymagania systemowe produktu IBM MQ.

Do uruchamiania aplikacji IBM MQ classes for Jakarta Messaging potrzebne są następujące komponenty oprogramowania:

- Menedżer kolejek systemu IBM MQ .
- Środowisko Java runtime environment (JRE) dla każdego systemu, w którym uruchamiane są aplikacje.
- IBM i W systemie IBM i: Qshell, czyli opcja 30 systemu operacyjnego.
- z/OS W systemie z/OS: z/OS UNIX System Services (z/OS UNIX).

Dostawca JSSE IBM zawiera dostawcę kryptograficznego z certyfikatem FIPS, dlatego można go programowo skonfigurować pod kątem zgodności ze standardem FIPS 140-2, który jest gotowy do natychmiastowego użycia. Dlatego zgodność ze standardem FIPS 140-2 może być obsługiwana bezpośrednio z serwisu IBM MQ classes for Jakarta Messaging.

Dostawca JSSE Oraclemoże mieć skonfigurowanego dostawcy kryptograficznego z certyfikatem FIPS, ale nie jest on gotowy do natychmiastowego użycia i nie jest dostępny do konfiguracji programowej. Dlatego w tym przypadku produkt IBM MQ classes for Jakarta Messaging nie może bezpośrednio włączyć

zgodności ze standardem FIPS 140-2. Może być możliwe ręczne włączenie takiej zgodności, ale produkt IBM nie może obecnie udzielać na ten temat wskazówek.

W aplikacjach IBM MQ classes for Jakarta Messaging można używać adresów protokołu Internet Protocol w wersji 6 (IPv6), jeśli adresy IPv6 są obsługiwane przez wirtualną maszynę języka Java (JVM) i implementację protokołu TCP/IP w systemie operacyjnym. Narzędzie administracyjne IBM MQ Jakarta Messaging , **JMS30Admin**, również akceptuje adresy IPv6 . Więcej informacji na temat tego narzędzia zawiera sekcja Konfigurowanie obiektów JMS i Jakarta Messaging przy użyciu narzędzi administracyjnych.

Narzędzie administracyjne IBM MQ JMS i IBM MQ Explorer używają Java Naming Directory Interface (JNDI) do uzyskania dostępu do usługi katalogowej, w której przechowywane są obiekty administrowane. Aplikacje IBM MQ classes for Jakarta Messaging mogą również używać języka JNDI do pobierania obiektów administrowanych z usługi katalogowej.

**Uwaga:** W przypadku systemu Jakarta Messaging 3.0nie można administrować interfejsem JNDI przy użyciu programu IBM MQ Explorer. Administrowanie interfejsem JNDI jest obsługiwane przez Jakarta Messaging 3.0 wariant **JMSAdmin**, który ma wartość **JMS30Admin**.

Dostawca usług to kod, który zapewnia dostęp do usługi katalogowej, odwzorowując wywołania JNDI do usługi katalogowej. Dostawca usług systemu plików w plikach `fscontext.jar` i `providerutil.jar` jest dostarczany z produktem IBM MQ classes for Jakarta Messaging. Dostawca usług systemu plików zapewnia dostęp do usługi katalogowej w oparciu o lokalny system plików.

Jeśli ma być używana usługa katalogowa oparta na serwerze LDAP, należy zainstalować i skonfigurować serwer LDAP lub mieć dostęp do istniejącego serwera LDAP. W szczególności należy skonfigurować serwer LDAP do przechowywania obiektów Java . Informacje na temat instalowania i konfigurowania serwera LDAP zawiera dokumentacja dostarczana z serwerem.

## JMS 2.0 Wymagania wstępne dla produktu IBM MQ classes for JMS

W tym temacie opisano, co należy wiedzieć przed użyciem produktu IBM MQ classes for JMS.
Do tworzenia i uruchamiania aplikacji IBM MQ classes for JMS niezbędne są pewne komponenty oprogramowania, które są wstępnie wymagane.

Informacje na temat wymagań wstępnych dla produktu IBM MQ classes for JMSzawiera sekcja Wymagania systemowe produktu IBM MQ.

Do tworzenia aplikacji IBM MQ classes for JMS potrzebny jest pakiet Java SE Software Development Kit (SDK). Szczegółowe informacje na temat pakietów JDK obsługiwanych przez system operacyjny zawiera sekcja Wymagania systemowe produktu IBM MQ.

Do uruchamiania aplikacji IBM MQ classes for JMS potrzebne są następujące komponenty oprogramowania:

- Menedżer kolejek systemu IBM MQ .
- Środowisko Java runtime environment (JRE) dla każdego systemu, w którym uruchamiane są aplikacje.
- IBM i W systemie IBM i: Qshell, czyli opcja 30 systemu operacyjnego.
- z/OS W systemie z/OS: z/OS UNIX System Services (z/OS UNIX).

Dostawca JSSE IBM zawiera dostawcę kryptograficznego z certyfikatem FIPS, dlatego można go programowo skonfigurować pod kątem zgodności ze standardem FIPS 140-2, który jest gotowy do natychmiastowego użycia. Dlatego zgodność ze standardem FIPS 140-2 może być obsługiwana bezpośrednio z poziomu produktów IBM MQ classes for Java i IBM MQ classes for JMS.

Dostawca JSSE Oraclemoże mieć skonfigurowanego dostawcy kryptograficznego z certyfikatem FIPS, ale nie jest on gotowy do natychmiastowego użycia i nie jest dostępny do konfiguracji programowej. Dlatego w tym przypadku IBM MQ classes for Java i IBM MQ classes for JMS nie mogą bezpośrednio włączyć zgodności ze standardem FIPS 140-2. Może być możliwe ręczne włączenie takiej zgodności, ale produkt IBM nie może obecnie udzielać na ten temat wskazówek.

W aplikacjach IBM MQ classes for JMS można używać adresów protokołu Internet Protocol w wersji 6 (IPv6), jeśli adresy IPv6 są obsługiwane przez wirtualną maszynę języka Java (JVM) i implementację

protokołu TCP/IP w systemie operacyjnym. Narzędzie administracyjne IBM MQ JMS (patrz sekcja Konfigurowanie obiektów JMS za pomocą narzędzia administracyjnego ) akceptuje również adresy IPv6 .

Narzędzie administracyjne IBM MQ JMS i IBM MQ Explorer używają Java Naming Directory Interface (JNDI) do uzyskania dostępu do usługi katalogowej, w której przechowywane są obiekty administrowane. Aplikacje IBM MQ classes for JMS mogą również używać języka JNDI do pobierania obiektów administrowanych z usługi katalogowej. Dostawca usług to kod, który zapewnia dostęp do usługi katalogowej, odwzorowując wywołania JNDI do usługi katalogowej. Dostawca usług systemu plików w plikach `fscontext.jar` i `providerutil.jar` jest dostarczany z produktem IBM MQ classes for JMS. Dostawca usług systemu plików zapewnia dostęp do usługi katalogowej w oparciu o lokalny system plików.

Jeśli ma być używana usługa katalogowa oparta na serwerze LDAP, należy zainstalować i skonfigurować serwer LDAP lub mieć dostęp do istniejącego serwera LDAP. W szczególności należy skonfigurować serwer LDAP do przechowywania obiektów Java . Informacje na temat instalowania i konfigurowania serwera LDAP zawiera dokumentacja dostarczana z serwerem.

## Instalowanie i konfigurowanie produktu IBM MQ classes for JMS/Jakarta Messaging

W tej sekcji opisano katalogi i pliki, które są tworzone podczas instalowania produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging, oraz opisano sposób konfigurowania produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging po instalacji.

**Pojęcia pokrewne**

“Korzystanie z adaptera zasobów IBM MQ” na stronie 450
Adapter zasobów umożliwia aplikacjom działającym na serwerze aplikacji dostęp do zasobów IBM MQ . Obsługuje on komunikację przychodzącą i wychodzącą.

### *Co jest instalowane w systemie IBM MQ classes for JMS*

Podczas instalowania produktu IBM MQ classes for JMStworzona jest pewna liczba plików i katalogów. W systemie Windowsniektóre czynności konfiguracyjne są wykonywane podczas instalacji przez automatyczne ustawianie zmiennych środowiskowych. Na innych platformach i w niektórych środowiskach Windows należy ustawić zmienne środowiskowe przed uruchomieniem aplikacji IBM MQ classes for JMS .

W przypadku większości systemów operacyjnych IBM MQ classes for JMS są instalowane jako komponent opcjonalny podczas instalowania produktu IBM MQ.

Więcej informacji na temat instalowania produktu IBM MQmożna znaleźć w następujących sekcjach:

- Multi Instalowanie produktu IBM MQ
- z/OS Instalowanie produktu IBM MQ for z/OS

**Ważne:** Oprócz przemieszczalnych plików JAR opisanych w sekcji “Przemieszczalne pliki JAR IBM MQ classes for JMS/Jakarta Messaging” na stronie 94, kopiowanie plików JAR IBM MQ classes for JMS lub bibliotek rodzimych do innych komputerów lub do innego położenia na komputerze, na którym zainstalowano plik IBM MQ classes for JMS , nie jest obsługiwane.

## Katalogi instalacyjne

Tabela 5 na stronie 92 przedstawia miejsce, w którym są zainstalowane pliki IBM MQ classes for JMS na każdej platformie.

*Tabela 5. IBM MQ classes for JMS Katalogi instalacyjne*

| Platforma | Katalog |
|---|---|
| Linux AIX AIX and Linux | *MQ_INSTALLATION_PATH*`/java` |

| *Tabela 5. IBM MQ classes for JMS Katalogi instalacyjne (kontynuacja)* | |
|---|---|
| **Platforma** | **Katalog** |
| Windows Windows | *MQ_INSTALLATION_PATH*\java |
| IBM i IBM i | /QIBM/ProdData/mqm/java |
| z/OS z/OS | *MQ_INSTALLATION_PATH*/java |

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Katalog instalacyjny zawiera następującą treść:

- Pliki JAR IBM MQ classes for JMS , w tym przemieszczalne pliki JAR, które znajdują się w katalogu *MQ_INSTALLATION_PATH*\java\lib .
- Biblioteki rodzime produktu IBM MQ , które są używane przez aplikacje korzystające z rodzimego interfejsu produktu Java .

  32-bitowe biblioteki rodzime są instalowane w katalogu *MQ_INSTALLATION_PATH*\java\lib , a 64-bitowe biblioteki rodzime można znaleźć w katalogu *MQ_INSTALLATION_PATH*\java\lib64 .

  Więcej informacji na temat bibliotek rodzimych IBM MQ zawiera sekcja “Konfigurowanie bibliotek JNI ( Java Native Interface)” na stronie 100.
- Dodatkowe skrypty, które zostały opisane w sekcji “Skrypty udostępnione z produktem IBM MQ classes for JMS/Jakarta Messaging” na stronie 127. Te skrypty znajdują się w katalogu *MQ_INSTALLATION_PATH*\java\bin .
- Specyfikacje interfejsu API IBM MQ classes for JMS . Narzędzie Javadoc zostało użyte do wygenerowania stron HTML zawierających specyfikacje interfejsu API.

  Strony HTML znajdują się w katalogu *MQ_INSTALLATION_PATH*\java\doc\WMQJMSClasses :

  – ALW W systemie AIX, Linux, and Windowsten podkatalog zawiera pojedyncze strony HTML.

  – IBM i W systemie IBM istrony HTML znajdują się w pliku o nazwie wmqjms_javadoc.jar.

  – z/OS W systemie z/OSstrony HTML znajdują się w pliku o nazwie wmqjms_javadoc.jar.
- Wsparcie dla OGSi. Pakunki OSGi są instalowane w katalogu java\lib\OSGi i opisane w sekcji “Obsługa środowiska OSGi z produktem IBM MQ classes for JMS” na stronie 129.
- Adapter zasobów IBM MQ , który można wdrożyć na dowolnym serwerze aplikacji zgodnym z Java Platform, Enterprise Edition 7 ( Java EE 7) lub Jakarta EE .

  Adapter zasobów IBM MQ znajduje się w katalogu *MQ_INSTALLATION_PATH*\java\lib\jca . Więcej informacji na ten temat zawiera sekcja “Korzystanie z adaptera zasobów IBM MQ” na stronie 450
- Windows W systemie Windowssymbole, których można użyć do debugowania, są instalowane w katalogu *MQ_INSTALLATION_PATH*\java\lib\symbols .

Katalog instalacyjny zawiera również niektóre pliki należące do innych komponentów IBM MQ .

## Aplikacje przykładowe

JMS 2.0 Niektóre aplikacje przykładowe są dostarczane z produktem IBM MQ classes for JMS. Tabela 6 na stronie 93 pokazuje, gdzie na każdej platformie są zainstalowane aplikacje przykładowe.

JM 3.0 W systemie IBM MQ classes for Jakarta Messagingprzygotowywane są nowe przykłady.

JMS 2.0

| *Tabela 6. Katalogi przykładów dla produktu IBM MQ classes for JMS* | |
|---|---|
| **Platforma** | **Katalog** |
| Linux AIX AIX and Linux | *MQ_INSTALLATION_PATH*/samp/jms |
| Windows Windows | *MQ_INSTALLATION_PATH*\tools\jms |
| IBM i IBM i | /QIBM/ProdData/mqm/java/samples/jms |
| z/OS z/OS | *MQ_INSTALLATION_PATH*/java/samples/jms |

W tej tabeli *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym zainstalowano produkt IBM MQ .

Po zakończeniu instalacji może być konieczne wykonanie pewnych czynności konfiguracyjnych w celu skompilowania i uruchomienia aplikacji.

"Ustawianie zmiennych środowiskowych dla IBM MQ classes for JMS/Jakarta Messaging" na stronie 97 opisuje ścieżkę klasy, która jest wymagana do uruchomienia przykładowych aplikacji IBM MQ classes for JMS . W tym temacie opisano również dodatkowe pliki JAR, które muszą być przywoływane w szczególnych okolicznościach, oraz zmienne środowiskowe, które należy ustawić w celu uruchomienia skryptów dostarczanych z produktem IBM MQ classes for JMS.

Aby kontrolować właściwości, takie jak śledzenie i rejestrowanie aplikacji, należy udostępnić plik właściwości konfiguracyjnych. Plik właściwości konfiguracyjnych IBM MQ classes for JMS jest opisany w sekcji "Plik konfiguracyjny IBM MQ classes for JMS/Jakarta Messaging" na stronie 102.

**Pojęcia pokrewne**

Problemy z wdrażaniem adaptera zasobów

**Zadania pokrewne**

"Korzystanie z przykładowych aplikacji IBM MQ classes for JMS" na stronie 124
Przykładowe aplikacje IBM MQ classes for JMS udostępniają przegląd wspólnych funkcji interfejsu API języka JMS . Można ich używać do weryfikowania konfiguracji serwera instalacji i przesyłania komunikatów oraz do tworzenia własnych aplikacji.

*Przemieszczalne pliki JAR IBM MQ classes for JMS/Jakarta Messaging*
Przemieszczalne pliki JAR można przenieść do systemów, w których musi być uruchomiony system IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging.

**Ważne:**

- Oprócz przemieszczalnych plików JAR opisanych w sekcji Relocatable JAR files, kopiowanie plików JAR IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging lub bibliotek rodzimych do innych komputerów lub do innego położenia na komputerze, na którym zainstalowano IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging , nie jest obsługiwane.
- Nie należy dołączać przemieszczalnych plików JAR do aplikacji wdrożonych na serwerach aplikacji Java EE , takich jak WebSphere Application Server lub WebSphere Liberty. W tych środowiskach adapter zasobów IBM MQ powinien zostać wdrożony i użyty. Należy zauważyć, że WebSphere Application Server osadza adapter zasobów IBM MQ , więc nie ma potrzeby wdrażania go ręcznie w tym środowisku.
- Aby uniknąć konfliktów programu ładującego klasy, nie zaleca się tworzenia pakunków przemieszczalnych plików JAR w obrębie wielu aplikacji w obrębie tego samego środowiska wykonawczego Java . W tym scenariuszu relokowalne pliki JAR IBM MQ są dostępne w ścieżce klasy środowiska wykonawczego Java .
- Jeśli relokowalne pliki JAR są umieszczane w pakunku aplikacji, należy upewnić się, że zostały uwzględnione wszystkie wstępnie wymagane pliki JAR zgodnie z opisem w sekcji Relocatable JAR files(Pliki JAR przemieszczalne). Należy również upewnić się, że istnieją odpowiednie procedury aktualizacji plików JAR w pakunku w ramach konserwacji aplikacji, aby upewnić się, że pliki IBM MQ

classes for JMS lub IBM MQ classes for Jakarta Messaging są nadal aktualne i że znane problemy są ponownie mediowane.

## Przemieszczalne pliki JAR

W przedsiębiorstwie następujące pliki można przenieść do systemów, na których ma być uruchomiony program IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging:

- V 9.3.0 V 9.3.0 `bcpkix-jdk15to18.jar` "4" na stronie 95
- V 9.3.5 `bcpkix-jdk18on.jar` "3" na stronie 95
- V 9.3.0 V 9.3.0 `bcprov-jdk15to18.jar` "4" na stronie 95
- V 9.3.5 `bcprov-jdk18on.jar` "3" na stronie 95
- V 9.3.0 V 9.3.0 `bcutil-jdk15to18.jar` "4" na stronie 95
- V 9.3.5 `bcutil-jdk18on.jar` "3" na stronie 95
- JMS 2.0 `com.ibm.mq.allclient.jar` "1" na stronie 95
- V 9.3.0 JM 3.0 V 9.3.0 `com.ibm.mq.jakarta.client.jar` "2" na stronie 95
- V 9.3.3 Removed `com.ibm.mq.traceControl.jar`
- `fscontext.jar`
- V 9.3.3 `jackson-annotations.jar`
- V 9.3.3 `jackson-core.jar`
- V 9.3.3 `jackson-databind.jar`
- `jakarta.jms-api.jar`
- `jms.jar`
- `org.json.jar`
- `providerutil.jar`

**Uwagi:**

1. JMS 2.0 i JMS 1.1
2. Jakarta Messaging 3.0
3. Continuous Delivery z wersji IBM MQ 9.3.5
4. Long Term Support i Continuous Delivery przed IBM MQ 9.3.5

## JMS Pliki JAR

V 9.3.0 V 9.3.0 `jms.jar` zawiera interfejsy JMS 1.1 i JMS 2.0 -mają one nazwę `javax.jms.*`.

V 9.3.0 JM 3.0 V 9.3.0 `jakarta.jms-api.jar` zawiera interfejsy Jakarta Messaging 3.0 -mają one nazwę `jakarta.jms.*`.

## fscontext.jar i providerutil.jar

Pliki `fscontext.jar` i `providerutil.jar` są wymagane, jeśli aplikacja wykonuje wyszukiwania JNDI przy użyciu kontekstu systemu plików.

## Dostawca zabezpieczeń bouncy Castle i pliki JAR obsługi CMS

Dostawca zabezpieczeń Bouncy Castle i pliki JAR obsługi CMS są wymagane. Więcej informacji na ten temat zawiera serwis Support for non-IBM JREs with AMS.

V 9.3.5 W przypadku produktu Continuous Delivery z produktu IBM MQ 9.3.5wymagane są następujące pliki JAR:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

LTS W systemach Long Term Support i Continuous Delivery przed IBM MQ 9.3.5wymagane są następujące pliki JAR:

- `bcpkix-jdk15to18.jar`
- `bcprov-jdk15to18.jar`
- `bcutil-jdk15to18.jar`

## org.json.jar

Plik `org.json.jar` jest wymagany, jeśli aplikacja IBM MQ classes for JMS używa tabeli definicji kanału klienta w formacie JSON.

## com.ibm.mq.allclient.jar i com.ibm.mq.jakarta.client.jar

Pliki `com.ibm.mq.allclient.jar` i `com.ibm.mq.jakarta.client.jar` zawierają klasy IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging, IBM MQ classes for Javaoraz PCF i Headers. Jeśli ten plik JAR zostanie przeniesiony do nowego położenia, należy upewnić się, że zostały wykonane kroki w celu zachowania tego nowego położenia w nowych pakietach poprawek produktu IBM MQ . Należy również upewnić się, że użycie plików jest znane działowi wsparcia IBM w przypadku uzyskania poprawki tymczasowej.

Aby określić wersję plików `com.ibm.mq.allclient.jar` i `com.ibm.mq.jakarta.client.jar`, użyj następującej komendy:

V 9.3.0 JM 3.0 V 9.3.0

```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
java -jar com.ibm.mq.allclient.jar
```

W poniższym przykładzie przedstawiono przykładowe dane wyjściowe tej komendy:

```
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
```

## jackson-annotations.jar, jackson-core.jar i jackson-databind.jar

V 9.3.3

Trzy pliki JAR Jackson są wymagane, jeśli aplikacja IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging tworzy bezpieczne połączenia TLS z menedżerem kolejek.

*Ustawianie zmiennych środowiskowych dla IBM MQ classes for JMS/Jakarta Messaging*
Przed skompilowaniem i uruchomieniem aplikacji IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging ustawienie zmiennej środowiskowej **CLASSPATH** musi zawierać plik archiwum Java (JAR) produktu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging . W zależności od wymagań może być konieczne dodanie innych plików JAR do ścieżki klasy. Aby uruchomić skrypty udostępnione z produktem IBM MQ classes for JMS i produktem IBM MQ classes for Jakarta Messaging, należy ustawić inne zmienne środowiskowe.

## Zanim rozpoczniesz

V 9.3.0 JM 3.0 V 9.3.0 W produkcie IBM MQ 9.3.0produkt Jakarta Messaging 3.0 jest obsługiwany na potrzeby tworzenia nowych aplikacji. Produkt IBM MQ 9.3.0 nadal obsługuje produkt JMS 2.0 dla istniejących aplikacji. Nie jest obsługiwane używanie zarówno interfejsu API Jakarta Messaging 3.0 , jak i interfejsu API JMS 2.0 w tej samej aplikacji. Więcej informacji na ten temat zawiera sekcja Używanie klas IBM MQ dla przesyłania komunikatów JMS/Jakarta.

**Ważne:** Ustawienie opcji Java *-Xbootclasspath* w celu uwzględnienia IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging nie jest obsługiwane.

## O tym zadaniu

Aby skompilować i uruchomić aplikacje IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging , należy użyć ustawienia **CLASSPATH** dla używanej platformy i wersji przesyłania komunikatów Java , jak pokazano w poniższych tabelach. Zamiast używać zmiennej środowiskowej można również określić ścieżkę klasy w komendzie **`java`** .

JMS 2.0 W przypadku systemu IBM MQ classes for JMSustawienie to obejmuje katalog przykładów, który umożliwia kompilowanie i uruchamianie przykładowych aplikacji IBM MQ classes for JMS .

JM 3.0 W systemie IBM MQ classes for Jakarta Messagingprzygotowywane są nowe przykłady.

JM 3.0

*Tabela 7. Ustawienia **CLASSPATH** dla Jakarta Messaging 3.0 do kompilowania i uruchamiania aplikacji IBM MQ classes for Jakarta Messaging*

| Platforma | CLASSPATH ustawienie |
|---|---|
| AIX AIX | CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar: |
| Linux Linux | CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar: |
| IBM i IBM i | CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar: |
| Windows Windows | CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.jakarta.client.jar; |

| *Tabela 7. Ustawienia **CLASSPATH** dla Jakarta Messaging 3.0 do kompilowania i uruchamiania aplikacji IBM MQ classes for Jakarta Messaging (kontynuacja)* | |
|---|---|
| **Platforma** | **CLASSPATH ustawienie** |
| z/OS z/OS | CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar; |

JMS 2.0

| *Tabela 8. Ustawienia **CLASSPATH** dla JMS 2.0 do kompilowania i uruchamiania aplikacji IBM MQ classes for JMS , w tym aplikacji przykładowych* | |
|---|---|
| **Platforma** | **Ustawienie CLASSPATH** |
| AIX AIX | CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: , *MQ_INSTALLATION_PATH*/samp/jms/samples: |
| Linux Linux | CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: , *MQ_INSTALLATION_PATH*/samp/jms/samples: |
| IBM i IBM i | CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: /QIBM/ProdData/mqm/java/samples/jms/samples: |
| Windows Windows | CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.allclient.jar; *MQ_INSTALLATION_PATH*\tools\jms\samples; |
| z/OS z/OS | CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: , *MQ_INSTALLATION_PATH*/java/samples/jms/samples: |

W tych tabelach zmienna *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym zainstalowano produkt IBM MQ .

Manifest pliku JAR `com.ibm.mq.jakarta.client.jar` lub `com.ibm.mq.allclient.jar` zawiera odwołania do większości innych plików JAR wymaganych przez aplikacje IBM MQ classes for JMS i dlatego nie trzeba dodawać tych plików JAR do ścieżki klasy. Te pliki JAR obejmują pliki wymagane przez aplikacje korzystające z interfejsu JNDI ( Java Naming Directory Interface ) do pobierania obiektów administrowanych z usługi katalogowej oraz przez aplikacje korzystające z interfejsu JTA ( Java Transaction API).

Dodatkowe pliki JAR należy jednak dołączyć do ścieżki klasy w następujących okolicznościach:

- Jeśli używane są klasy wyjścia kanału, które implementują interfejsy wyjścia kanału zdefiniowane w pakiecie `com.ibm.mq` , zamiast tych zdefiniowanych w pakiecie `com.ibm.mq.exits` , należy dodać plik JAR IBM MQ classes for Java , `com.ibm.mq.jar`, do ścieżki klasy.
- Jeśli aplikacja używa interfejsu JNDI do pobierania obiektów administrowanych z usługi katalogowej, należy również dodać następujące pliki JAR do ścieżki klasy:
  - `fscontext.jar`
  - `providerutil.jar`
- Jeśli aplikacja używa agenta JTA, należy również dodać do ścieżki klasy łańcuch `jta.jar` .

**Uwaga:** Te dodatkowe pliki JAR są wymagane tylko do kompilowania aplikacji, a nie do ich uruchamiania.

Skrypty udostępnione z produktem IBM MQ classes for JMS i produktem IBM MQ classes for Jakarta Messaging używają następujących zmiennych środowiskowych:

**ŚCIEŻKA_DANYCH MQ_JAVA_data**
Ta zmienna środowiskowa określa katalog dla danych wyjściowych dziennika i śledzenia.

**ŚCIEŻKA_INSTALACJI_MQJAVA**
Ta zmienna środowiskowa określa katalog, w którym zainstalowano produkt IBM MQ classes for JMS .

**ŚCIEŻKA_BIBLIOTEKI_MQQ**
Ta zmienna środowiskowa określa katalog, w którym są przechowywane biblioteki produktu IBM MQ classes for JMS , tak jak to pokazano w poprzednich tabelach.

## Procedura

- Windows
  W systemie Windowspo zainstalowaniu produktu IBM MQuruchom komendę **setmqenv**.
  Jeśli ta komenda nie zostanie uruchomiona jako pierwsza, po wywołaniu komendy **dspmqver** może zostać wyświetlony następujący komunikat o błędzie:

  ```
  AMQ8351: IBM MQ
  lub składnik IBM MQ JRE nie został zainstalowany.
  ```

  **Uwaga:** Tego komunikatu należy się spodziewać, jeśli nie zainstalowano środowiska IBM MQ Java runtime environment (JRE) (patrz sekcja Sprawdzanie wymagań wstępnych dotyczących dodatkowych opcji systemu Windows).
- Linux AIX
  W systemach AIX and Linux ustaw samodzielnie zmienne środowiskowe:

  JMS 2.0 W przypadku systemu JMS 2.0należy użyć jednego z następujących skryptów, aby ustawić zmienne środowiskowe:

  – Jeśli używana jest 32-bitowa maszyna JVM, należy użyć skryptu `setjmsenv`.
  – Jeśli w systemie AIX lub Linux używana jest 64-bitowa maszyna JVM, należy użyć skryptu `setjmsenv64`.

  JM 3.0 W przypadku systemu Jakarta Messaging 3.0należy użyć jednego z następujących skryptów, aby ustawić zmienne środowiskowe:

  – Jeśli używana jest 32-bitowa maszyna JVM, należy użyć skryptu `setjms30env`.
  – Jeśli używana jest 64-bitowa maszyna JVM, należy użyć skryptu `setjms30env64`.

  Te skrypty znajdują się w katalogu *MQ_INSTALLATION_PATH*`/java/bin` , gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym zainstalowano produkt IBM MQ .

  Skryptów tych można używać na wiele sposobów. Skryptu można użyć jako podstawy do ustawienia wymaganych zmiennych środowiskowych, jak pokazano w tabeli, lub dodać je do pliku `.profile` za pomocą edytora tekstu. Jeśli używana jest konfiguracja nietypowa, zmodyfikuj zawartość skryptu zgodnie z potrzebami. Alternatywnie można uruchomić skrypt w każdej sesji, z której mają być uruchamiane skrypty startowe JMS . Po wybraniu tej opcji należy uruchomić skrypt w każdym oknie powłoki podczas procesu weryfikacji w systemie JMS :

  – JMS 2.0 W systemie JMS 2.0wpisz `. ./setjmsenv` lub `../setjmsenv64`.
  – JM 3.0 W systemie Jakarta Messaging 3.0wpisz `. ./setjms30env` lub `. ./setjms30env64`.

  IBM i W systemie IBM inależy ustawić zmienną środowiskową **QIBM_MULTI_THREADED** na wartość Y. Aplikacje wielowątkowe można następnie uruchamiać w taki sam sposób, jak aplikacje jednowątkowe. Więcej informacji na ten temat zawiera sekcja Konfigurowanie produktu IBM MQ z produktami Java i JMS.

**Zadania pokrewne**
“Korzystanie z przykładowych aplikacji IBM MQ classes for JMS” na stronie 124

Przykładowe aplikacje IBM MQ classes for JMS udostępniają przegląd wspólnych funkcji interfejsu API języka JMS . Można ich używać do weryfikowania konfiguracji serwera instalacji i przesyłania komunikatów oraz do tworzenia własnych aplikacji.

**Odsyłacze pokrewne**

"Skrypty udostępnione z produktem IBM MQ classes for JMS/Jakarta Messaging" na stronie 127
Udostępniono szereg skryptów ułatwiających wykonywanie typowych zadań, które muszą być wykonywane podczas korzystania z produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging.

*Konfigurowanie bibliotek JNI ( Java Native Interface)*

Aplikacje IBM MQ classes for JMS , które łączą się z menedżerem kolejek przy użyciu transportu powiązań lub łączą się z menedżerem kolejek przy użyciu transportu klienta i używają programów obsługi wyjścia kanału napisanych w językach innych niż Java, muszą być uruchomione w środowisku umożliwiającym dostęp do bibliotek rodzimego interfejsu języka Java (JNI).

## Zanim rozpoczniesz

Więcej informacji na temat używania środowiska WebSphere Application Server zawiera sekcja Konfigurowanie dostawcy przesyłania komunikatów produktu IBM MQ przy użyciu informacji o bibliotekach rodzimych .

## O tym zadaniu

Aby skonfigurować to środowisko, należy skonfigurować ścieżkę do biblioteki środowiska w taki sposób, aby maszyna JVM ( Java Virtual Machine ) mogła załadować bibliotekę mqjbnd przed uruchomieniem aplikacji IBM MQ classes for JMS .

IBM MQ udostępnia dwie biblioteki JNI ( Java Native Interface):

**mqjbnd**
Ta biblioteka jest używana przez aplikacje, które łączą się z menedżerem kolejek przy użyciu transportu powiązań. Udostępnia on interfejs między programem IBM MQ classes for JMS i menedżerem kolejek. Do nawiązania połączenia z dowolnym menedżerem kolejek produktu IBM MQ 9.3 (lub wcześniejszym) można użyć biblioteki mqjbnd zainstalowanej z produktem IBM MQ 9.3 .

**mqjexitstub02**
Biblioteka mqjexitstub02 jest ładowana przez produkt IBM MQ classes for JMS , gdy aplikacja nawiązuje połączenie z menedżerem kolejek przy użyciu transportu klienta i używa programu obsługi wyjścia kanału napisanego w języku innym niż Java.

Na niektórych platformach produkt IBM MQ instaluje 32-bitowe i 64-bitowe wersje tych bibliotek JNI. Położenie bibliotek dla każdej platformy przedstawia Tabela 1.

*Tabela 9. Położenie bibliotek IBM MQ classes for JMS dla każdej platformy*

| **Platforma** | **Katalog zawierający biblioteki IBM MQ classes for JMS** |
|---|---|
| AIX<br>Linux<br>(platformy POWER, x86-64 i zSeries s390x ) | *MQ_INSTALLATION_PATH*/java/lib (biblioteki 32-bitowe)<br>*MQ_INSTALLATION_PATH*/java/lib64 (biblioteki 64-bitowe) |
| Windows | *MQ_INSTALLATION_PATH*\java\lib (biblioteki 32-bitowe)<br>*MQ_INSTALLATION_PATH*\java\lib64 (biblioteki 64-bitowe) |

| *Tabela 9. Położenie bibliotek IBM MQ classes for JMS dla każdej platformy (kontynuacja)* | |
|---|---|
| **Platforma** | **Katalog zawierający biblioteki IBM MQ classes for JMS** |
| z/OS | *MQ_INSTALLATION_PATH*/java/lib<br>(biblioteki 31-bitowe i 64-bitowe) |

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

**Uwaga:** z/OS W systemie z/OSmożna użyć 31-bitowej lub 64-bitowej maszyny Java Virtual Machine (JVM). Nie trzeba określać, które biblioteki JNI mają być używane; produkt IBM MQ classes for JMS może samodzielnie określić, które biblioteki JNI mają zostać załadowane.

## Procedura

1. Skonfiguruj właściwość **`java.library.path`** maszyny JVM, którą można wykonać na dwa sposoby:
   - Podając argument maszyny JVM, jak pokazano w poniższym przykładzie:

     ```
     -Djava.library.path=path_to_library_directory
     ```

     Linux Na przykład w przypadku 64-bitowej maszyny JVM w systemie Linux dla domyślnej instalacji lokalizacji należy podać:

     ```
     -Djava.library.path=/opt/mqm/java/lib64
     ```

   - Skonfigurowanie środowiska powłoki w taki sposób, aby maszyna JVM ustawiała własny plik `java.library.path`. Ta ścieżka różni się w zależności od platformy i miejsca, w którym zainstalowano produkt IBM MQ. Na przykład w przypadku 64-bitowej maszyny JVM i domyślnego miejsca instalacji produktu IBM MQ można użyć następujących ustawień:

     AIX
     ```
     export LIBPATH=/usr/mqm/java/lib64:$LIBPATH
     ```

     Linux
     ```
     export LD_LIBRARY_PATH=/opt/mqm/java/lib64:$LD_LIBRARY_PATH
     ```

     Windows
     ```
     set PATH=C:\Program Files\IBM\MQ\java\lib64;%PATH%
     ```

   Poniżej przedstawiono przykład stosu wyjątków, który jest widoczny, gdy środowisko nie zostało poprawnie skonfigurowane:

```
Przyczyna: com.ibm.mq.jmqi.local.LocalMQ$4: CC=2;RC=2495;
AMQ8598: Załadowanie rodzimej biblioteki JNI produktu WebSphere MQ nie powiodło się: mqjbnd.
    w com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1268)
    w com.ibm.mq.jmqi.local.LocalMQ$1.run(LocalMQ.java:309)
    w java.security.AccessController.doPrivileged(AccessController.java:400)
    w com.ibm.mq.jmqi.local.LocalMQ.initialise_inner(LocalMQ.java:259)
    w com.ibm.mq.jmqi.local.LocalMQ.initialise(LocalMQ.java:221)
    w com.ibm.mq.jmqi.local.LocalMQ. < init> (LocalMQ.java:1350)
    w com.ibm.mq.jmqi.local.LocalServer. < init> (LocalServer.java:230)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)

w sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:86)

    at
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.jav
a:58)
    w java.lang.reflect.Constructor.newInstance(Constructor.java:542)
    w com.ibm.mq.jmqi.JmqiEnvironment.getInstance(JmqiEnvironment.java:706)
    w com.ibm.mq.jmqi.JmqiEnvironment.getMQI(JmqiEnvironment.java:640)

w com.ibm.msg.client.wmq.factories.WMQConnectionFactory.createV7ProviderConnection(WMQConnectio
nFactory.java:8437)
```

```
    ... 7 więcej
Przyczyna: java.lang.UnsatisfiedLinkError: mqjbnd (Nie znaleziono w java.library.path)
    w java.lang.ClassLoader.loadLibraryWithPath(ClassLoader.java:1235)
    w java.lang.ClassLoader.loadLibraryWithClassLoader(ClassLoader.java:1205)
    w java.lang.System.loadLibrary(System.java:534)
    w com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1240)
    ... 20 więcej
```

2. Po skonfigurowaniu środowiska 32-bitowego lub 64-bitowego uruchom aplikację IBM MQ classes for JMS za pomocą komendy:

   ```
   java application-name
   ```

   gdzie *nazwa-aplikacji* jest nazwą aplikacji IBM MQ classes for JMS , która ma zostać uruchomiona.

   Wyjątek zawierający IBM MQ kod przyczyny 2495 (MQRC_MODULE_NOT_FOUND) jest zgłaszany przez IBM MQ classes for JMS , jeśli:

   - Aplikacja IBM MQ classes for JMS jest uruchamiana w 32-bitowym systemie Java runtime environment, a środowisko 64-bitowe zostało skonfigurowane dla IBM MQ classes for JMS, ponieważ 32-bitowy system Java runtime environment nie może załadować 64-bitowej biblioteki rodzimej Java .
   - Aplikacja IBM MQ classes for JMS jest uruchamiana w 64-bitowym systemie Java runtime environment, a środowisko 32-bitowe zostało skonfigurowane dla IBM MQ classes for JMS, ponieważ 64-bitowy system Java runtime environment nie może załadować 32-bitowej biblioteki rodzimej Java .

*Plik konfiguracyjny IBM MQ classes for JMS/Jakarta Messaging*
Pliki konfiguracyjne produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging określają właściwości, które są używane do konfigurowania produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging.

**Uwaga:** Właściwości zdefiniowane w pliku konfiguracyjnym można również ustawić jako właściwości systemowe maszyny JVM. Jeśli właściwość jest ustawiona zarówno w pliku konfiguracyjnym, jak i jako właściwość systemowa, właściwość systemowa ma pierwszeństwo. Dlatego, jeśli jest to wymagane, można nadpisać dowolną właściwość w pliku konfiguracyjnym, podając ją jako właściwość systemową w komendzie **java** .

Plik konfiguracyjny produktu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging ma format standardowego pliku właściwości Java . Przykładowy plik konfiguracyjny o nazwie `jms.config` znajduje się w podkatalogu `bin` katalogu instalacyjnego IBM MQ classes for JMS . Plik ten zawiera wszystkie obsługiwane właściwości i ich wartości domyślne.

Można wybrać nazwę i położenie pliku konfiguracyjnego serwera IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging . Podczas uruchamiania aplikacji należy użyć komendy **java** w następującym formacie:

```
java -Dcom.ibm.msg.client.config.location= config_file_url application_name
```

W komendzie *config_file_url* to adres URL (URL) określający nazwę i położenie pliku konfiguracyjnego IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging . Obsługiwane są następujące typy adresów URL: http, file, ftp i jar.

Poniżej przedstawiono przykład komendy **java** :

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config MyAppClass
```

Ta komenda identyfikuje plik konfiguracyjny IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging jako plik `D:\mydir\mjms.config` w lokalnym systemie Windows .

Po uruchomieniu aplikacji produkt IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging odczytuje treść pliku konfiguracyjnego i zapisuje określone właściwości w wewnętrznej składnicy właściwości. Jeśli komenda **java** nie zidentyfikuje pliku konfiguracyjnego lub jeśli nie można znaleźć

pliku konfiguracyjnego, program IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging użyje wartości domyślnych dla wszystkich właściwości.

Plik konfiguracyjny produktu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging może być używany z dowolnym obsługiwanym transportem między aplikacją a menedżerem kolejek lub brokerem.

## Nadpisywanie właściwości określonych w pliku konfiguracyjnym IBM MQ MQI client

Plik konfiguracyjny IBM MQ MQI client może również określać właściwości, które są używane do konfigurowania produktu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging. Jednak właściwości określone w pliku konfiguracyjnym IBM MQ MQI client mają zastosowanie tylko wtedy, gdy aplikacja nawiązuje połączenie z menedżerem kolejek w trybie klienta.

Jeśli jest to wymagane, można nadpisać dowolny atrybut w pliku konfiguracyjnym IBM MQ MQI client , określając go jako właściwość w pliku konfiguracyjnym IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging . Aby nadpisać atrybut w pliku konfiguracyjnym IBM MQ MQI client , należy użyć wpisu w pliku konfiguracyjnym IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging w następującym formacie:

```
com.ibm.mq.cfg. stanza. propName = propValue
```

Zmienne we wpisie mają następujące znaczenie:

***sekcja***
Nazwa sekcji w pliku konfiguracyjnym IBM MQ MQI client , która zawiera atrybut

***propName***
Nazwa atrybutu określona w pliku konfiguracyjnym IBM MQ MQI client

***propValue***
Wartość właściwości, która nadpisuje wartość atrybutu określoną w pliku konfiguracyjnym IBM MQ MQI client .

Alternatywnie można nadpisać atrybut w pliku konfiguracyjnym IBM MQ MQI client , określając właściwość jako właściwość systemową w komendzie **java** . Użyj poprzedniego formatu, aby określić właściwość jako właściwość systemową.

Tylko następujące atrybuty w pliku konfiguracyjnym IBM MQ MQI client dotyczą produktu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging. Jeśli zostaną podane lub nadpisane inne atrybuty, nie będzie to miało żadnego efektu. W szczególności należy zauważyć, że `ChannelDefinitionFile` i `ChannelDefinitionDirectory` w sekcji CHANNELS pliku konfiguracyjnego klienta nie są używane. Szczegółowe informacje na temat używania tabeli definicji kanału klienta z produktem IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messagingzawiera sekcja “Używanie tabeli definicji kanału klienta z produktem IBM MQ classes for JMS” na stronie 292 .

*Tabela 10. W której sekcji pliku konfiguracyjnego klienta znajduje się atrybut*

| **sekcja** | **Atrybut** |
|---|---|
| Sekcja CHANNELS w pliku konfiguracyjnym klienta | Put1DefaultAlwaysSync |
| Sekcja CHANNELS w pliku konfiguracyjnym klienta | DefRecon |
| Sekcja CHANNELS w pliku konfiguracyjnym klienta | ReconDelay |
| Sekcja CHANNELS w pliku konfiguracyjnym klienta | PasswordProtection |
| ClientExit-sekcja ścieżki pliku konfiguracyjnego klienta | ExitsDefaultPath |
| ClientExit-sekcja ścieżki pliku konfiguracyjnego klienta | ExitsDefaultPath64 |
| ClientExit-sekcja ścieżki pliku konfiguracyjnego klienta | JavaExitsClasspath |

| *Tabela 10. W której sekcji pliku konfiguracyjnego klienta znajduje się atrybut (kontynuacja)* | |
|---|---|
| **sekcja** | **Atrybut** |
| Sekcja JMQI pliku konfiguracyjnego klienta | useMQCSPauthentication |
| SekcjaMessageBuffer pliku konfiguracyjnego klienta | MaximumSize |
| SekcjaMessageBuffer pliku konfiguracyjnego klienta | PurgeTime |
| SekcjaMessageBuffer pliku konfiguracyjnego klienta | UpdatePercentage |
| Sekcja TCP pliku konfiguracyjnego klienta | ClntRcvBufSize |
| Sekcja TCP pliku konfiguracyjnego klienta | ClntSndBufSize |
| Sekcja TCP pliku konfiguracyjnego klienta | Limit_czasu_połączenia |
| Sekcja TCP pliku konfiguracyjnego klienta | KeepAlive |

Więcej informacji na temat konfiguracji IBM MQ MQI client zawiera plik konfiguracyjny IBM MQ MQI client , `mqclient.ini`

*Korzystanie ze standardowego śledzenia środowiska Java do konfigurowania śledzenia JMS*
Użyj sekcji Java Standard Environment Trace Settings (Standardowe ustawienia śledzenia środowiska), aby skonfigurować narzędzie śledzenia systemu IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging .

**`com.ibm.msg.client.commonservices.trace.outputName` = *traceOutputnazwa***
*`traceOutputName`* jest nazwą katalogu i pliku, do którego są wysyłane dane wyjściowe śledzenia.

Domyślnie informacje śledzenia są zapisywane w pliku śledzenia w bieżącym katalogu roboczym aplikacji. Nazwa pliku śledzenia zależy od środowiska, w którym działa aplikacja:

- V 9.3.0 JM 3.0 V 9.3.0 Jeśli w produkcie IBM MQ 9.3.0aplikacja załadowała plik IBM MQ classes for Jakarta Messaging z przemieszczalnego pliku JAR `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0) lub plik IBM MQ classes for JMS z przemieszczalnego pliku JAR `com.ibm.mq.allclient.jar` (JMS 2.0), dane śledzenia są zapisywane w pliku o nazwie `mqjavaclient_%PID%.cl%u.trc`.
- W systemach IBM MQ 9.1.5 i IBM MQ 9.1.0 Fix Pack 5:
  - Jeśli aplikacja załadowała plik IBM MQ classes for JMS z przemieszczalnego pliku JAR `com.ibm.mq.allclient.jar`, dane śledzenia są zapisywane w pliku o nazwie `mqjavaclient_%PID%.cl%u.trc`.
  - Jeśli aplikacja załadowała plik IBM MQ classes for JMS z pliku JAR `com.ibm.mqjms.jar`, dane śledzenia są zapisywane w pliku o nazwie `mqjava_%PID%.cl%u.trc`.
- W systemie IBM MQ 9.0.0 Fix Pack 2:
  - Jeśli aplikacja załadowała plik IBM MQ classes for JMS z przemieszczalnego pliku JAR `com.ibm.mq.allclient.jar`, dane śledzenia są zapisywane w pliku o nazwie `mqjavaclient_%PID%.trc`.
  - Jeśli aplikacja załadowała plik IBM MQ classes for JMS z pliku JAR `com.ibm.mqjms.jar`, dane śledzenia są zapisywane w pliku o nazwie `mqjava_%PID%.trc`.
- W przypadku produktu IBM MQ classes for JMS dla systemu IBM MQ 9.0.0 Fix Pack 1 lub wcześniejszego dane śledzenia są zapisywane w pliku o nazwie `mqjms_%PID%.trc`.

gdzie *%PID%* jest identyfikatorem procesu śledzonej aplikacji, a *%u* jest unikalną liczbą w celu odróżnienia plików wątków uruchamiających śledzenie w różnych programach ładujących klasy Java .

Jeśli identyfikator procesu jest niedostępny, generowana jest liczba losowa z przedrostkiem `f`. Aby dołączyć identyfikator procesu do określonej nazwy pliku, należy użyć łańcucha `%PID%`.

Jeśli zostanie podany katalog alternatywny, musi on istnieć i użytkownik musi mieć uprawnienia do zapisu w tym katalogu. Jeśli użytkownik nie ma uprawnień do zapisu, dane wyjściowe śledzenia są zapisywane w pliku `System.err`.

**com.ibm.msg.client.commonservices.trace.include = *includeList***
*includeList* to lista śledzonych pakietów i klas lub wartości specjalnych `ALL` lub `NONE`.

Nazwy pakietów lub klas należy oddzielać średnikiem `;`. *includeList* przyjmuje wartość domyślną `ALL`i śledzi wszystkie pakiety i klasy w programie IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging.

**Uwaga:** Można dołączyć pakiet, ale następnie wykluczyć podpakiety tego pakietu. Na przykład, jeśli zostanie dołączony pakiet `a.b` i zostanie wykluczony pakiet `a.b.x`, dane śledzenia będą obejmować wszystkie elementy `a.b.y` i `a.b.z`, ale nie `a.b.x` ani `a.b.x.1`.

**com.ibm.msg.client.commonservices.trace.exclude = *excludeList***
*excludeList* to lista pakietów i klas, które nie są śledzone, wartości specjalne `ALL` lub `NONE`.

Nazwy pakietów lub klas należy oddzielać średnikiem `;`. Wartością domyślną parametru *excludeList* jest `NONE`, co oznacza, że żadne pakiety i klasy w programie IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging nie są śledzone.

**Uwaga:** Można wykluczyć pakiet, ale następnie dołączyć podpakiety tego pakietu. Na przykład, jeśli pakiet `a.b` zostanie wykluczony i zostanie dołączony pakiet `a.b.x`, dane śledzenia będą obejmować wszystkie elementy `a.b.x` i `a.b.x.1`, ale nie `a.b.y` ani `a.b.z`.

Każdy pakiet lub klasa, która jest określona na tym samym poziomie, jako zarówno uwzględniona, jak i wykluczona, jest włączana.

**com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayB***
*maxArrayBytes* to maksymalna liczba bajtów, które są śledzone z dowolnej tablicy bajtów.

Jeśli parametr *maxArrayBytes* jest ustawiony na dodatnią liczbę całkowitą, ogranicza on liczbę bajtów w tablicy bajtów, które są zapisywane w pliku śledzenia. Obcina ona tablicę bajtów po zapisaniu pliku *maxArrayBytes* . Ustawienie parametru *maxArrayBytes* zmniejsza wielkość wynikowego pliku śledzenia i zmniejsza wpływ śledzenia na wydajność aplikacji.

Wartość `0` tej właściwości oznacza, że do pliku śledzenia nie jest wysyłana żadna zawartość żadnej tablicy bajtów.

Wartością domyślną jest `-1`, co powoduje usunięcie limitu liczby bajtów w tablicy bajtów, które są wysyłane do pliku śledzenia.

**com.ibm.msg.client.commonservices.trace.limit = *maxTraceB***
*maxTraceBytes* to maksymalna liczba bajtów, które są zapisywane w pliku wyjściowym śledzenia.

Produkt *maxTraceBytes* współpracuje z produktem *traceCycles*. Jeśli liczba bajtów danych śledzenia jest bliska limitu, plik jest zamykany i uruchamiany jest nowy plik danych wyjściowych śledzenia.

Wartość `0` oznacza, że plik wyjściowy śledzenia ma zerową długość. Wartością domyślną jest `-1`, co oznacza, że ilość danych zapisywanych w pliku wyjściowym śledzenia jest nieograniczona.

**com.ibm.msg.client.commonservices.trace.count = *traceCycles***
*traceCycles* to liczba plików wyjściowych śledzenia, które mają być cyklicznie przeglądane.

Jeśli bieżący plik danych wyjściowych śledzenia osiągnie limit określony przez parametr *maxTraceBytes*, plik zostanie zamknięty. Dalsze dane wyjściowe śledzenia są zapisywane do następnego pliku wyjściowego śledzenia w kolejności. Każdy plik wyjściowy śledzenia jest wyróżniany przyrostkiem liczbowym dołączonym do nazwy pliku. Bieżący lub najnowszy plik wyjściowy śledzenia to `mqjms.trc.0`, a następny plik wyjściowy śledzenia to `mqjms.trc.1`. Starsze pliki śledzenia są zgodne z tym samym wzorcem numeracji aż do limitu.

Wartością domyślną parametru *traceCycles* jest 1. Jeśli *traceCycles* ma wartość 1, gdy bieżący plik danych wyjściowych śledzenia osiągnie maksymalną wielkość, plik zostanie zamknięty i usunięty. Zostanie uruchomiony nowy plik wyjściowy śledzenia o takiej samej nazwie. Dlatego w danym momencie istnieje tylko jeden plik wyjściowy śledzenia.

**com.ibm.msg.client.commonservices.trace.parameter = *traceParameters***
Parametr *traceParameters* określa, czy parametry metody i zwracane wartości są uwzględniane w danych śledzenia.

Wartością domyślną parametru *traceParameters* jest TRUE. Jeśli parametr *traceParameters* ma wartość FALSE, śledzone są tylko sygnatury metod.

**com.ibm.msg.client.commonservices.trace.startup = *uruchamianie***
Istnieje faza inicjowania IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging , podczas której zasoby są przydzielane. Główne narzędzie śledzenia jest inicjowane w fazie przydzielania zasobów.

Jeśli parametr *startup* ma wartość TRUE, używane jest śledzenie przy uruchamianiu. Informacje śledzenia są generowane natychmiast i obejmują konfigurację wszystkich komponentów, w tym samego narzędzia śledzenia. Informacje śledzenia uruchamiania mogą być używane do diagnozowania problemów z konfiguracją. Informacje śledzenia uruchamiania są zawsze zapisywane w pliku `System.err`.

Wartością domyślną parametru *startup* jest FALSE.

Parametr *startup* jest sprawdzany przed zakończeniem inicjowania. Z tego powodu właściwość tę należy określić tylko w wierszu komend jako właściwość systemową Java . Nie należy go podawać w pliku konfiguracyjnym IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging .

**com.ibm.msg.client.commonservices.trace.compress = *compressedTrace***
Ustaw parametr *compressedTrace* na wartość TRUE , aby skompresować dane wyjściowe śledzenia.

Wartością domyślną parametru *compressedTrace* jest FALSE.

Jeśli parametr *compressedTrace* ma wartość TRUE, dane wyjściowe śledzenia są kompresowane. Domyślna nazwa pliku wyjściowego śledzenia ma rozszerzenie `.trz`. Jeśli kompresja jest ustawiona na FALSE(wartość domyślna), plik ma rozszerzenie `.trc` , aby wskazać, że jest nieskompresowany. Jeśli jednak nazwa pliku dla danych wyjściowych śledzenia została określona w parametrze *traceOutputName* , zostanie użyta ta nazwa; do pliku nie zostanie zastosowany przyrostek.

Skompresowane dane wyjściowe śledzenia są mniejsze niż nieskompresowane. Ponieważ liczba operacji we/wy jest mniejsza, można ją zapisać szybciej niż w przypadku nieskompresowanych danych śledzenia. Śledzenie skompresowane ma mniejszy wpływ na wydajność produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging niż śledzenie nieskompresowane.

Jeśli zostaną ustawione opcje *maxTraceBytes* i *traceCycles* , w miejsce wielu plików tekstowych zostanie utworzonych wiele skompresowanych plików śledzenia.

Jeśli plik IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging kończy się w sposób niekontrolowany, skompresowany plik śledzenia może nie być poprawny. Z tego powodu kompresja śledzenia może być używana tylko wtedy, gdy system IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging zostanie zamknięty w sposób kontrolowany. Kompresji śledzenia należy używać tylko wtedy, gdy badane problemy nie powodują nieoczekiwanego zatrzymania maszyny JVM. Kompresji śledzenia nie należy używać podczas diagnozowania problemów, które mogą spowodować zamknięcie systemu `System.Halt()` lub nieprawidłowe, niekontrolowane zakończenia działania maszyny JVM.

**com.ibm.msg.client.commonservices.trace.level = *traceLevel***
*traceLevel* określa poziom filtrowania dla śledzenia. Zdefiniowane poziomy śledzenia są następujące:

- `TRACE_NONE`: `0`
- `TRACE_EXCEPTION`: `1`
- `TRACE_WARNING`: `3`

- TRACE_INFO: 6
- TRACE_ENTRYEXIT: 8
- TRACE_DATA: 9
- TRACE_ALL: Integer.MAX_VALUE

Każdy poziom śledzenia obejmuje wszystkie niższe poziomy. Na przykład, jeśli poziom śledzenia jest ustawiony na TRACE_INFO, do śledzenia zapisywany jest każdy punkt śledzenia o zdefiniowanym poziomie TRACE_EXCEPTION, TRACE_WARNINGlub TRACE_INFO . Wszystkie pozostałe punkty śledzenia są wykluczane.

**com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace***

*standaloneTrace* określa, czy usługa śledzenia klienta IBM MQ JMS jest używana w środowisku WebSphere Application Server .

Jeśli parametr *standaloneTrace* ma wartość TRUE, właściwości śledzenia klienta IBM MQ JMS są używane do określenia konfiguracji śledzenia.

Jeśli parametr *standaloneTrace* ma wartość FALSE, a klient IBM MQ JMS jest uruchomiony w kontenerze WebSphere Application Server , używana jest usługa śledzenia WebSphere Application Server . Generowane informacje śledzenia zależą od ustawień śledzenia serwera aplikacji.

Wartością domyślną parametru *standaloneTrace* jest FALSE.

*Sekcja rejestrowania*
Użyj sekcji Logging, aby skonfigurować narzędzie rejestrowania IBM MQ classes for JMS .

W sekcji Logging (Rejestrowanie) można uwzględnić następujące właściwości:

**com.ibm.msg.client.commonservices.log.outputName = ścieżka**
Nazwa pliku dziennika używanego przez narzędzie rejestrowania IBM MQ classes for JMS . Wartością domyślną jest mqjms.log, która jest zapisywana w bieżącym katalogu roboczym środowiska Java Runtime Environment, w którym działa program IBM MQ classes for JMS .

Właściwość może przyjmować jedną z następujących wartości:

- pojedyncza nazwa ścieżki
- rozdzielana przecinkami lista nazw ścieżek (wszystkie dane są rejestrowane we wszystkich plikach)

Każda nazwa ścieżki może być pełną lub względną nazwą ścieżki lub:

**"stderr" lub "System.err"**
Reprezentuje strumień standardowego wyjścia błędów.

**"stdout" lub "System.out"**
Reprezentuje strumień wyjścia standardowego.

**com.ibm.msg.client.commonservices.log.maxBytes**
Maksymalna liczba bajtów, które są rejestrowane od dowolnego wywołania do danych komunikatu dziennika.

**Dodatnia liczba całkowita**
Dane są zapisywane do tej wartości liczby bajtów na wywołanie dziennika.

**0**
Nie są zapisywane żadne dane.

**-1**
Zapisywane są nieograniczone dane (domyślnie).

**com.ibm.msg.client.commonservices.log.limit**
Maksymalna liczba bajtów, które są zapisywane w dowolnym pliku dziennika (wartość domyślna to 262144).

**Dodatnia liczba całkowita**
Dane są zapisywane do tej wartości w bajtach na plik dziennika.

**0**
Nie są zapisywane żadne dane.

**-1**
Zapisywane są nieograniczone dane.

**com.ibm.msg.client.commonservices.log.count**
Liczba plików dziennika, które mają być cykliczne. Gdy każdy plik osiągnie poziom śledzenia `com.ibm.msg.client.commonservices.trace.limit` , rozpocznie się w następnym pliku, wartością domyślną jest 3.

**Dodatnia liczba całkowita**
Liczba plików do cyklicznego przechodzenia.

**0**
Pojedynczy plik.

*Sekcja środowiska Java SE Specifics*
Sekcja Java SE Specifics służy do konfigurowania właściwości, które są używane, gdy IBM MQ classes for JMS są używane w środowisku Java Standard Edition .

**com.ibm.msg.client.commonservices.j2se.produceJavaCore = PRAWDA|FAŁSZ**
Określa, czy plik JavaCore jest zapisywany natychmiast po wygenerowaniu pliku FDC przez IBM MQ classes for JMS . Jeśli ta opcja ma wartość TRUE, w katalogu roboczym środowiska Java Runtime Environment, w którym działa IBM MQ classes for JMS , tworzony jest plik JavaCore.

**TRUE**
Generowanie modułu podstawowego Java, w zależności od możliwości środowiska wykonawczego Java Runtime Environment.

**FALSE**
Nie generuj modułu podstawowego Java; jest to wartość domyślna.

*IBM MQ Sekcja Properties*
Sekcja IBM MQ Properties (Właściwości) służy do ustawiania właściwości wpływających na sposób interakcji IBM MQ classes for JMS z produktem IBM MQ.

**com.ibm.msg.client.wmq.compat.base.internal.MQQueue.smallMsgsBufferReductionThreshold**
Jeśli aplikacja używająca programu IBM MQ classes for JMS nawiązuje połączenie z menedżerem kolejek produktu IBM MQ przy użyciu trybu migracji dostawcy usługi przesyłania komunikatów produktu IBM MQ , program IBM MQ classes for JMS używa domyślnego buforu o wielkości 4 kB podczas odbierania komunikatów. Jeśli komunikat, który aplikacja próbuje uzyskać, jest większy niż 4 kB, serwer IBM MQ classes for JMS zmienia wielkość buforu, aby była na tyle duża, aby pomieścić komunikat. Po odebraniu kolejnych komunikatów używana jest większa wielkość buforu.

Ta właściwość określa, kiedy wielkość buforu jest zmniejszana do 4 kB. Domyślnie po odebraniu dziesięciu kolejnych komunikatów, które są mniejsze niż większy bufor, wielkość buforu jest zmniejszana z powrotem do 4 kB. Aby przywrócić wielkość buforu do wartości 4 kB za każdym razem, gdy komunikat jest odbierany, należy ustawić tę właściwość na wartość 0.

**0**

Bufor jest zawsze resetowany do wielkości domyślnej.

**10**

Jest to wartość domyślna. Wielkość buforu zostanie zmieniona po dziesiątym komunikacie.

**com.ibm.msg.client.wmq.receiveConversionCCSID**
Jeśli aplikacja używająca programu IBM MQ classes for JMS nawiązuje połączenie z menedżerem kolejek produktu IBM MQ przy użyciu trybu normalnego dostawcy usługi przesyłania komunikatów produktu IBM MQ , właściwość `receiveConversionCCSID` można ustawić w taki sposób, aby przesłonić domyślną wartość identyfikatora CCSID w strukturze MQMD używanej do odbierania komunikatów z menedżera kolejek. Domyślnie deskryptor MQMD zawiera pole CCSID ustawione na

wartość 1208, ale można to zmienić, jeśli na przykład menedżer kolejek nie może dokonać konwersji komunikatów na tę stronę kodową.

Poprawne wartości to dowolny poprawny numer CCSID lub jedna z następujących wartości:

**-1**
Użyj wartości domyślnej platformy.

**1208**
Jest to wartość domyślna.

*Sekcja specyfiki trybu klienta*
Sekcja Tryb klienta służy do określania właściwości, które są używane, gdy program IBM MQ classes for JMS łączy się z menedżerem kolejek korzystającym z transportu CLIENT.

**com.ibm.mq.polling.RemoteRequestEntry**
Określa odstęp czasu odpytywania używany przez program IBM MQ classes for JMS do sprawdzania zerwanych połączeń podczas oczekiwania na odpowiedź z menedżera kolejek.

**Dodatnia liczba całkowita**
Liczba milisekund oczekiwania przed sprawdzeniem. Wartością domyślną jest 10000 lub 10 sekund. Wartość minimalna wynosi 3000, a niższe wartości są traktowane w taki sam sposób, jak ta wartość minimalna.

*Właściwości używane do konfigurowania zachowania klienta JMS*
Te właściwości służą do konfigurowania zachowania klienta JMS .

**com.ibm.mq.jms.SupportMQExtensions PRAWDA|FAŁSZ**
Specyfikacja JMS 2.0 wprowadza zmiany w sposobie działania niektórych zachowań. IBM MQ 8.0 zawiera właściwość `com.ibm.mq.jms.SupportMQExtensions`, którą można ustawić na wartość *TRUE*, aby przywrócić te zmienione zachowania do poprzednich implementacji. Przywrócenie zmienionych zachowań może być konieczne w przypadku aplikacji JMS 2.0 , a także w przypadku niektórych aplikacji, które korzystają z interfejsu API JMS 1.1 , ale działają w oparciu o IBM MQ 8.0 IBM MQ classes for JMS.

**PRAWDA**
Następujące trzy obszary funkcjonalności zostały przywrócone przez ustawienie parametru SupportMQExtensions na wartość TRUE:

**Priorytet komunikatu**
Do komunikatów można przypisać priorytet *0* - *9*. W przypadku wersji wcześniejszych niż JMS 2.0komunikaty mogą również używać wartości *-1*, co oznacza, że używany jest domyślny priorytet kolejki. JMS 2.0 nie zezwala na ustawienie priorytetu komunikatu *-1* . Włączenie opcji `SupportMQExtensions` umożliwia użycie wartości *-1* .

**Identyfikator klienta**
Specyfikacja JMS 2.0 wymaga, aby identyfikatory klientów o wartości innej niż NULL były sprawdzane pod kątem unikalności podczas nawiązywania połączenia. Włączenie opcji `SupportMQExtensions`oznacza, że to wymaganie nie jest brane pod uwagę i że można ponownie wykorzystać identyfikator klienta.

**`NoLocal`**
Specyfikacja JMS 2.0 wymaga, aby po włączeniu tej stałej konsument nie mógł odbierać komunikatów publikowanych przez ten sam identyfikator klienta. Przed produktem JMS 2.0ten atrybut został ustawiony na subskrybencie, aby uniemożliwić mu odbieranie komunikatów publikowanych przez jego własne połączenie. Włączenie opcji `SupportMQExtensions` przywraca to zachowanie do poprzedniej implementacji.

**FAŁSZ**
Zmiany zachowania są zachowywane.

**com.ibm.msg.client.jms.ByteStreamReadOnlyAfterSend= PRAWDA|FAŁSZ**
W produkcie IBM MQ 8.0.0 Fix Pack 2po wysłaniu przez aplikację komunikatu strumienia lub bajtów produkt IBM MQ classes for JMS może ustawić stan wysłanego komunikatu na tylko do odczytu lub tylko do zapisu.

**PRAWDA**
Po wysłaniu obiekty są ustawione jako tylko do odczytu. Ustawienie tej wartości zapewnia zgodność ze specyfikacją JMS 2.0

**FAŁSZ**
Obiekty są ustawiane do zapisu tylko po wysłaniu. Jest to wartość domyślna.

**Pojęcia pokrewne**

“Właściwość SupportMQExtensions” na stronie 338
W specyfikacji JMS 2.0 wprowadzono zmiany w sposobie działania niektórych zachowań. IBM MQ 8.0 i nowsze zawierają właściwość **com.ibm.mq.jms.SupportMQExtensions**, którą można ustawić na wartość TRUE , aby przywrócić poprzednie implementacje tych zmienionych zachowań.

*Konfiguracja STEPLIB dla systemu IBM MQ classes for JMS na platformie z/OS*
W systemie z/OSbiblioteka STEPLIB używana w czasie wykonywania musi zawierać biblioteki SCSQAUTH i SCSQANLE systemu IBM MQ . Biblioteki te należy określić w startowym kodzie JCL lub przy użyciu pliku `.profile` .

W produkcie z/OS UNIX System Servicesmożna je dodać przy użyciu wiersza w pliku `.profile` , jak pokazano w poniższym fragmencie kodu, zastępując łańcuch `thlqual` kwalifikatorem zestawu danych wysokiego poziomu wybranym podczas instalowania produktu IBM MQ:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

W innych środowiskach zwykle konieczna jest edycja kodu JCL uruchamiania w celu uwzględnienia SCSQAUTH i SCSQANLE w konkatenacji STEPLIB:

```
STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
        DD DSN=thlqual.SCSQANLE,DISP=SHR
```

*IBM MQ classes for JMS i narzędzia do zarządzania oprogramowaniem*
Narzędzia do zarządzania oprogramowaniem, takie jak Apache Maven, mogą być używane z systemami IBM MQ classes for JMS oraz IBM MQ classes for Jakarta Messaging.

Wiele dużych organizacji programistycznych używa tych narzędzi do centralnego zarządzania repozytoriami bibliotek innych firm.

Pliki IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging składają się z wielu plików JAR. Podczas tworzenia aplikacji w języku Java za pomocą tego interfejsu API na komputerze, na którym jest opracowywana aplikacja, wymagana jest instalacja serwera IBM MQ , klienta IBM MQ lub klienta IBM MQ o wartości SupportPac .

Aby użyć takiego narzędzia i dodać pliki JAR, które składają się na plik IBM MQ classes for JMS , do repozytorium zarządzanego centralnie, należy pamiętać o następujących kwestiach:

- Repozytorium lub kontener musi być dostępny tylko dla programistów w organizacji. Dystrybucja poza organizacją nie jest dozwolona.
- Repozytorium musi zawierać pełny i spójny zestaw plików JAR z pojedynczej wersji produktu IBM MQ lub pakietu poprawek.
- Użytkownik jest odpowiedzialny za zaktualizowanie repozytorium przy użyciu dowolnej konserwacji udostępnionej przez dział wsparcia IBM .

W repozytorium muszą być zainstalowane następujące pliki JAR:

- JMS 2.0 `com.ibm.mq.allclient.jar` i `jms.jar` są wymagane, jeśli używany jest IBM MQ classes for JMS.
- JM 3.0 `com.ibm.mq.jakarta.client.jar` i `jakarta.jms-api.jar` są wymagane, jeśli używany jest produkt IBM MQ classes for Jakarta Messaging.

- Parametr `fscontext.jar` jest wymagany, jeśli używany jest produkt IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging i uzyskiwany jest dostęp do obiektów administrowanych JMS , które są przechowywane w kontekście JNDI systemu plików.
- `providerutil.jar` jest wymagany, jeśli używany jest serwer IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging i uzyskiwany jest dostęp do obiektów administrowanych JMS , które są przechowywane w kontekście JNDI systemu plików.
- Dostawca zabezpieczeń Bouncy Castle i pliki JAR obsługi CMS są wymagane do obsługi środowisk JRE innych niżIBM . Więcej informacji na ten temat zawiera sekcja Obsługa środowisk JRE firm innych niżIBM.

### *Uruchamianie aplikacji IBM MQ classes for JMS w katalogu Java security manager*

Program IBM MQ classes for JMS można uruchomić z włączoną opcją Java security manager .
Aby pomyślnie uruchomić aplikacje z włączoną opcją Java security manager , należy skonfigurować środowisko Java Virtual Machine (JVM) przy użyciu odpowiedniego pliku konfiguracyjnego strategii.

Najprostszym sposobem utworzenia odpowiedniego pliku definicji strategii jest zmiana pliku konfiguracyjnego strategii dostarczonego wraz ze środowiskiem Java runtime environment (JRE).
W większości systemów plik ten znajduje się w katalogu `lib/security/java.policy` względem katalogu środowiska JRE. Plik konfiguracyjny strategii można edytować przy użyciu preferowanego edytora lub programu narzędzia strategii dostarczanego ze środowiskiem JRE.

## Przykładowy plik konfiguracyjny strategii

Poniżej przedstawiono przykład pliku konfiguracyjnego strategii, który umożliwia pomyślne uruchomienie programu IBM MQ classes for JMS w ramach domyślnego menedżera zabezpieczeń. Ten plik należy dostosować w celu określenia położenia określonych plików i katalogów: *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ , *MQ_DATA_DIRECTORY* reprezentuje położenie katalogu danych produktu MQ , a *QM_NAME* jest nazwą menedżera kolejek, dla którego jest konfigurowany dostęp.

```
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";
};
```

W przykładzie instrukcja `grant` zawiera uprawnienia wymagane przez IBM MQ classes for JMS. Aby użyć tych instrukcji nadawania uprawnień w pliku konfiguracyjnym strategii, może być konieczne zmodyfikowanie nazw ścieżek w zależności od miejsca, w którym zainstalowano produkt IBM MQ classes for JMS i w którym są przechowywane aplikacje.

Przykładowe aplikacje dostarczone wraz z programem IBM MQ classes for JMSoraz skrypty do ich uruchamiania nie włączają menedżera zabezpieczeń.

**Ważne:**

Narzędzie śledzenia IBM MQ classes for JMS wymaga dalszych uprawnień, ponieważ wykonuje dodatkowe zapytania dotyczące właściwości systemu, a także dalsze operacje systemu plików.

Odpowiedni plik strategii bezpieczeństwa szablonu do uruchomienia w menedżerze zabezpieczeń z włączonym śledzeniem znajduje się w katalogu `samples/wmqjava` instalacji produktu IBM MQ pod nazwą `example.security.policy`.

## *Konfiguracja poinstalacyjna dla aplikacji IBM MQ classes for JMS*

W tym temacie opisano uprawnienia wymagane przez aplikacje produktu IBM MQ classes for JMS do uzyskania dostępu do zasobów menedżera kolejek. Wprowadzono również tryby połączenia i opisano sposób konfigurowania menedżera kolejek, aby aplikacje mogły nawiązywać połączenia w trybie klienta.

**Pamiętaj, aby sprawdzić plik readme produktu IBM MQ . Może on zawierać informacje, które zastępują informacje zawarte w tym temacie.**

*Obiekty używane przez JMS , które wymagają autoryzacji dla użytkowników nieuprzywilejowanych*
Użytkownicy bez uprawnień muszą mieć nadaną autoryzację, aby uzyskać dostęp do kolejek używanych przez produkt JMS. Każda aplikacja produktu JMS wymaga autoryzacji do menedżera kolejek, z którym działa.

Szczegółowe informacje na temat kontroli dostępu w produkcie IBM MQzawiera sekcja Konfigurowanie zabezpieczeń.

Aplikacje IBM MQ classes for JMS wymagają uprawnień `connect` i `inq` do menedżera kolejek. Odpowiednie autoryzacje można ustawić za pomocą komendy sterującej **setmqaut** , na przykład:

```
setmqaut -m QM1 -t qmgr -g jmsappsgroup +connect +inq
```

W przypadku domeny punkt z punktem wymagane są następujące uprawnienia:

- Kolejki używane przez obiekty MessageProducer wymagają uprawnienia `put` .
- Kolejki używane przez obiekty MessageConsumer i QueueBrowser muszą mieć uprawnienia `get`, `inqi` `browse` .
- Metoda QueueSession.createTemporaryQueue () wymaga dostępu do kolejki modelowej określonej we właściwości TEMPMODEL obiektu fabryki QueueConnection. Domyślnie ta kolejka modelowa to SYSTEM.TEMP.MODEL.QUEUE.

Jeśli któraś z tych kolejek jest kolejkami aliasowymi, ich kolejki docelowe wymagają uprawnień do zapytań. Jeśli kolejka docelowa jest kolejką klastra, wymagane jest również uprawnienie do przeglądania.

W przypadku domeny publikowania/subskrypcji używane są następujące kolejki, jeśli produkt IBM MQ classes for JMS łączy się z menedżerem kolejek produktu IBM MQ w trybie migracji dostawcy przesyłania komunikatów produktu IBM MQ :

- SYSTEM.JMS.ADMIN.QUEUE
- SYSTEM.JMS.REPORT.QUEUE
- SYSTEM.JMS.MODEL.QUEUE
- SYSTEM.JMS.PS.STATUS.QUEUE
- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.SUBSCRIBER.QUEUE
- SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- SYSTEM.BROKER.CONTROL.QUEUE

Więcej informacji na temat trybu migracji dostawcy usługi przesyłania komunikatów produktu IBM MQ zawiera sekcja Konfigurowanie właściwości produktu JMS **PROVIDERVERSION** .

Dodatkowo, jeśli program IBM MQ classes for JMS łączy się z menedżerem kolejek w tym trybie, każda aplikacja publikująca komunikaty musi mieć dostęp do kolejki strumienia określonej przez fabrykę lub obiekt tematu TopicConnection. Domyślnie jest to kolejka SYSTEM.BROKER.DEFAULT.STREAM.

Jeśli używany jest konsument ConnectionConsumer, adapter zasobów IBM MQ lub dostawca przesyłania komunikatów produktu WebSphere Application Server IBM MQ , może być wymagana dodatkowa autoryzacja.

Kolejki, które mają być odczytywane przez ConnectionConsumer , muszą mieć uprawnienia `get`, `inqi` `browse` . Systemowa kolejka niedostarczonych komunikatów oraz dowolna kolejka wycofanych komunikatów lub kolejka raportów używana przez ConnectionConsumer muszą mieć uprawnienia `put` i `passall` .

Gdy aplikacja używa trybu normalnego dostawcy usługi przesyłania komunikatów produktu IBM MQ do przesyłania komunikatów w trybie publikowania/subskrypcji, korzysta ze zintegrowanej funkcji publikowania/subskrypcji udostępnianej przez menedżer kolejek. Informacje na temat zabezpieczania używanych tematów i kolejek zawiera sekcja Zabezpieczenia publikowania/subskrypcji .

*Tryby połączenia dla IBM MQ classes for JMS*
Aplikacja IBM MQ classes for JMS może nawiązać połączenie z menedżerem kolejek w trybie klienta lub w trybie powiązań. W trybie klienta program IBM MQ classes for JMS łączy się z menedżerem kolejek za pośrednictwem protokołu TCP/IP. W trybie powiązań produkt IBM MQ classes for JMS nawiązuje połączenie bezpośrednio z menedżerem kolejek przy użyciu rodzimego interfejsu języka Java (JNI).

Aplikacja działająca w produkcie WebSphere Application Server w systemie z/OS może nawiązać połączenie z menedżerem kolejek w trybie powiązań lub w trybie klienta, ale aplikacja działająca w dowolnym innym środowisku w systemie z/OS może nawiązać połączenie z menedżerem kolejek tylko w trybie powiązań. Aplikacja działająca na dowolnej innej platformie może nawiązać połączenie z menedżerem kolejek w trybie powiązań lub w trybie klienta.

Z bieżącym menedżerem kolejek można używać bieżącej lub wcześniejszej obsługiwanej wersji produktu IBM MQ classes for JMS oraz bieżącej lub wcześniejszej obsługiwanej wersji menedżera kolejek z bieżącą wersją produktu IBM MQ classes for JMS. W przypadku mieszania różnych wersji funkcja jest ograniczona do poziomu wcześniejszej wersji.

W poniższych sekcjach opisano szczegółowo każdy z trybów połączenia.

## Tryb klienta

Aby nawiązać połączenie z menedżerem kolejek w trybie klienta, aplikacja programu IBM MQ classes for JMS może działać w tym samym systemie, w którym działa menedżer kolejek, lub w innym systemie. W każdym przypadku program IBM MQ classes for JMS łączy się z menedżerem kolejek za pośrednictwem protokołu TCP/IP.

## Tryb powiązań

Aby nawiązać połączenie z menedżerem kolejek w trybie powiązań, aplikacja IBM MQ classes for JMS musi działać w tym samym systemie, w którym działa menedżer kolejek.

Produkt IBM MQ classes for JMS nawiązuje połączenie bezpośrednio z menedżerem kolejek przy użyciu rodzimego interfejsu języka Java (JNI). Aby użyć transportu powiązań, należy uruchomić plik IBM MQ classes for JMS w środowisku, które ma dostęp do bibliotek rodzimego interfejsu produktu IBM MQ Java . Więcej informacji na ten temat zawiera sekcja “Konfigurowanie bibliotek JNI ( Java Native Interface)” na stronie 100 .

IBM MQ classes for JMS obsługuje następujące wartości opcji *ConnectOption*:

- MQCNO_FASTPATH_BINDING,
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_IZOLOWANE_POWIĄZANIE
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR (mqcno_restrict_conn_q_mgr)

Aby zmienić opcje połączenia używane przez konsolę IBM MQ classes for JMS, należy zmodyfikować właściwość fabryki połączeń CONNOPT.

Więcej informacji na temat opcji połączenia zawiera sekcja “Nawiązywanie połączenia z menedżerem kolejek przy użyciu wywołania MQCONNX” na stronie 761

Aby używać transportu powiązań, używane środowisko wykonawcze Java musi obsługiwać identyfikator kodowanego zestawu znaków (CCSID) menedżera kolejek, z którym łączy się program IBM MQ classes for JMS .

Szczegółowe informacje na temat określania identyfikatorów CCSID obsługiwanych przez środowisko Java Runtime Environment można znaleźć w sekcji IBM MQ FDC z identyfikatorem sondy 21 wygenerowanej podczas używania klas IBM MQ V7 dla systemu Java lub klas IBM MQ V7 dla systemu JMS.

*Konfigurowanie menedżera kolejek w taki sposób, aby aplikacje IBM MQ classes for JMS mogły nawiązywać połączenia w trybie klienta*
Aby skonfigurować menedżer kolejek w taki sposób, aby aplikacje IBM MQ classes for JMS mogły nawiązywać połączenia w trybie klienta, należy utworzyć definicję kanału połączenia z serwerem i uruchomić program nasłuchujący.

## Tworzenie definicji kanału połączenia z serwerem

Na wszystkich platformach do utworzenia definicji kanału połączenia z serwerem można użyć komendy MQSC DEFINE CHANNEL. Zapoznaj się z poniższym przykładem:

```
DEFINE CHANNEL(JAVA.CHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

IBM i W systemie IBM imożna użyć komendy CL CRTMQMCHL, tak jak w poniższym przykładzie:

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN)
TRPTYPE(*TCP)
MQMNAME(QMGRNAME)
```

W tej komendzie *QMGRNAME* jest nazwą menedżera kolejek.

Windows Linux W systemach Linux i Windowsmożna również utworzyć definicję kanału połączenia z serwerem za pomocą programu IBM MQ Explorer.

z/OS W systemie z/OS do utworzenia definicji kanału połączenia z serwerem można użyć panelu operacji i panelu sterowania.

Nazwa kanału (JAVA.CHANNEL w poprzednich przykładach) musi być taki sam, jak nazwa kanału określona we właściwości CHANNEL fabryki połączeń używanej przez aplikację do nawiązywania połączenia z menedżerem kolejek. Wartością domyślną właściwości CHANNEL jest SYSTEM.DEF.SVRCONN.

## Uruchamianie programu nasłuchującego

Jeśli program nasłuchujący dla menedżera kolejek nie został jeszcze uruchomiony, należy go uruchomić.

Multi W systemie Wiele platformmożna użyć komendy MQSC START LISTENER, aby uruchomić proces nasłuchujący po utworzeniu obiektu nasłuchiwania za pomocą komendy MQSC DEFINE LISTENER, jak pokazano w poniższym przykładzie:

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)
START LISTENER(LISTENER.TCP)
```

z/OS W systemie z/OSnależy użyć tylko komendy START LISTENER, jak w poniższym przykładzie, ale należy pamiętać, że przestrzeń adresowa inicjatora kanału musi zostać uruchomiona przed uruchomieniem programu nasłuchującego:

```
START LISTENER TRPTYPE(TCP) PORT(1414)
```

IBM i W systemie IBM ido uruchomienia programu nasłuchującego można również użyć komendy CL STRMQMLSR, jak w poniższym przykładzie:

```
STRMQMLSR PORT(1414) MQMNAME(QMGRNAME)
```

W tej komendzie *QMGRNAME* jest nazwą menedżera kolejek.

ALW W systemie AIX, Linux, and Windowsdo uruchomienia programu nasłuchującego można również użyć komendy sterującej **runmqlsr** , tak jak w poniższym przykładzie:

```
runmqlsr -t tcp -p 1414 -m QMgrName
```

W tej komendzie *QMgrName* jest nazwą menedżera kolejek.

Windows Linux W systemach Linux i Windowsprogram nasłuchujący można również uruchomić za pomocą komendy IBM MQ Explorer.

z/OS W systemie z/OSdo uruchomienia programu nasłuchującego można również użyć operacji i paneli sterowania.

Numer portu, na którym nasłuchuje program nasłuchujący, musi być taki sam, jak numer portu określony we właściwości PORT fabryki połączeń używanej przez aplikację do nawiązania połączenia z menedżerem kolejek. Wartością domyślną właściwości PORT jest 1414.

### *Narzędzie do weryfikowania instalacji (IVT) typu punkt z punktem dla systemu IBM MQ classes for JMS*

Program IVT (point-to-point installation verification test) jest dostarczany z programami IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging. Program nawiązuje połączenie z menedżerem kolejek w trybie powiązań lub w trybie klienta i wysyła komunikat do kolejki o nazwie SYSTEM.DEFAULT.LOCAL.QUEUE, a następnie odbiera komunikat z kolejki. Program może tworzyć i konfigurować wszystkie obiekty, które są wymagane dynamicznie w czasie wykonywania, lub może używać interfejsu JNDI do pobierania obiektów administrowanych z usługi katalogowej.

Uruchom test weryfikujący instalację bez używania interfejsu JNDI jako pierwszego, ponieważ test jest samodzielny i nie wymaga użycia usługi katalogowej. Opis obiektów administrowanych zawiera sekcja Konfigurowanie obiektów JMS za pomocą narzędzia administracyjnego.

### Test weryfikujący instalację punkt-punkt bez użycia interfejsu JNDI

W tym teście program IVT tworzy i konfiguruje wszystkie obiekty, które są wymagane dynamicznie w czasie wykonywania i nie używa interfejsu JNDI.

Multi Na wielu platformach dostępny jest skrypt uruchamiający program IVT. Skrypt ma nazwę **IVTRun** w systemach AIX and Linux i **IVTRun.bat** w systemie Windows. skrypt znajduje się w podkatalogu `bin` katalogu instalacyjnego IBM MQ classes for JMS . Ścieżka klasy musi zawierać łańcuch `com.ibm.mqjms.jar`.

Aby uruchomić test w trybie powiązań, wprowadź następującą komendę:

```
IVTRun -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

Aby uruchomić test w trybie klienta, należy najpierw skonfigurować menedżer kolejek zgodnie z opisem w sekcji “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym” na stronie 1101. Należy zauważyć, że domyślnym kanałem, który ma być używany, jest **SYSTEM.DEF.SVRCONN** , a kolejką, która ma być używana, jest **SYSTEM.DEFAULT.LOCAL.QUEUE**, a następnie należy wprowadzić następującą komendę:

```
IVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]
[-v providerVersion ] [-ccsid ccsid ] [-t]
```

z/OS W systemach z/OS nie jest dostępny żaden równoważny skrypt. Zamiast tego należy uruchomić narzędzie IVT w trybie powiązań, wywołując bezpośrednio klasę Java . W systemie z/OSmożna wybrać jedną z dwóch identycznych funkcjonalnie instancji programu IVT:

- Plik `com.ibm.mq.jms.MQJMSIVT`, który jest dostępny z produktem IBM MQ classes for JMS (JMS 2.0). Aby można było używać tego programu, ścieżka klasy musi zawierać łańcuch `com.ibm.mqjms.jar` lub `com.ibm.mq.allclient.jar`.
- Plik `com.ibm.mq.jakarta.jms.MQJMSIVT`, który jest dostępny z produktem IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0). Aby można było użyć tego programu, ścieżka klasy musi zawierać łańcuch `com.ibm.mq.jakarta.client.jar`.

Aby uruchomić test w trybie powiązań w systemie z/OS, wprowadź następującą komendę:

```
java com.ibm.mq.jms.MQJMSIVT -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

Parametry komend mają następujące znaczenie:

**-m *menedżer_kolejek***
Nazwa menedżera kolejek, z którym łączy się program IVT. Jeśli test zostanie uruchomiony w trybie powiązań i parametr ten zostanie pominięty, program IVT nawiąże połączenie z domyślnym menedżerem kolejek.

**-host *nazwa_hosta***
Nazwa hosta lub adres IP systemu, w którym działa menedżer kolejek.

**-port *port***
Numer portu, na którym nasłuchuje proces nasłuchujący menedżera kolejek. Wartością domyślną jest 1414.

**-channel *kanał***
Nazwa kanału MQI używanego przez program IVT do nawiązywania połączenia z menedżerem kolejek. Wartością domyślną jest `SYSTEM.DEF.SVRCONN`.

**-v *providerVersion***
Poziom wersji menedżera kolejek, z którym program IVT ma się połączyć.

Ten parametr służy do ustawiania właściwości PROVIDERVERSION obiektu fabryki MQQueueConnectioni ma takie same poprawne wartości jak właściwość PROVIDERVERSION. Więcej informacji na temat tego parametru, w tym jego poprawne wartości, zawiera sekcja JMS: changes to PROVIDERVERSION property (JMS: zmiany właściwości PROVIDERVERSION) oraz opis właściwości PROVIDERVERSION w sekcji Properties of IBM MQ classes for JMS objects(Właściwości obiektów).

Wartością domyślną jest `unspecified`.

**-ccsid *id_ccsid***
Identyfikator (CCSID) kodowanego zestawu znaków lub strony kodowej, który ma być używany przez połączenie. Wartością domyślną jest 819.

**-t**
Śledzenie jest włączone. Domyślnie śledzenie jest wyłączone.

Pomyślny test generuje dane wyjściowe podobne do poniższych przykładowych danych wyjściowych:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
WebSphere MQ classes for Java(tm) Message Service 7.0
Installation Verification Test

Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620005e03
JMSTimestamp:   1187170264000
JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:    null
JMSRedelivered:  false
```

```
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 28
JMSXAppID: 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
```

## Test weryfikujący instalację punkt-punkt przy użyciu interfejsu JNDI


W tym teście program IVT używa interfejsu JNDI do pobierania obiektów administrowanych z usługi katalogowej.

Przed uruchomieniem testu należy skonfigurować usługę katalogową opartą na serwerze LDAP (Lightweight Directory Access Protocol) lub lokalnym systemie plików. Należy również skonfigurować narzędzie administracyjne IBM MQ JMS , tak aby mogło ono używać usługi katalogowej do przechowywania obiektów administrowanych. Więcej informacji na temat tych wymagań wstępnych zawiera sekcja “Wymagania wstępne dla produktu IBM MQ classes for JMS” na stronie 91. Informacje na temat konfigurowania narzędzia administracyjnego produktu IBM MQ JMS zawiera sekcja Konfigurowanie narzędzia administracyjnego produktu JMS.

Program IVT musi mieć możliwość użycia interfejsu JNDI w celu pobrania obiektu fabryki MQQueueConnectioni obiektu MQQueue z usługi katalogowej. W celu utworzenia tych obiektów administrowanych udostępniono skrypt. Skrypt ma nazwę IVTSetup w systemach AIX and Linux i IVTSetup.bat w systemie Windowsi znajduje się w podkatalogu `bin` katalogu instalacyjnego IBM MQ classes for JMS . Aby uruchomić skrypt, wprowadź następującą komendę:

```
IVTSetup
```

Skrypt wywołuje narzędzie administracyjne IBM MQ JMS w celu utworzenia obiektów administrowanych.

Obiekt fabryki MQQueueConnectionjest powiązany z nazwą ivtQCF i jest tworzony z wartościami domyślnymi dla wszystkich jego właściwości, co oznacza, że program IVT działa w trybie powiązań i nawiązuje połączenie z domyślnym menedżerem kolejek. Aby uruchomić program IVT w trybie klienta lub nawiązać połączenie z menedżerem kolejek innym niż domyślny, należy użyć narzędzia administracyjnego produktu IBM MQ JMS lub programu IBM MQ Explorer w celu zmiany odpowiednich właściwości obiektu fabryki MQQueueConnection. Informacje na temat korzystania z narzędzia administracyjnego IBM MQ Explorer JMS zawiera sekcja Konfigurowanie obiektów JMS za pomocą narzędzia administracyjnego. Informacje na temat korzystania z produktu IBM MQ Explorerzawiera sekcja Wprowadzenie do produktu IBM MQ Explorer lub pomoc dostarczana z produktem IBM MQ Explorer.

Obiekt MQQueue jest powiązany z nazwą ivtQ i jest tworzony z wartościami domyślnymi dla wszystkich jego właściwości, z wyjątkiem właściwości QUEUE, która ma wartość SYSTEM.DEFAULT.LOCAL.QUEUE.

Po utworzeniu obiektów administrowanych można uruchomić program IVT. Aby uruchomić test przy użyciu interfejsu JNDI, wprowadź następującą komendę:

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

Parametry komendy mają następujące znaczenie:

**-url "*providerURL*"**
Adres URL (URL) usługi katalogowej. URL może mieć jeden z następujących formatów:

- `ldap://`*`hostname`*`/`*`contextName`* dla usługi katalogowej opartej na serwerze LDAP
- `file:/`*`directoryPath`* dla usługi katalogowej opartej na lokalnym systemie plików

Należy pamiętać, że URL należy ująć w cudzysłów (").

**-icf *initCtxFakt***
Nazwa klasy fabryki kontekstu początkowego, która musi mieć jedną z następujących wartości:

- `com.sun.jndi.ldap.LdapCtxFactory`dla usługi katalogowej opartej na serwerze LDAP. Jest to wartość domyślna.
- `com.sun.jndi.fscontext.RefFSContextFactory`dla usługi katalogowej opartej na lokalnym systemie plików.

**-t**
Śledzenie jest włączone. Domyślnie śledzenie jest wyłączone.

Test zakończony powodzeniem generuje dane wyjściowe podobne do danych wyjściowych dla pomyślnego testu bez używania interfejsu JNDI. Główna różnica polega na tym, że dane wyjściowe wskazują, że test używa interfejsu JNDI do pobierania obiektu fabryki MQQueueConnectioni obiektu MQQueue.

Chociaż nie jest to ściśle konieczne, dobrą praktyką jest porządkowanie po teście przez usunięcie obiektów administrowanych utworzonych przez skrypt IVTSetup. W tym celu udostępniono skrypt. Skrypt ma nazwę IVTTidy w systemach AIX and Linux i IVTTidy.bat w systemie Windowsi znajduje się w podkatalogu `bin` katalogu instalacyjnego IBM MQ classes for JMS .

## Określanie problemu dla testu weryfikującego instalację punkt-punkt


Test weryfikujący instalację może zakończyć się niepowodzeniem z następujących powodów:

- Jeśli program IVT zapisze komunikat informujący, że nie może znaleźć klasy, sprawdź, czy ścieżka klasy jest ustawiona poprawnie, zgodnie z opisem w sekcji “Ustawianie zmiennych środowiskowych dla IBM MQ classes for JMS/Jakarta Messaging” na stronie 97.
- Test może zakończyć się niepowodzeniem z następującym komunikatem:

```
Failed to connect to queue manager ' qmgr ' with connection mode ' connMode '
and host name ' hostname '
```

  i powiązany kod przyczyny 2059. Zmienne w komunikacie mają następujące znaczenie:

  ***QMGR***
  Nazwa menedżera kolejek, z którym program IVT próbuje się połączyć. Wstawienie tego komunikatu jest puste, jeśli program IVT próbuje nawiązać połączenie z domyślnym menedżerem kolejek w trybie powiązań.

  ***connMode***
  Tryb połączenia: `Bindings` lub `Client`.

  ***nazwa_hosta***
  Nazwa hosta lub adres IP systemu, w którym działa menedżer kolejek.

  Ten komunikat oznacza, że menedżer kolejek, z którym próbuje nawiązać połączenie program IVT, nie jest dostępny. Sprawdź, czy menedżer kolejek jest uruchomiony i jeśli program IVT próbuje nawiązać połączenie z domyślnym menedżerem kolejek, upewnij się, że menedżer kolejek jest zdefiniowany jako domyślny menedżer kolejek dla systemu.

- Test może zakończyć się niepowodzeniem z następującym komunikatem:

```
Failed to open MQ queue 'SYSTEM.DEFAULT.LOCAL.QUEUE'
```

Ten komunikat oznacza, że kolejka SYSTEM.DEFAULT.LOCAL.QUEUE nie istnieje w menedżerze kolejek, z którym jest połączony program IVT. Alternatywnie, jeśli kolejka istnieje, program IVT nie może otworzyć kolejki, ponieważ nie ma możliwości umieszczania i pobierania komunikatów. Sprawdź, czy kolejka istnieje i czy można w niej wstawiać i pobierać komunikaty.

- Test może zakończyć się niepowodzeniem z następującym komunikatem:

```
Unable to bind to object
```

Ten komunikat oznacza, że istnieje połączenie z serwerem LDAP, ale serwer LDAP nie jest poprawnie skonfigurowany. Albo serwer LDAP nie jest skonfigurowany do przechowywania obiektów Java , albo uprawnienia do obiektów lub przyrostka nie są poprawne. Więcej informacji na ten temat zawiera dokumentacja serwera LDAP.

- Test może zakończyć się niepowodzeniem z następującym komunikatem:

```
The security authentication was not valid that was supplied for
QueueManager ' qmgr ' with connection mode 'Client' and host name ' hostname '
```

Ten komunikat oznacza, że menedżer kolejek nie jest poprawnie skonfigurowany do akceptowania połączenia klienta z systemu. Szczegółowe informacje można znaleźć w sekcji “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym” na stronie 1101.

### *Narzędzie do weryfikowania instalacji (IVT) publikowania/subskrypcji dla produktu IBM MQ classes for JMS*

Wraz z produktem IBM MQ classes for JMSdostarczany jest program testu weryfikującego instalację (IVT) publikowania/subskrypcji. Program nawiązuje połączenie z menedżerem kolejek w trybie powiązań lub w trybie klienta, subskrybuje temat, publikuje komunikat w temacie, a następnie odbiera komunikat, który właśnie opublikował. Program może tworzyć i konfigurować wszystkie obiekty, które są wymagane dynamicznie w czasie wykonywania, lub może używać interfejsu JNDI do pobierania obiektów administrowanych z usługi katalogowej.

Uruchom test weryfikujący instalację bez używania interfejsu JNDI jako pierwszego, ponieważ test jest samodzielny i nie wymaga użycia usługi katalogowej. Opis obiektów administrowanych zawiera sekcja Konfigurowanie obiektów JMS za pomocą narzędzia administracyjnego.

### Test weryfikujący instalację publikowania/subskrypcji bez użycia interfejsu JNDI

W tym teście program IVT tworzy i konfiguruje wszystkie obiekty, które są wymagane dynamicznie w czasie wykonywania i nie używa interfejsu JNDI.

Dostępny jest skrypt uruchamiający program IVT. Skrypt ma nazwę PSIVTRun w systemach AIX and Linux i PSIVTRun.bat w systemie Windowsi znajduje się w podkatalogu `bin` katalogu instalacyjnego IBM MQ classes for JMS .

Aby uruchomić test w trybie powiązań, wprowadź następującą komendę:

```
PSIVTRun -nojndi [-m qmgr ] [-bqm brokerQmgr ] [-v providerVersion ] [-t]
```

Aby uruchomić test w trybie klienta, należy najpierw skonfigurować menedżer kolejek zgodnie z opisem w sekcji “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym” na stronie 1101 , która wskazuje, że kanał, który ma być używany, ma ustawioną wartość domyślną SYSTEM.DEF.SVRCONN, a następnie wprowadź następującą komendę:

```
PSIVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]
[-bqm brokerQmgr ] [-v providerVersion ] [-ccsid ccsid ] [-t]
```

Parametry komend mają następujące znaczenie:

**-m** ***menedżer_kolejek***
Nazwa menedżera kolejek, z którym łączy się program IVT. Jeśli test zostanie uruchomiony w trybie powiązań i parametr ten zostanie pominięty, program IVT nawiąże połączenie z domyślnym menedżerem kolejek.

**-host** ***nazwa_hosta***
Nazwa hosta lub adres IP systemu, w którym działa menedżer kolejek.

**-port** ***port***
Numer portu, na którym nasłuchuje proces nasłuchujący menedżera kolejek. Wartością domyślną jest 1414.

**-channel** ***kanał***
Nazwa kanału MQI używanego przez program IVT do nawiązywania połączenia z menedżerem kolejek. Wartością domyślną jest `SYSTEM.DEF.SVRCONN`.

**-bqm** ***brokerQmgr***
Nazwa menedżera kolejek, w którym działa broker. Wartością domyślną jest nazwa menedżera kolejek, z którym łączy się program IVT.

Ten parametr nie ma zastosowania w przypadku menedżera kolejek w wersji v 7 lub nowszej.

**-v** ***providerVersion***
Poziom wersji menedżera kolejek, z którym program IVT ma się połączyć.

Ten parametr służy do ustawiania właściwości PROVIDERVERSION obiektu fabryki MQTopicConnectioni ma takie same poprawne wartości jak właściwość PROVIDERVERSION. Więcej informacji na temat tego parametru, w tym jego poprawne wartości, zawiera opis właściwości PROVIDERVERSION w sekcji Właściwości obiektów IBM MQ classes for JMS.

Wartością domyślną jest `unspecified`.

**-ccsid** ***id_ccsid***
Identyfikator (CCSID) kodowanego zestawu znaków lub strony kodowej, który ma być używany przez połączenie. Wartością domyślną jest 819.

**-t**
Śledzenie jest włączone. Domyślnie śledzenie jest wyłączone.

Pomyślny test generuje dane wyjściowe podobne do poniższych przykładowych danych wyjściowych:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. 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:414d5120514d5f6d627720202020202001edb14620006706
JMSTimestamp:   1187182520203
JMSCorrelationID: ID:414d5120514d5f6d627720202020202001edb14620006704
JMSDestination:  topic://MQJMS/PSIVT/Information
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
```

```
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 26
JMSXAppID: QM_mbw
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_ConnectionID: 414D5143514D5F6D627720202020202001EDB14620006601
JMS_IBM_PutTime: 12552020
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSPSIVT program
Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT finished
```

## Test weryfikujący instalację publikowania/subskrypcji przy użyciu interfejsu JNDI

W tym teście program IVT używa interfejsu JNDI do pobierania obiektów administrowanych z usługi katalogowej.

Przed uruchomieniem testu należy skonfigurować usługę katalogową opartą na serwerze LDAP (Lightweight Directory Access Protocol) lub lokalnym systemie plików. Należy również skonfigurować narzędzie administracyjne IBM MQ JMS , tak aby mogło ono używać usługi katalogowej do przechowywania obiektów administrowanych. Więcej informacji na temat tych wymagań wstępnych zawiera sekcja “Wymagania wstępne dla produktu IBM MQ classes for JMS” na stronie 91. Informacje na temat konfigurowania narzędzia administracyjnego produktu IBM MQ JMS zawiera sekcja Konfigurowanie narzędzia administracyjnego produktu JMS.

Program IVT musi mieć możliwość użycia interfejsu JNDI w celu pobrania obiektu fabryki MQTopicConnectioni obiektu MQTopic z usługi katalogowej. W celu utworzenia tych obiektów administrowanych udostępniono skrypt. Skrypt ma nazwę IVTSetup w systemach AIX and Linux i IVTSetup.bat w systemie Windowsi znajduje się w podkatalogu `bin` katalogu instalacyjnego IBM MQ classes for JMS . Aby uruchomić skrypt, wprowadź następującą komendę:

```
IVTSetup
```

Skrypt wywołuje narzędzie administracyjne IBM MQ JMS w celu utworzenia obiektów administrowanych.

Obiekt fabryki MQTopicConnectionjest powiązany z nazwą ivtTCF i jest tworzony z wartościami domyślnymi dla wszystkich jego właściwości, co oznacza, że program IVT działa w trybie powiązań, łączy się z domyślnym menedżerem kolejek i używa wbudowanej funkcji publikowania/subskrybowania. Jeśli program IVT ma działać w trybie klienta, nawiąż połączenie z menedżerem kolejek innym niż domyślny menedżer kolejek lub użyj funkcji IBM Integration Bus zamiast wbudowanej funkcji publikowania/subskrypcji, aby zmienić odpowiednie właściwości obiektu fabryki MQTopicConnection, należy użyć narzędzia administracyjnego produktu IBM MQ JMS lub Eksploratora IBM MQ . Informacje na temat korzystania z narzędzia administracyjnego IBM MQ JMS zawiera sekcja Konfigurowanie obiektów JMS za pomocą narzędzia administracyjnego. Więcej informacji na temat korzystania z programu Eksplorator IBM MQ zawiera pomoc dostarczana z programem IBM MQ Explorer.

Obiekt MQTopic jest powiązany z nazwą ivtT i jest tworzony z wartościami domyślnymi dla wszystkich jego właściwości, z wyjątkiem właściwości TOPIC, która ma wartość MQJMS/PSIVT/Information.

Po utworzeniu obiektów administrowanych można uruchomić program IVT. Aby uruchomić test przy użyciu interfejsu JNDI, wprowadź następującą komendę:

```
PSIVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

Parametry komendy mają następujące znaczenie:

**-url "*providerURL*"**
Adres URL (URL) usługi katalogowej. URL może mieć jeden z następujących formatów:

- `ldap://`*`hostname`*`/`*`contextName`* dla usługi katalogowej opartej na serwerze LDAP
- `file:/`*`directoryPath`* dla usługi katalogowej opartej na lokalnym systemie plików

Należy pamiętać, że URL należy ująć w cudzysłów (").

**-icf** ***initCtxFakt***
Nazwa klasy fabryki kontekstu początkowego, która musi mieć jedną z następujących wartości:

- `com.sun.jndi.ldap.LdapCtxFactory`dla usługi katalogowej opartej na serwerze LDAP. Jest to wartość domyślna.
- `com.sun.jndi.fscontext.RefFSContextFactory`dla usługi katalogowej opartej na lokalnym systemie plików.

**-t**
Śledzenie jest włączone. Domyślnie śledzenie jest wyłączone.

Test zakończony powodzeniem generuje dane wyjściowe podobne do danych wyjściowych dla pomyślnego testu bez używania interfejsu JNDI. Główna różnica polega na tym, że dane wyjściowe wskazują, że test używa interfejsu JNDI do pobrania obiektu fabryki MQTopicConnectioni obiektu MQTopic.

Chociaż nie jest to ściśle konieczne, dobrą praktyką jest porządkowanie po teście przez usunięcie obiektów administrowanych utworzonych przez skrypt IVTSetup. W tym celu udostępniono skrypt. Skrypt ma nazwę IVTTidy w systemach AIX and Linux i IVTTidy.bat w systemie Windowsi znajduje się w podkatalogu `bin` katalogu instalacyjnego IBM MQ classes for JMS .

## Określanie problemu dla testu weryfikującego instalację publikowania/subskrypcji

Test weryfikujący instalację może zakończyć się niepowodzeniem z następujących powodów:

- Jeśli program IVT zapisze komunikat informujący, że nie może znaleźć klasy, sprawdź, czy ścieżka klasy jest ustawiona poprawnie, zgodnie z opisem w sekcji “Ustawianie zmiennych środowiskowych dla IBM MQ classes for JMS/Jakarta Messaging” na stronie 97.
- Test może zakończyć się niepowodzeniem z następującym komunikatem:

```
Failed to connect to queue manager ' qmgr ' with
connection mode ' connMode ' and host name ' hostname '
```

i powiązany kod przyczyny 2059. Zmienne w komunikacie mają następujące znaczenie:

***QMGR***
Nazwa menedżera kolejek, z którym program IVT próbuje się połączyć. Wstawienie tego komunikatu jest puste, jeśli program IVT próbuje nawiązać połączenie z domyślnym menedżerem kolejek w trybie powiązań.

***connMode***
Tryb połączenia: `Bindings` lub `Client`.

***nazwa_hosta***
Nazwa hosta lub adres IP systemu, w którym działa menedżer kolejek.

Ten komunikat oznacza, że menedżer kolejek, z którym próbuje nawiązać połączenie program IVT, nie jest dostępny. Sprawdź, czy menedżer kolejek jest uruchomiony i jeśli program IVT próbuje nawiązać połączenie z domyślnym menedżerem kolejek, upewnij się, że menedżer kolejek jest zdefiniowany jako domyślny menedżer kolejek dla systemu.

- Test może zakończyć się niepowodzeniem z następującym komunikatem:

```
Unable to bind to object
```

Ten komunikat oznacza, że istnieje połączenie z serwerem LDAP, ale serwer LDAP nie jest poprawnie skonfigurowany. Albo serwer LDAP nie jest skonfigurowany do przechowywania obiektów Java , albo

uprawnienia do obiektów lub przyrostka nie są poprawne. Więcej informacji na ten temat zawiera dokumentacja serwera LDAP.

- Test może zakończyć się niepowodzeniem z następującym komunikatem:

```
The security authentication was not valid that was supplied for
QueueManager 'qmgr' with connection mode 'Client' and host name 'hostname'
```

Ten komunikat oznacza, że menedżer kolejek nie jest poprawnie skonfigurowany do akceptowania połączenia klienta z systemu. Więcej informacji na ten temat zawiera “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym” na stronie 1101.

## JMS 2.0 *Korzystanie z przykładowych aplikacji IBM MQ classes for JMS*

Przykładowe aplikacje IBM MQ classes for JMS udostępniają przegląd wspólnych funkcji interfejsu API języka JMS . Można ich używać do weryfikowania konfiguracji serwera instalacji i przesyłania komunikatów oraz do tworzenia własnych aplikacji.

### O tym zadaniu

Jeśli potrzebna jest pomoc przy tworzeniu własnych aplikacji, można użyć przykładowych aplikacji jako punktu początkowego. Dla każdej aplikacji udostępniono zarówno wersję źródłową, jak i skompilowaną. Zapoznaj się z przykładowym kodem źródłowym i zidentyfikuj kluczowe kroki, które należy wykonać, aby utworzyć każdy wymagany obiekt dla aplikacji (ConnectionFactory, Connection, Session, Destination i Producer, lub Consumer), a także aby ustawić konkretne właściwości, które są wymagane do określenia sposobu działania aplikacji. Więcej informacji na ten temat zawiera “Pisanie aplikacji IBM MQ classes for JMS/Jakarta Messaging” na stronie 145. Przykłady mogą ulec zmianie w przyszłych wersjach produktu IBM MQ.

W przypadku systemu JMS 2.0 Tabela 11 na stronie 124 przedstawia miejsce, w którym na każdej platformie są zainstalowane aplikacje przykładowe IBM MQ classes for JMS .

**Uwaga:**

JM 3.0 W systemie IBM MQ classes for Jakarta Messagingprzygotowywane są nowe przykłady.

*Tabela 11. Katalogi instalacyjne przykładowych aplikacji IBM MQ classes for JMS*

| Platforma | Katalog |
|---|---|
| AIX AIX<br>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 |

W tym katalogu znajdują się podkatalogi zawierające co najmniej jedną przykładową aplikację (patrz Tabela 12 na stronie 124).

*Tabela 12. IBM MQ classes for JMS aplikacje przykładowe*

| Nazwa próbki | Opis |
|---|---|
| JmsBrowser .java | Przeglądarka kolejek systemu JMS , która sprawdza wszystkie dostępne komunikaty w nazwanej kolejce, bez ich usuwania, w kolejności, w jakiej byłyby odbierane przez aplikację konsumującą. |

| *Tabela 12. IBM MQ classes for JMS aplikacje przykładowe (kontynuacja)* | |
|---|---|
| **Nazwa próbki** | **Opis** |
| JmsConsumer.java | Aplikacja przeglądarki kolejek systemu JMS , która sprawdza wszystkie dostępne komunikaty w nazwanej kolejce bez usuwania ich w kolejności, w jakiej zostałyby odebrane przez aplikację konsumującą, poprzez wyszukiwanie instancji fabryki połączeń i instancji miejsca docelowego w kontekście początkowym (ten przykład obsługuje tylko kontekst systemu plików). |
| JmsJndiConsumer.java | Aplikacja konsumenta (odbiornika lub subskrybenta) JMS , która odbiera komunikat z nazwanego miejsca docelowego (kolejki lub tematu) przez wyszukiwanie instancji fabryki połączeń i instancji miejsca docelowego w kontekście początkowym (ten przykład obsługuje tylko kontekst systemu plików). |
| JmsJndiProducer.java | Aplikacja producenta JMS (nadawca lub publikator), która wysyła prosty komunikat do nazwanego miejsca docelowego (kolejki lub tematu), wyszukując instancję fabryki połączeń i instancję miejsca docelowego w kontekście początkowym (ten przykład obsługuje tylko kontekst systemu plików). |
| JmsProducer.java | Aplikacja producenta JMS (wysyłająca lub publikująca), która wysyła prosty komunikat do nazwanego miejsca docelowego (kolejki lub tematu). |
| **/interaktywny/** | |
| SampleConsumerJava.java | Odbierz komunikaty z tematu/kolejki. |
| SampleProducerJava.java | Wyślij komunikaty do tematu/kolejki. |
| **/interactive/helper/** | |
| BaseOptions.java | Klasa abstrakcyjna, którą można rozszerzyć w celu udostępnienia funkcji opcji użytkownika. |
| IsValidType.java | Klasa abstrakcyjna dla klas sprawdzania poprawności. |
| JmsApp.java | Klasa abstrakcyjna, którą można rozszerzyć w celu udostępnienia funkcjonalności konsumenta/producenta. |
| Keys.java | Zestaw kluczy definiujących opcje dla przykładowych aplikacji. |
| Literals.java | Zestaw stałych literałów. |
| MyContext.java | Kontekst, w którym prezentowane są opcje. |
| Options.java | Udostępnia funkcje dla opcji użytkownika. |
| OptionsPresenter.java | Kontekst, w którym prezentowane są bieżące opcje. |
| **/simple/** | |
| SimpleAsyncPutPTP.java | Prosta aplikacja do przesyłania komunikatów w trybie punkt z punktem. Komunikat jest wysyłany asynchronicznie (zwana również przesyłaniem komunikatów *fire-and-forget* ). Nie są odbierane żadne komunikaty. |
| SimpleDurableSub.java | Prosta aplikacja demonstrująca trwałe narzędzie subskrypcji. |

| *Tabela 12. IBM MQ classes for JMS aplikacje przykładowe (kontynuacja)* | |
|---|---|
| **Nazwa próbki** | **Opis** |
| SimpleJNDILookup.java | Minimalna i prosta aplikacja demonstrująca wyszukiwanie obiektów JMS przy użyciu kontekstu początkowego. Nie jest nawiązywane żadne połączenie z menedżerem kolejek ani nie są wysyłane ani odbierane żadne komunikaty. |
| SimpleMQMDRead.java, | Prosta aplikacja demonstrująca sposób, w jaki aplikacja JMS może korzystać z pól deskryptora komunikatu (MQMD) produktu MQ jako właściwości komunikatu JMS . Nie są wysyłane żadne komunikaty; zakłada się, że używana kolejka jest zapełniona niektórymi komunikatami. |
| SimpleMQMDWrite.java | Prosta aplikacja demonstrująca, w jaki sposób aplikacja JMS może zapisywać pola deskryptora komunikatu (MQMD) produktu MQ . Nie są odbierane żadne komunikaty. |
| SimplePTP.java, | Minimalna i prosta aplikacja do przesyłania komunikatów w trybie punkt z punktem. |
| SimplePubSub.java | Minimalna i prosta aplikacja do przesyłania komunikatów w trybie publikowania i subskrypcji. |
| SimpleReadAheadPTP.java | Prosta aplikacja do przesyłania komunikatów w trybie punkt z punktem. Komunikaty są przesyłane strumieniowo z menedżera kolejek (zwanego również narzędziem do odczytu z wyprzedzeniem). Nie są wysyłane żadne komunikaty; zakłada się, że używana kolejka jest zapełniona niektórymi komunikatami. |
| SimpleRequestor.java | Prosta aplikacja używająca requestera do wysyłania komunikatu żądania, a następnie oczekiwania na odpowiedź i odbierania odpowiedzi. Uwaga: zakłada się, że inna aplikacja przetworzy komunikat żądania i wyśle komunikat odpowiedzi. |
| SimpleResponder.java | Prosta aplikacja, która nasłuchuje w miejscu docelowym komunikatu, a następnie wysyła odpowiedź do miejsca docelowego replyTo komunikatu. Aplikacja została napisana w celu działania w połączeniu z przykładem SimpleRequestor . |
| SimpleRetainedPub.java | Prosta aplikacja demonstrująca zachowaną publikację. Nie są odbierane żadne komunikaty. |
| SimpleWMQJMSPTP.java, | Minimalna i prosta aplikacja do przesyłania komunikatów w trybie punkt z punktem. |
| SimpleWMQJMSPubSub.java | Minimalna i prosta aplikacja do przesyłania komunikatów w trybie publikowania/subskrypcji. |

IBM MQ classes for JMS udostępnia skrypt o nazwie `runjms` , który może być używany do uruchamiania przykładowych aplikacji. Ten skrypt konfiguruje środowisko IBM MQ , aby umożliwić uruchamianie przykładowych aplikacji IBM MQ classes for JMS .

Tabela 13 na stronie 126 przedstawia położenie skryptu na każdej platformie:

| *Tabela 13. Położenie skryptu `runjms`* | |
|---|---|
| **Platforma** | **Katalog** |
|  AIX<br>  Linux | *MQ_INSTALLATION_PATH*`/java/bin/runjms` |
|  Windows | *MQ_INSTALLATION_PATH*`\java\bin\runjms.bat` |

| *Tabela 13. Położenie skryptu `runjms` (kontynuacja)* | |
|---|---|
| **Platforma** | **Katalog** |
| IBM i | `/qibm/proddata/mqm/java/bin/runjms`<br>lub wersji<br>`/qibm/proddata/mqm/java/bin/runjms64` |
| z/OS | *MQ_INSTALLATION_PATH*`/java/bin/runjms` |

Aby użyć skryptu `runjms` do wywołania przykładowej aplikacji, wykonaj następujące kroki:

## Procedura

1. Otwórz wiersz komend i przejdź do katalogu zawierającego przykładową aplikację, która ma zostać uruchomiona.
2. Wprowadź następującą komendę:

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   Przykładowa aplikacja wyświetla listę parametrów, których potrzebuje.
3. Wprowadź następującą komendę, aby uruchomić przykład z następującymi parametrami:

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

### Przykład

Linux Na przykład, aby uruchomić przykład `JmsBrowser` w systemie Linux, wprowadź następujące komendy:

```
cd /opt/mqm/samp/jms/samples
/opt/mqm/java/bin/runjms JmsBrowser -m QM1 -d LQ1
```

### Pojęcia pokrewne

“Co jest instalowane w systemie IBM MQ classes for JMS” na stronie 92
Podczas instalowania produktu IBM MQ classes for JMStworzona jest pewna liczba plików i katalogów. W systemie Windowsniektóre czynności konfiguracyjne są wykonywane podczas instalacji przez automatyczne ustawianie zmiennych środowiskowych. Na innych platformach i w niektórych środowiskach Windows należy ustawić zmienne środowiskowe przed uruchomieniem aplikacji IBM MQ classes for JMS .

## *Skrypty udostępnione z produktem IBM MQ classes for JMS/Jakarta Messaging*

Udostępniono szereg skryptów ułatwiających wykonywanie typowych zadań, które muszą być wykonywane podczas korzystania z produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging.

Tabela 14 na stronie 127 zawiera listę wszystkich skryptów i ich użycia. Skrypty znajdują się w podkatalogu `bin` katalogu instalacyjnego produktu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging .

| *Tabela 14. Skrypty udostępniane z produktem IBM MQ classes for JMS i produktem IBM MQ classes for Jakarta Messaging* | |
|---|---|
| **Użyteczność** | **Użycie** |
| Procedura czyszcząca [1] | Ten skrypt jest obsługiwany w celu zachowania zgodności z poprzednimi wersjami, ale nie wykonuje żadnej funkcji. Ręczne czyszczenie informacji o subskrypcji nie jest już konieczne. |

*Tabela 14. Skrypty udostępniane z produktem IBM MQ classes for JMS i produktem IBM MQ classes for Jakarta Messaging (kontynuacja)*

| **Użyteczność** | **Użycie** |
|---|---|
| DefaultConfiguration | Uruchamia domyślną aplikację konfiguracyjną na platformach innych niż Windows. |
| formatLog [1] | Ten skrypt jest obsługiwany w celu zachowania zgodności z poprzednimi wersjami, ale nie wykonuje żadnej funkcji. Dane wyjściowe dziennika są teraz generowane w postaci czytelnego tekstu. |
| IVTRun [1]<br>IVTSetup [1]<br>IVTTidy [1] | Używany w teście weryfikacji instalacji punkt z punktem, zgodnie z opisem w sekcji "Narzędzie do weryfikowania instalacji (IVT) typu punkt z punktem dla systemu IBM MQ classes for JMS" na stronie 116. |
| JM 3.0 JMS30Admin [1] | Uruchamia narzędzie administracyjne IBM MQ Jakarta Messaging zgodnie z opisem w sekcji Uruchamianie narzędzia administracyjnego. |
| JM 3.0 JMS30Admin.config | Plik konfiguracyjny narzędzia administracyjnego IBM MQ Jakarta Messaging , zgodnie z opisem w sekcji Konfigurowanie narzędzia administracyjnego JMS. |
| JMS 2.0 Administrator JMS [1] | Uruchamia narzędzie administracyjne IBM MQ JMS zgodnie z opisem w sekcji Uruchamianie narzędzia administracyjnego. |
| JMS 2.0 JMSAdmin.config | Plik konfiguracyjny narzędzia administracyjnego IBM MQ JMS , zgodnie z opisem w sekcji Konfigurowanie narzędzia administracyjnego JMS. |
| PSIVTRun [1] | Uruchamia program testowy do weryfikowania instalacji publikowania/subskrypcji zgodnie z opisem w sekcji "Narzędzie do weryfikowania instalacji (IVT) publikowania/subskrypcji dla produktu IBM MQ classes for JMS" na stronie 120. |
| PSReportDump.klasa | Ta klasa jest utrzymywana w celu zachowania zgodności z poprzednimi wersjami, ale nie wykonuje żadnej funkcji. |
| V 9.3.0 JM 3.0<br>V 9.3.0 setjms30env "2" na stronie 129 | Linux AIX W przypadku systemu Jakarta Messaging 3.0ustawia zmienne środowiskowe na potrzeby uruchamiania aplikacji IBM MQ classes for JMS w 32-bitowej wirtualnej maszynie języka Java (JVM) w systemach AIX and Linux , zgodnie z opisem w sekcji "Ustawianie zmiennych środowiskowych dla IBM MQ classes for JMS/Jakarta Messaging" na stronie 97. |
| JMS 2.0 setjmsenv "2" na stronie 129 | Linux AIX W przypadku systemu JMS 2.0ustawia zmienne środowiskowe na potrzeby uruchamiania aplikacji IBM MQ classes for JMS w 32-bitowej wirtualnej maszynie języka Java (JVM) w systemach AIX and Linux , zgodnie z opisem w sekcji "Ustawianie zmiennych środowiskowych dla IBM MQ classes for JMS/Jakarta Messaging" na stronie 97. |
| V 9.3.0 JM 3.0<br>V 9.3.0 setjms30env64 "2" na stronie 129 | Linux AIX W przypadku systemu Jakarta Messaging 3.0ustawia zmienne środowiskowe na potrzeby uruchamiania aplikacji IBM MQ classes for JMS w 64-bitowej maszynie JVM w systemach AIX and Linux , zgodnie z opisem w sekcji "Ustawianie zmiennych środowiskowych dla IBM MQ classes for JMS/Jakarta Messaging" na stronie 97. |

| *Tabela 14. Skrypty udostępniane z produktem IBM MQ classes for JMS i produktem IBM MQ classes for Jakarta Messaging (kontynuacja)* | |
|---|---|
| **Użyteczność** | **Użycie** |
| JMS 2.0 setjmsenv64 "2" na stronie 129 | Linux AIX W przypadku systemu JMS 2.0ustawia zmienne środowiskowe na potrzeby uruchamiania aplikacji IBM MQ classes for JMS w 64-bitowej maszynie JVM w systemach AIX and Linux , zgodnie z opisem w sekcji "Ustawianie zmiennych środowiskowych dla IBM MQ classes for JMS/Jakarta Messaging" na stronie 97. |

**Uwaga:**

1. W systemie Windowsnazwa pliku ma rozszerzenie `.bat`.
2. Te skrypty są dostępne tylko w systemie AIX and Linux . W systemie Windowspo zainstalowaniu produktu IBM MQuruchom komendę **setmqenv**. Więcej informacji na ten temat zawiera sekcja "Ustawianie zmiennych środowiskowych dla IBM MQ classes for JMS/Jakarta Messaging" na stronie 97.

## *Obsługa środowiska OSGi z produktem IBM MQ classes for JMS*

Środowisko OSGi udostępnia środowisko, które obsługuje wdrażanie aplikacji jako pakunków. Pakunki OSGi są dostarczane jako część pliku IBM MQ classes for JMS.

Plik IBM MQ classes for JMS zawiera następujące pakunki OSGi.

**com.ibm.msg.client.osgi.jms*version_number*.jar**
: Wspólna warstwa kodu w IBM MQ classes for JMS. Informacje na temat architektury warstwowej klas IBM MQ dla produktu JMSmożna znaleźć na stronie IBM MQ classes for JMS architecture(IBM MQ classes for JMS architecture).

**com.ibm.msg.client.osgi.jms.prereq_*version_number*.jar**
: Wstępnie wymagane pliki archiwum Java (JAR) dla wspólnej warstwy.

**com.ibm.msg.client.osgi.commonservices.j2se_*version_number*.jar**
: Wspólne usługi dla aplikacji Java Platform, Standard Edition (Java SE).

**com.ibm.msg.client.osgi.nls_*version_number*.jar**
: Komunikaty dla warstwy wspólnej.

**com.ibm.msg.client.osgi.wmq_*version_number*.jar**
: Dostawca przesyłania komunikatów IBM MQ w produkcie IBM MQ classes for JMS. Więcej informacji na temat architektury warstwowej produktu IBM MQ classes for JMSzawiera sekcja Klasy produktuIBM MQ dla architektury JMS.

**com.ibm.msg.client.osgi.wmq.prereq_*version_number*.jar**
: Wstępnie wymagane pliki JAR dla dostawcy przesyłania komunikatów produktu IBM MQ .

**com.ibm.msg.client.osgi.wmq.nls_*version_number*.jar**
: Komunikaty dla dostawcy przesyłania komunikatów produktu IBM MQ .

**com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar**
: V 9.3.0 JM 3.0 V 9.3.0 W przypadku produktu Jakarta Messaging 3.0ten plik JAR umożliwia aplikacjom korzystanie zarówno z pliku IBM MQ classes for JMS , jak i z pliku IBM MQ classes for Java, a także zawiera kod do obsługi komunikatów PCF.

**com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar**
: V 9.3.0 JM 3.0 V 9.3.0 W przypadku systemu Jakarta Messaging 3.0ten plik JAR udostępnia wymagania wstępne dla pliku `com.ibm.mq.jakarta.osgi.allclient_`*version_number*`.jar`.

**com.ibm.mq.osgi.allclient_*version_number*.jar**

JMS 2.0 W przypadku systemu JMS 2.0ten plik JAR umożliwia aplikacjom używanie zarówno pliku IBM MQ classes for JMS , jak i pliku IBM MQ classes for Java, a także zawiera kod do obsługi komunikatów PCF.

**com.ibm.mq.osgi.allclientprereqs_*version_number*.jar**

JMS 2.0 W przypadku systemu JMS 2.0ten plik JAR udostępnia wymagania wstępne dla pliku `com.ibm.mq.osgi.allclient_version_number.jar`.

gdzie *version_number* jest numerem wersji zainstalowanego produktu IBM MQ .

Pakunki są instalowane w podkatalogu `java/lib/OSGi` instalacji IBM MQ lub w folderze `java\lib\OSGi` w systemie Windows.

W produkcie IBM MQ 8.0należy używać pakunków `com.ibm.mq.osgi.allclient_8.0.0.0.jar`i `com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar` dla wszystkich nowych aplikacji. Użycie tych pakunków powoduje usunięcie ograniczenia uniemożliwiającego uruchomienie zarówno produktu IBM MQ classes for JMS , jak i produktu IBM MQ classes for Java w ramach tego samego środowiska OSGi. Jednak wszystkie inne ograniczenia nadal mają zastosowanie.

Pakunek `com.ibm.mq.osgi.java`*version_number*`.jar`, który jest również instalowany w podkatalogu `java/lib/OSGi` instalacji IBM MQ lub w folderze `java\lib\OSGi` w systemie Windows, jest częścią katalogu IBM MQ classes for Java. Ten pakunek nie może być załadowany do środowiska wykonawczego OSGi z załadowanym IBM MQ classes for JMS .

Pakunki OSGi dla IBM MQ classes for JMS zostały zapisane w specyfikacji OSGi Release 4. Nie działają one w środowisku OSGi Release 3.

Należy poprawnie ustawić ścieżkę systemową lub ścieżkę do biblioteki, aby środowisko wykonawcze OSGi mogło znaleźć wszystkie wymagane pliki DLL lub biblioteki współużytkowane.

Jeśli dla IBM MQ classes for JMSużywane są pakunki OSGi, tematy tymczasowe nie będą działać. Ponadto klasy wyjścia kanału napisane w języku Java nie są obsługiwane z powodu problemu związanego z ładowaniem klas w środowisku z wieloma programami ładującymi klasy, takim jak OSGi. Pakunek użytkownika może mieć informacje o pakunkach IBM MQ classes for JMS , ale pakunki IBM MQ classes for JMS nie mają informacji o żadnym pakunku użytkownika. W rezultacie program ładujący klasy używany w pakunku IBM MQ classes for JMS nie może załadować klasy wyjścia kanału, która znajduje się w pakunku użytkownika.

Więcej informacji na temat środowiska OSGi zawiera serwis WWW OSGi Alliance .

## z/OS MQ Adv. VUE Połączenia klienta JMS/Jakarta Messaging z aplikacjami wsadowymi działającymi w systemie z/OS

W pewnych warunkach aplikacja IBM MQ classes for JMS/Jakarta Messaging w systemie z/OS może nawiązać połączenie z menedżerem kolejek w systemie z/OS przy użyciu połączenia klienta. Użycie połączenia klienckiego może uprościć topologie IBM MQ .

Używając połączenia klienckiego, jeśli aplikacja IBM MQ classes for JMS/Jakarta Messaging jest uruchomiona w środowisku wsadowym i spełniony jest jeden z następujących warunków, aplikacja może nawiązać połączenie ze zdalnym menedżerem kolejek produktu z/OS :

- V 9.3.4 LTS Kod IBM MQ classes for JMS/Jakarta Messaging jest dostępny w wersji IBM MQ 9.3.4 lub nowszej albo Long Term Support z zastosowaną poprawką APAR PH56722 . Menedżer kolejek może być w dowolnej obsługiwanej wersji.
- Menedżer kolejek, z którym nawiązywane jest połączenie, działa z upoważnieniem IBM MQ Advanced for z/OS Value Unit Edition i dlatego ma parametr **ADVCAP** ustawiony na wartość ENABLED. Menedżer kolejek może być w dowolnej obsługiwanej wersji.

Więcej informacji na temat IBM MQ Advanced for z/OS Value Unit Edition zawiera sekcja Identyfikatory produktówIBM MQ i informacje o eksporcie.

See WYŚWIETLENIE QMGR for more information on **ADVCAP** and URUCHOM QMGR for more information on **QMGRPROD**.

Należy zauważyć, że zadanie wsadowe jest jedynym obsługiwanym środowiskiem. Nie jest obsługiwane środowisko JMS/Jakarta Messaging dla systemu CICS lub JMS/Jakarta Messaging dla systemu IMS.

Aplikacja IBM MQ classes for JMS/Jakarta Messaging w systemie z/OS nie może używać połączenia w trybie klienta do nawiązywania połączenia z menedżerem kolejek, który nie jest uruchomiony w systemie z/OS.

Jeśli aplikacja IBM MQ classes for JMS/Jakarta Messaging w systemie z/OS próbuje nawiązać połączenie w trybie klienta i nie jest to dozwolone, generowany jest komunikat o wyjątku JMSFMQ0005 .

### Obsługa Advanced Message Security (AMS)

Aplikacje klienckie IBM MQ classes for JMS/Jakarta Messaging mogą korzystać z produktu AMS podczas nawiązywania połączeń ze zdalnymi menedżerami kolejek produktu z/OS , z zastrzeżeniem warunków opisanych wcześniej w tym temacie.

Aby używać produktu AMS w ten sposób, aplikacje klienckie muszą używać magazynu kluczy typu `jceracfks` w pliku `keystore.conf`, gdzie:

- Przedrostek nazwy właściwości to `jceracfks` , a w przedrostku nazwy nie jest rozróżniana wielkość liter.
- Magazyn kluczy jest plikiem kluczy RACF .
- Hasła nie są wymagane i zostaną zignorowane. Dzieje się tak dlatego, że pliki kluczy RACF nie używają haseł.
- Jeśli zostanie określony dostawca, musi to być dostawca `IBMJCE`.

W przypadku użycia opcji `jceracfks` z opcją AMSplik kluczy musi mieć następującą postać: `safkeyring://`*`user`*`/`*`keyring`*, gdzie:

- `safkeyring` jest literałem i w nazwie nie jest rozróżniana wielkość liter
- *`user`* to identyfikator użytkownika RACF , który jest właścicielem pliku kluczy.
- *`keyring`* to nazwa pliku kluczy RACF , a w nazwie pliku kluczy rozróżniana jest wielkość liter

W poniższym przykładzie używany jest standardowy plik kluczy AMS dla użytkownika `JOHNDOE`:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

**Pojęcia pokrewne**

“Połączenia klienta Java z aplikacjami wsadowymi działającymi w systemie z/OS” na stronie 382
W pewnych warunkach aplikacja IBM MQ classes for Java w systemie z/OS może nawiązać połączenie z menedżerem kolejek w systemie z/OS przy użyciu połączenia klienta. Użycie połączenia klienckiego może uprościć topologie IBM MQ .

## Uzyskiwanie osobno produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging

Biblioteki i programy narzędziowe niezbędne do uruchamiania aplikacji przy użyciu programu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging są dostępne w samorozpakowujących się plikach JAR pobranych z serwisu Fix Central. Można to zrobić, aby uzyskać tylko te pliki, na przykład w celu wdrożenia w narzędziu do zarządzania oprogramowaniem lub w celu użycia z autonomicznymi aplikacjami klienckimi.

### Zanim rozpoczniesz

Przed rozpoczęciem tego zadania należy się upewnić, że na komputerze jest zainstalowane środowisko Java runtime environment (JRE) i że środowisko JRE zostało dodane do ścieżki systemowej.

Instalator Java , który jest używany w tym procesie instalacji, nie wymaga uruchomienia przez użytkownika root ani przez żadnego konkretnego użytkownika. Jedynym wymaganiem jest, aby użytkownik, dla którego jest on uruchamiany, miał prawo do zapisu w katalogu, w którym mają być zapisywane pliki.

V 9.3.0 JM 3.0 V 9.3.0 W produkcie IBM MQ 9.3.0produkt Jakarta Messaging 3.0 jest obsługiwany na potrzeby tworzenia nowych aplikacji. Produkt IBM MQ 9.3.0 nadal obsługuje produkt JMS 2.0 dla istniejących aplikacji. Nie jest obsługiwane używanie zarówno interfejsu API Jakarta Messaging 3.0 , jak i interfejsu API JMS 2.0 w tej samej aplikacji. Więcej informacji na ten temat zawiera sekcja Używanie klas IBM MQ dla przesyłania komunikatów JMS/Jakarta.

## O tym zadaniu

Samorozpakowujący się plik JAR jest używany w celu zminimalizowania wielkości pobieranego pliku i czasu potrzebnego na wyodrębnienie. Dokładna zawartość tego pliku JAR i podkatalogów, do których pliki są wyodrębniane, zależy od wersji pliku IBM MQ.

Po uruchomieniu samorozpakowującego pliku JAR wyświetlana jest umowa licencyjna IBM MQ , która musi zostać zaakceptowana. Umożliwia również zmianę katalogu macierzystego dla ekstrakcji.

W systemie IBM MQ 9.3samorozpakowujący się plik JAR wyodrębnia pliki do następującej struktury katalogów:

**wmq/JavaEE**
Pliki EAR i RAR adaptera zasobów produktu IBM MQ .

JMS 2.0 Następujące pliki są przeznaczone do użycia z obiektami JMS 2.0 i JMS 1.1 :

- `wmq.jmsra.ivt.ear`
- `wmq.jmsra.rar`

JM 3.0 Istnieje również równoważny zestaw do użycia z obiektami Jakarta Messaging 3.0 :

- `wmq.jakarta.jmsra.ivt.ear`. Zawiera pliki testu sprawdzającego instalację.
- `wmq.jakarta.jmsra.rar`. Zawiera pliki adaptera zasobów.

**wmq/JavaSE**

**wmq/JavaSE/bin**
Narzędzia **JMSAdmin** i **JMS30Admin** . Służy do definiowania obiektów JNDI , które reprezentują obiekty JMS lub Jakarta Messaging .

JMS 2.0 Następujące pliki są przeznaczone do użycia z obiektami JMS 2.0 i JMS 1.1 :

- `JMSAdmin.bat`
- `JMSAdmin`
- `JMSAdmin.config`

JM 3.0 Istnieje również równoważny zestaw do użycia z obiektami Jakarta Messaging 3.0 :

- `JMS30Admin.bat`. Plik, który jest używany do uruchamiania narzędzia w systemie Windows.
- `JMS30Admin`. Skrypt używany do uruchamiania narzędzia na platformach Linux i UNIX .
- `JMS30Admin.config`. Przykładowy plik konfiguracyjny narzędzia.

**Uwaga:**

- Przed dodaniem narzędzia **JMSAdmin** do samorozpakowującego się pliku JAR pliki w tym katalogu znajdowały się w katalogu macierzystym `wmq/JavaSE`.
- Klient, który jest instalowany przy użyciu samorozpakowującego się pliku JAR, może użyć narzędzia **JMSAdmin** lub **JMS30Admin** do utworzenia obiektów administrowanych przesyłania komunikatów produktu Java w kontekście systemu plików (plik `.bindings` ). Klient może również wyszukać i używać tych obiektów administrowanych.

- JMS 2.0 Narzędzie **JMSAdmin** używane z obiektami JMS 2.0 i JMS 1.1 zostało dodane do samorozpakowującego się pliku JAR w katalogu IBM MQ 9.2.0 Fix Pack 2 i IBM MQ 9.2.2.
- JM 3.0 Narzędzie **JMS30Admin** używane z obiektami Jakarta Messaging 3.0 zostało dodane do samorozpakowującego się pliku JAR w katalogu IBM MQ 9.3.0.

**wmq/JavaSE/lib**

Produkt Advanced Message Security używa następujących pakietów Bouncy Castle typu open source do obsługi składni komunikatów szyfrujących (CMS). Patrz serwis Support for non-IBM JREs with AMS.

V 9.3.5 W systemie Continuous Delivery z serwisu IBM MQ 9.3.5:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

LTS W systemach Long Term Support i Continuous Delivery przed IBM MQ 9.3.5:

- `bcpkix-jdk15on.jar`
- `bcprov-jdk15on.jar`
- `bcutil-jdk15on.jar`

Każdy z następujących plików zawiera klasy dla określonego poziomu JMS lub Jakarta Messaging :

- JMS 2.0 `com.ibm.mq.allclient.jar` (JMS 2.0 i JMS 1.1)
- JM 3.0 `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0)

Inne wstępnie wymagane pliki JAR:

- V 9.3.3 Removed `com.ibm.mq.traceControl.jar`. Służy do dynamicznego sterowania śledzeniem dla aplikacji IBM MQ classes for JMS .
- `fscontext.jar`. Wymagany, jeśli aplikacja wykonuje wyszukiwania JNDI przy użyciu kontekstu systemu plików.
- V 9.3.3 `jackson-annotations.jar`, `jackson-core.jar`, `jackson-databind.jar`: zawiera klasy używane do wykonywania odwzorowań CipherSuite i CipherSpec podczas tworzenia bezpiecznych połączeń TLS z menedżerem kolejek.
- JM 3.0 `jakarta.jms-api.jar`. Zawiera definicje interfejsu Jakarta Messaging 3.0 i wyjątków.
- JMS 2.0 `jms.jar`. Zawiera definicje interfejsu JMS 2.0 i wyjątków.
- `org.json.jar`. Zawiera klasy, które umożliwiają produktowi IBM MQ classes for JMS interpretowanie plików CCDT w formacie JSON.
- `providerutil.jar`. Wymagany, jeśli aplikacja wykonuje wyszukiwania JNDI przy użyciu kontekstu systemu plików.

**Uwaga:** Stabilized `com.ibm.mq.allclient.jar` i `com.ibm.mq.jakarta.client.jar` zawierają kopię pliku IBM MQ classes for Java. Jednak w systemie IBM MQ 9.0klasy te są deklarowane jako funkcjonalnie ustabilizowane na poziomie dostarczonym z produktem IBM MQ 8.0. Patrz sekcja Dezaktualizacje, stabilizacje i usunięcia w wersji IBM MQ 9.0.

**wmq/OSGi**

Pakunki klienta OSGi produktu IBM MQ :

- 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

gdzie *V.R.M.F* jest numerem wersji, wydania, modyfikacji i pakietu poprawek.

## Procedura

1. Pobierz plik JAR klienta IBM MQ Java / JMS z serwisu Fix Central.
   a) Kliknij ten odsyłacz: IBM MQ Java / JMS klient.
   b) Znajdź klienta dla danej wersji produktu IBM MQ na wyświetlonej liście dostępnych poprawek.
      Na przykład:

      ```
      release level: 9.3.0.0-IBM-MQ-Install-Java-All
      Long Term Support: 9.3.0.0 IBM MQ JMS and Java 'All Client'
      ```

      Następnie kliknij nazwę pliku klienta i postępuj zgodnie z procedurą pobierania.
2. Rozpocznij wyodrębnianie z katalogu, do którego pobrano plik.
   Aby rozpocząć wyodrębnianie, wprowadź komendę w następującym formacie:

   ```
   java -jar V.R.M.F-IBM-MQ-Install-Java-All.jar
   ```

   gdzie *V.R.M.F* jest numerem wersji produktu, na przykład `9.3.0.0`, a *V.R.M.F*`-IBM-MQ-Install-Java-All.jar` jest nazwą pliku, który został pobrany z serwisu Fix Central.
   Na przykład, aby wyodrębnić klienta JMS dla wersji IBM MQ 9.3.0 , należy użyć następującej komendy:

   ```
   java -jar 9.3.0.0-IBM-MQ-Install-Java-All.jar
   ```

   **Uwaga:** Aby przeprowadzić tę instalację, na komputerze musi być zainstalowane środowisko JRE i dodane do ścieżki systemowej.

   Po wprowadzeniu komendy wyświetlane są następujące informacje:

   ```
   Przed użyciem, wyodrębnieniem lub zainstalowaniem produktu IBM MQ V9.3należy zaakceptować
   1. IBM International License Agreement for Evaluation of
   Programy 2. IBM International Program License Agreement i dodatkowe
   informacje licencyjne. Przeczytaj uważnie poniższe umowy licencyjne.

   Umowę licencyjną można wyświetlić oddzielnie za pomocą
   --viewLicenseopcja umowy licencyjnej.

   Naciśnij klawisz Enter, aby wyświetlić teraz warunki licencji, lub 'x', aby pominąć.
   ```
3. Przejrzyj i zaakceptuj warunki licencji:
   a) Aby wyświetlić licencję, naciśnij klawisz Enter.
      Alternatywnie naciśnij klawisz x, aby pominąć wyświetlanie licencji.
      Po wyświetleniu licencji lub natychmiast po naciśnięciu klawisza x zostanie wyświetlony następujący komunikat:

      ```
      Dodatkowe informacje licencyjne można wyświetlać oddzielnie przy użyciu
      --viewLicenseOpcja informacji.

      Naciśnij klawisz Enter, aby wyświetlić teraz dodatkowe informacje licencyjne, lub 'x', aby
      pominąć.
      ```
   b) Aby wyświetlić dodatkowe warunki licencji, naciśnij klawisz Enter.
      Można również nacisnąć klawisz x, aby pominąć wyświetlanie dodatkowych warunków licencji.
      Po wyświetleniu dodatkowych warunków licencji lub natychmiast po naciśnięciu klawisza x zostanie wyświetlony następujący komunikat:

      ```
      Wybierając opcję "Zgadzam się" poniżej, zgadzasz się na warunki
      Umowa licencyjna i warunki inne niżIBM (jeśli mają zastosowanie). Jeśli ten parametr nie
      zostanie podany
      zgadzam się, wybierz opcję "Nie zgadzam się".
      ```

```
Wybierz opcję [ 1] Zgadzam się lub [ 2] Nie zgadzam się:
```

c) Aby zaakceptować umowę licencyjną i kontynuować wybieranie katalogu instalacyjnego, wybierz 1.

Alternatywnie wybierz opcję 2, aby natychmiast zakończyć instalację.

Jeśli zostanie wybrana wartość 1, zostanie wyświetlony komunikat podobny do następującego:

```
Podaj katalog dla plików produktu lub pozostaw to pole puste, aby zaakceptować wartość
domyślną.
Domyślnym katalogiem docelowym jest H: \downloads

Katalog docelowy dla plików produktu?
```

4. Określ katalog nadrzędny dla ekstrakcji.

Domyślnym położeniem jest katalog bieżący.

- Jeśli chcesz wyodrębnić pliki produktu do położenia domyślnego, naciśnij klawisz Enter bez określania wartości.
- Aby wyodrębnić pliki produktu do innego położenia, podaj nazwę katalogu, do którego mają zostać wyodrębnione pliki, a następnie naciśnij klawisz Enter, aby rozpocząć wyodrębnianie.

Podana nazwa katalogu nie może jeszcze istnieć, w przeciwnym razie po rozpoczęciu wyodrębniania zgłaszany jest błąd i nie są instalowane żadne pliki.

Jeśli ten katalog jeszcze nie istnieje, zostanie utworzony określony katalog, a pliki programu zostaną wyodrębnione do tego katalogu. Podczas instalacji w podanym katalogu nadrzędnym zostanie utworzony nowy katalog o nazwie `wmq` .

W katalogu `wmq` tworzone są trzy podkatalogi `JavaEE`, `JavaSE`i `OSGio` następującej zawartości:

**JavaEE**

JM 3.0 `wmq.jakarta.jmsra.ivt.ear`

JM 3.0 `wmq.jakarta.jmsra.rar`

JMS 2.0 `wmq.jmsra.ivt.ear`

JMS 2.0 `wmq.jmsra.rar`

**JavaSE**

Ten katalog zawiera następujące podkatalogi i pliki:

**JavaSE/lib**

V 9.3.5 `bcpkix-jdk18on.jar`

LTS `bcpkix-jdk15on.jar`

V 9.3.5 `bcprov-jdk18on.jar`

LTS `bcprov-jdk15on.jar`

V 9.3.5 `bcutil-jdk18on.jar`

LTS `bcutil-jdk15on.jar`

JMS 2.0 `com.ibm.mq.allclient.jar`

JM 3.0 `com.ibm.mq.jakarta.client.jar`

V 9.3.3 Removed `com.ibm.mq.traceControl.jar`

`fscontext.jar`

V 9.3.3 `jackson-annotations.jar`

V 9.3.3 `jackson-core.jar`

V 9.3.3 jackson-databind.jar

jms.jar

org.json.jar

providerutil.jar

**JavaSE/bin**

JMSAdmin.bat

JMSAdmin

JMSAdmin.config

**Środowisko OSGi**

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

Po zakończeniu wyodrębniania zostanie wyświetlony komunikat z potwierdzeniem, jak pokazano w poniższym przykładzie:

```
Rozpakowywanie plików do H: \downloads\wmq
Successfully extracted all product files.
```

## Allowlisting w IBM MQ classes for JMS/Jakarta Messaging

Mechanizm serializacji i deserializacji obiektu Java został zidentyfikowany jako potencjalne zagrożenie bezpieczeństwa. Dozwolone listy w systemach IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging zapewniają pewną ochronę przed ryzykiem związanym z serializacją.

### O tym zadaniu

Mechanizm serializacji i deserializacji obiektu Java został zidentyfikowany jako potencjalne ryzyko związane z bezpieczeństwem, ponieważ deserializacja tworzy instancje dowolnych obiektów Java , w których istnieje możliwość, że złośliwie wysłane dane mogą powodować różne problemy. Jedna zauważalna aplikacja serializacji znajduje się w produkcie Jakarta Messaging 3.0 i w produkcie Java Message Service 2.0 ObjectMessages , które używają serializacji do hermetyzowania i przesyłania dowolnych obiektów.

Lista zaakceptowanych operacji przekształcania do postaci szeregowej jest potencjalną mitygacją niektórych czynników ryzyka, które są związane z serializacją. Jawne określenie, które klasy mogą być hermetyzowane w komunikatach ObjectMessagesi z nich wyodrębniane, powoduje, że lista zaakceptowanych klas zapewnia pewną ochronę przed ryzykiem związanym z serializacją.

**Pojęcia pokrewne**

“Uruchamianie aplikacji IBM MQ classes for JMS w katalogu Java security manager” na stronie 111
Program IBM MQ classes for JMS można uruchomić z włączoną opcją Java security manager .
Aby pomyślnie uruchomić aplikacje z włączoną opcją Java security manager , należy skonfigurować środowisko Java Virtual Machine (JVM) przy użyciu odpowiedniego pliku konfiguracyjnego strategii.

### *Zezwalaj na listę pojęć*

W produkcie IBM MQ classes for JMS i IBM MQ classes for Jakarta Messagingistnieje obsługa zezwalania na wyświetlanie klas w implementacji interfejsu JMS ObjectMessage . Umożliwia to potencjalne łagodzenie niektórych zagrożeń dla bezpieczeństwa, które mogą być związane z mechanizmem przekształcania obiektu Java do postaci szeregowej i z postaci szeregowej.

## Allowlisting w IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging

**Ważne:**

Tam, gdzie to możliwe, termin *lista zaakceptowanych* (allowlist) jest stosowany w miejsce terminu *biała lista* (whitelist). W przypadku systemu IBM MQ 9.0 i nowszych wersji obejmuje to niektóre nazwy właściwości systemowych Java wymienione w tym temacie. Nie jest konieczna zmiana dotychczasowych konfiguracji. Dotychczasowe nazwy właściwości systemowych również będą działać.

IBM MQ classes for JMS (JMS 2.0) V 9.3.0 V 9.3.0 i IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) obsługują listę dozwolonych klas w implementacji interfejsu JMS ObjectMessage .

- JMS 2.0 W systemie IBM MQ classes for JMSodpowiednie nazwy właściwości to **com.ibm.mq.jms.allowlist.***.
- JM 3.0 W systemie IBM MQ classes for Jakarta Messagingodpowiednie nazwy właściwości to **com.ibm.mq.jakarta.jms.allowlist.***

Lista zaakceptowanych definiuje, które klasy Java mogą być przekształcane do postaci szeregowej za pomocą metody ObjectMessage.setObject() i przekształcane z postaci szeregowej za pomocą metody ObjectMessage.getObject().

- JMS 2.0 Próba przekształcenia do postaci szeregowej lub przekształcenia z postaci szeregowej instancji klasy, która nie znajduje się na liście zaakceptowanych ObjectMessage , powoduje zgłoszenie wyjątku javax.jms.MessageFormatException z przyczyną java.io.InvalidClassException .
- JM 3.0 Próba przekształcenia do postaci szeregowej lub przekształcenia z postaci szeregowej instancji klasy, która nie znajduje się na liście zaakceptowanych, przy użyciu elementu ObjectMessage powoduje zgłoszenie wyjątku jakarta.jms.MessageFormatException z wyjątkiem java.io.InvalidClassException jako jego przyczyną.

## Tworzenie listy zaakceptowanych

**Ważne:** Produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging nie można dystrybuować z listą zaakceptowanych. Wybór klas do przesłania przy użyciu właściwości ObjectMessages jest wyborem projektu aplikacji i IBM MQ nie może tego poprzedzać.

Z tego powodu mechanizm allowlisting zezwala na dwa tryby działania:

**Wykrywanie**
W tym trybie mechanizm tworzy listę pełnych nazw klas, zgłaszając wszystkie klasy, które zostały zaobserwowane jako przekształcone do postaci szeregowej lub przekształcone z postaci szeregowej w klasie ObjectMessages.

**Wymuszanie**
W tym trybie mechanizm wymusza wyświetlanie listy zaakceptowanych, odrzucając próby przekształcenia do postaci szeregowej lub przekształcenia z postaci szeregowej klas, które nie znajdują się na liście zaakceptowanych.

Aby użyć tego mechanizmu, należy najpierw uruchomić tryb wykrywania (DISCOVERY) w celu zebrania listy obecnie przekształconych do postaci szeregowej i przekształconych z postaci szeregowej klas, przejrzeć listę i użyć jej jako podstawy dla listy zaakceptowanych. Może być nawet wskazane użycie listy bez zmian, ale przed podjęciem takiej decyzji należy przejrzeć listę.

## Kontrolowanie mechanizmu listy zaakceptowanych

Dostępne są trzy właściwości systemowe umożliwiające sterowanie mechanizmem wyświetlania listy zaakceptowanych:

**com.ibm.mq.jms.allowlist (JMS 2.0) i com.ibm.mq.jakarta.jms.allowlist (Jakarta Messaging 3.0)**
Tę właściwość można określić w jeden z następujących sposobów:

- Nazwa ścieżki do pliku, który zawiera listę zaakceptowanych, w formacie identyfikatora URI pliku (rozpoczynającego się od `file:`). W trybie wykrywania (DISCOVERY) ten plik jest zapisywany przez

mechanizm allowlisting. Plik nie może istnieć. Jeśli plik istnieje, mechanizm zgłasza wyjątek zamiast go zastępować. W trybie ENFORCEMENT ten plik jest odczytywany przez mechanizm allowlisting.

- Rozdzielane przecinkami pełne nazwy klas, które tworzą listę zaakceptowanych.

Jeśli ta właściwość nie jest ustawiona, mechanizm listy zaakceptowanych jest nieaktywny.

Jeśli używany jest plik Java security manager, należy upewnić się, że pliki JAR IBM MQ classes for JMS mają dostęp do odczytu i zapisu do tego pliku.

**com.ibm.mq.jms.allowlist.discover (JMS 2.0) i com.ibm.mq.jakarta.jms.allowlist.discover (Jakarta Messaging 3.0)**

- Jeśli ta właściwość nie jest ustawiona lub ma wartość false, mechanizm listy zaakceptowanych działa w trybie ENFORCEMENT.
- Jeśli ta właściwość ma wartość true, a lista zaakceptowanych została określona jako identyfikator URI pliku, mechanizm listy zaakceptowanych działa w trybie wykrywania (DISCOVERY).
- Jeśli ta właściwość jest ustawiona na wartość true, a lista zaakceptowanych została określona jako lista nazw klas, mechanizm listy zaakceptowanych zgłasza odpowiedni wyjątek.
- Jeśli ta właściwość ma wartość true, a lista zaakceptowanych nie została określona przy użyciu właściwości com.ibm.mq.jms.allowlist lub com.ibm.mq.jakarta.jms.allowlist, mechanizm listy zaakceptowanych jest nieaktywny.
- Jeśli ta właściwość ma wartość true, a plik listy zaakceptowanych już istnieje, mechanizm listy zaakceptowanych zgłasza wyjątek java.io.InvalidClassException , a wpisy nie są dodawane do pliku.

**com.ibm.mq.jms.allowlist.mode (JMS 2.0) i com.ibm.mq.jakarta.jms.allowlist.mode (Jakarta Messaging 3.0)**

Tę właściwość łańcuchową można określić na jeden z trzech sposobów:

- Jeśli ta właściwość jest ustawiona na wartość SERIALIZE, tryb wymuszania wykonuje sprawdzanie poprawności listy zaakceptowanych tylko dla metody ObjectMessage.setObject().
- Jeśli ta właściwość jest ustawiona na wartość DESERIALIZE, tryb wymuszania wykonuje sprawdzanie poprawności listy zaakceptowanych tylko dla metody ObjectMessage.getObject().
- Jeśli ta właściwość nie jest ustawiona lub ma inną wartość, tryb ENFORCEMENT wykonuje sprawdzanie poprawności listy zaakceptowanych zarówno dla metody ObjectMessage.getObject(), jak i dla metody ObjectMessage.setObject().

## Format pliku listy zaakceptowanych

Oto główne cechy formatu pliku listy zaakceptowanych:

- Plik listy zaakceptowanych ma domyślne kodowanie pliku platformy z odpowiednimi dla platformy końcami wierszy.

  **Uwaga:** Jeśli używany jest plik listy zaakceptowanych, plik ten jest zawsze zapisywany i odczytywany przy użyciu domyślnego kodowania plików dla maszyny JVM.

  Jest to możliwe, jeśli plik listy zaakceptowanych jest generowany w jeden z następujących sposobów:

  – z/OS Generowane przez autonomiczną aplikację działającą w systemie z/OS i używane przez inne autonomiczne aplikacje działające również w systemie z/OS.

  – Generowany przez aplikację działającą w produkcie WebSphere Application Server na dowolnej platformie i używany przez inną instancję produktu WebSphere Application Server.

  – Multi Generowany przez autonomiczną aplikację działającą w systemie IBM MQ for Multiplatformsi używany przez inne autonomiczne aplikacje działające w systemie IBM MQ for Multiplatformslub przez aplikacje działające w systemie WebSphere Application Server na dowolnej platformie.

Ponieważ jednak produkt WebSphere Application Server używa kodu ASCII, a autonomiczna maszyna JVM używa kodu EBCDIC, wystąpią problemy z kodowaniem plików, jeśli plik listy zaakceptowanych zostanie wygenerowany w jeden z następujących sposobów:

  - Generowany w systemie z/OS, następnie używany przez autonomiczne aplikacje działające na platformie innej niż z/OS lub WebSphere Application Server.
  - Generowane przez program WebSphere Application Server lub autonomiczną aplikację działającą na platformie innej niż z/OS, a następnie używane przez autonomiczną aplikację działającą w systemie z/OS.
- Każdy niepusty wiersz zawiera pełną nazwę klasy. Puste wiersze są ignorowane.
- Komentarze mogą być uwzględniane-wszystko, co następuje po znaku '#', do końca wiersza, jest ignorowane.
- Istnieje bardzo podstawowy mechanizm dzikiej zwierzyny:
  - '*' może być **ostatnim** elementem nazwy klasy.
  - Znak '*' odpowiada **pojedynczemu** elementowi nazwy klasy, czyli klasie, ale nie części pakietu.

  Dlatego łańcuch `com.ibm.mq.*` będzie zgodny z wartością `com.ibm.mq.MQMessage` , ale nie z wartością `com.ibm.mq.jmqi.remote.api.RemoteFAP`.

  Znaki wieloznaczne nie działają dla klas w pakiecie domyślnym, który jest dla klas bez jawnej nazwy pakietu, dlatego nazwa klasy "*" jest odrzucana.
- Nieprawidłowo sformatowane pliki allowlist, na przykład pliki zawierające wpis `com.ibm.mq.*.Message`, w którym znak wieloznaczny nie jest ostatnim elementem, powodują zgłoszenie wyjątku java.lang.IllegalArgumentException .
- Pusty plik allowlist powoduje całkowite wyłączenie użycia elementu ObjectMessage.

## Format listy zaakceptowanych jako listy rozdzielanej przecinkami

Ten sam mechanizm obsługi znaków wieloznacznych jest dostępny dla listy zaakceptowanych jako lista rozdzielana przecinkami.

- Znak '*' może zostać rozwinięty przez system operacyjny, jeśli zostanie podany w wierszu komend, w skrypcie powłoki lub w pliku wsadowym, dlatego może wymagać specjalnej obsługi.
- Znak komentarza '#' ma zastosowanie tylko wtedy, gdy określony jest plik. Jeśli parametr allowlist jest określony jako rozdzielana przecinkami lista nazw klas, to przy założeniu, że system operacyjny lub powłoka nie przetwarza go, ponieważ jest to domyślny znak komentarza w wielu powłokach AIX and Linux , jest on traktowany jako zwykły znak.

## Kiedy następuje zezwolenie na umieszczenie w wykazie?

Metoda allowlisting jest inicjowana, gdy aplikacja po raz pierwszy uruchamia metodę ObjectMessage setMessage() lub getMessage().

Właściwości systemowe są wartościowane, otwierany jest plik listy zaakceptowanych, a w trybie wymuszania (ENFORCEMENT) lista klas listy zaakceptowanych jest ładowana podczas inicjowania mechanizmu. W tym momencie wpis jest zapisywany w pliku dziennika IBM MQ JMS dla aplikacji.

Po zainicjowaniu mechanizmu jego parametry mogą nie zostać zmienione. Ponieważ czas inicjowania nie jest łatwo przewidywany, ponieważ zależy on od zachowania aplikacji. Dlatego też ustawienia właściwości systemowych i zawartość pliku listy zaakceptowanych powinny być traktowane jako ustalone od momentu uruchomienia aplikacji. Nie należy zmieniać właściwości ani zawartości pliku listy zaakceptowanych podczas działania aplikacji, ponieważ wyniki nie są gwarantowane.

## Punkty do rozważenia

Najlepszym podejściem do łagodzenia ryzyka wbudowanego w mechanizm serializacji Java byłoby zbadanie alternatywnych metod przesyłania danych, takich jak użycie formatu JSON zamiast komunikatu

ObjectMessage. Za pomocą mechanizmów AMS ( Advanced Message Security ) można zwiększyć bezpieczeństwo, upewniając się, że komunikaty pochodzą z zaufanych źródeł.

Jeśli z aplikacją używany jest mechanizm Java security manager , należy nadać następujące uprawnienia:

- FilePermission w dowolnym pliku listy zaakceptowanych, który jest używany, z uprawnieniem do odczytu w trybie ENFORCEMENT, z uprawnieniem do zapisu w trybie DISCOVER.
- JMS 2.0 PropertyPermission (odczyt) w właściwościach **com.ibm.mq.jms.allowlist**, **com.ibm.mq.jms.allowlist.discover**i **com.ibm.mq.jms.allowlist.mode** .
- JM 3.0 PropertyPermission (odczyt) w właściwościach **com.ibm.mq.jakarta.jms.allowlist**, **com.ibm.mq.jakarta.jms.allowlist.discover**i **com.ibm.mq.jakarta.jms.allowlist.mode** .

## Więcej informacji

Więcej informacji na temat list zaakceptowanych zawierają “Konfigurowanie i używanie listy zaakceptowanych JMS lub Jakarta Messaging” na stronie 140 i “Allowlisting w WebSphere Application Server” na stronie 142 .

**Pojęcia pokrewne**

“Uruchamianie aplikacji IBM MQ classes for JMS w katalogu Java security manager” na stronie 111
Program IBM MQ classes for JMS można uruchomić z włączoną opcją Java security manager .
Aby pomyślnie uruchomić aplikacje z włączoną opcją Java security manager , należy skonfigurować środowisko Java Virtual Machine (JVM) przy użyciu odpowiedniego pliku konfiguracyjnego strategii.

# *Konfigurowanie i używanie listy zaakceptowanych JMS lub Jakarta Messaging*

W tej sekcji opisano sposób działania listy zaakceptowanych oraz sposób jej skonfigurowania przy użyciu funkcji zawartej w pliku IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging w celu wygenerowania pliku listy zaakceptowanych zawierającego listę typów komunikatów ObjectMessages , które mogą być przetwarzane przez aplikację.

## Zanim rozpoczniesz

**Ważne:**

Tam, gdzie to możliwe, termin *lista zaakceptowanych* (allowlist) jest stosowany w miejsce terminu *biała lista* (whitelist). W przypadku systemu IBM MQ 9.0 i nowszych wersji obejmuje to niektóre nazwy właściwości systemowych Java wymienione w tym temacie. Nie jest konieczna zmiana dotychczasowych konfiguracji. Dotychczasowe nazwy właściwości systemowych również będą działać.

Przed rozpoczęciem tej czynności należy zapoznać się z sekcją “Zezwalaj na listę pojęć” na stronie 136

## O tym zadaniu

Ponieważ JMS i Jakarta Messaging współużytkują wiele wspólnych elementów, dalsze odwołania do JMS w tym temacie mogą być traktowane jako odwołania do obu tych elementów. Wszelkie różnice są podświetlane w razie potrzeby.

Po włączeniu funkcji allowlisting program IBM MQ classes for JMS używa tej funkcji w następujący sposób:

- Jeśli aplikacja chce wysłać komunikat ObjectMessage, może go utworzyć na jeden z dwóch sposobów, wywołując:
  - Session.createObjectMessage(Serializable), przekazywanie obiektu, który ma być zawarty w komunikacie.
  - Session.createObjectMessage() method, to create an empty ObjectMessage, a then calling ObjectMessage.setObject(Serializable) to store the object to be sent inside the ObjectMessage.

Po wywołaniu metod Session.createObjectMessage(Serializable) lub ObjectMessage.setObject(Serializable) klasy JMS sprawdzają, czy przekazany obiekt jest typu, który jest wymieniony na liście zaakceptowanych.

Jeśli jest to wymieniony typ, obiekt jest przekształcany do postaci szeregowej i zapisywany w obiekcie ObjectMessage. Jeśli jednak obiekt jest typu, który nie znajduje się na liście zaakceptowanych, usługa IBM MQ classes for JMS zgłasza wyjątek JMSException zawierający komunikat:

```
JMSCC0052: Wystąpił wyjątek podczas przekształcania obiektu do postaci szeregowej:
'java.io.InvalidClassException: < klasa obiektu>; klasa nie może być przekształcona do postaci
szeregowej
lub przekształcony z postaci szeregowej, ponieważ nie został uwzględniony na liście
zaakceptowanych '< allowlist>'.
```

z powrotem do aplikacji.

**Ważne:** Jeśli wyjątek jest zgłaszany przez metodę Session.createObjectMessage(Serializable), komunikat ObjectMessage nie zostanie utworzony. Podobnie, jeśli wyjątek JMSException jest zgłaszany przez metodę ObjectMessage.setObject(Serializable), obiekt nie zostanie dodany do komunikatu ObjectMessage.

- Jeśli aplikacja odbierze komunikat ObjectMessage, wywołuje metodę ObjectMessage.getObject() w celu pobrania zawartego w nim obiektu. Po wywołaniu tej metody IBM MQ classes for JMS sprawdza typ obiektu zawartego w obiekcie ObjectMessage, aby sprawdzić, czy ten obiekt jest typu określonego na liście zaakceptowanych.

  Jeśli tak, obiekt jest przekształcany z postaci szeregowej i zwracany do aplikacji. Jeśli jednak obiekt jest typu, który nie znajduje się na liście zaakceptowanych, usługa IBM MQ classes for JMS zgłasza wyjątek JMSException zawierający komunikat:

  ```
  JMSCC0053: Wystąpił wyjątek podczas przekształcania komunikatu z postaci szeregowej:
  'java.io.InvalidClassException: < object class>; klasa może nie być
  serializowane lub deserializowane, ponieważ nie zostały uwzględnione w
  lista zaakceptowanych '< lista_zaakceptowanych >'. '.
  ```

  z powrotem do aplikacji.

Na przykład załóżmy, że aplikacja zawiera następujący kod w celu wysłania komunikatu ObjectMessage zawierającego obiekt typu java.net.URI:

```
java.net.URL testURL = new java.net.URL("https://www.ibm.com/");
ObjectMessage msg = session.createObjectMessage(testURL);
sender.send(msg);
```

Ponieważ opcja allowlisting nie jest włączona, aplikacja może pomyślnie umieścić komunikat w wymaganym miejscu docelowym.

Jeśli utworzono plik o nazwie `C:\allowlist.txt` zawierający pojedynczy wpis, `java.net.URL`i ponownie uruchomiono aplikację z ustawioną właściwością systemową Java :

```
-Dcom.ibm.mq.jms.allowlist=file:/C:/allowlist.txt
```

Funkcja allowlist jest włączona. Aplikacja nadal może utworzyć i wysłać komunikat ObjectMessage zawierający obiekt typu java.net.URI , ponieważ ten typ jest określony na liście zaakceptowanych.

Jeśli jednak plik `allowlist.txt` zostanie zmieniony w taki sposób, że plik zawiera pojedynczy wpis `java.util.Calendar`, ponieważ funkcja listy zaakceptowanych jest nadal włączona podczas wywoływania przez aplikację:

```
ObjectMessage msg = session.createObjectMessage(testURL);
```

IBM MQ classes for JMS sprawdź listę zaakceptowanych i sprawdź, czy nie zawiera ona wpisu dla java.net.URI.

W wyniku tego zgłaszany jest wyjątek JMSException zawierający komunikat JMSCC0052 .

Podobnie załóżmy, że istnieje inna aplikacja, która odbiera komunikaty ObjectMessages przy użyciu tego kodu:

```
ObjectMessage message = (ObjectMessage)receiver.receive(30000);
if (message != null) {
    Object messageBody = objectMessage.getObject();
    if (messageBody instanceof java.net.URI) {
      :    :    :    :    :    :    :
```

Jeśli opcja allowlisting nie jest włączona, aplikacja może odbierać komunikaty ObjectMessages , które zawierają obiekt dowolnego typu. Przed wykonaniem odpowiedniego przetwarzania aplikacja sprawdza, czy obiekt jest typu java.net.URL .

Jeśli teraz aplikacja zostanie uruchomiona z właściwością systemową Java :

```
-Dcom.ibm.mq.jms.allowlist=java.net.URL
```

, funkcja allowlisting jest włączona. Gdy aplikacja wywołuje:

```
Object messageBody = objectMessage.getObject();
```

Metoda ObjectMessage.getObject() zwraca tylko obiekty typu java.net.URL.

Jeśli obiekt zawarty w obiekcie ObjectMessage nie jest tego typu, metoda ObjectMessage.getObject() zgłasza wyjątek JMSException zawierający komunikat JMSCC0053 . Następnie aplikacja musi zdecydować, co zrobić z komunikatem. Na przykład komunikat może zostać przeniesiony do kolejki niedostarczonych komunikatów dla tego menedżera kolejek.

Aplikacja zwraca wartość normalnie tylko wtedy, gdy obiekt wewnątrz obiektu ObjectMessage jest typu java.net.URL.

## Procedura

1. Uruchom aplikację, która przetwarza komunikaty ObjectMessages, z następującymi właściwościami systemu Java :

   ```
   -Dcom.ibm.mq.jms.allowlist.discover=true
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   Po uruchomieniu aplikacji IBM MQ classes for JMS tworzy plik, który zawiera typy obiektów przetwarzanych przez aplikację.
2. Po przetworzeniu przez aplikację reprezentatywnej próbki komunikatów ObjectMessages w danym okresie należy ją zatrzymać.

   Plik allowlist zawiera teraz listę wszystkich typów obiektów zawartych w elemencie ObjectMessages , które zostały przetworzone przez aplikację podczas jej działania.

   Jeśli aplikacja została uruchomiona przez wystarczający czas, ta lista zawiera wszystkie możliwe typy obiektów zawartych w elemencie ObjectMessages , które aplikacja prawdopodobnie obsłuży.
3. Zrestartuj aplikację z następującym zestawem właściwości systemowych:

   ```
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   Powoduje to włączenie opcji allowlisting i jeśli IBM MQ classes for JMS wykryje komunikat ObjectMessage typu, który nie znajduje się na liście zaakceptowanych, zgłaszany jest wyjątek JMSException zawierający komunikat JMSCC0052 lub JMSCC0053 .

### *Allowlisting w WebSphere Application Server*

Sposób użycia opcji IBM MQ classes for JMS allowlisting w pliku WebSphere Application Server.

**Ważne:**

Tam, gdzie to możliwe, termin *lista zaakceptowanych* (allowlist) jest stosowany w miejsce terminu *biała lista* (whitelist). W przypadku systemu IBM MQ 9.0 i nowszych wersji obejmuje to niektóre nazwy właściwości systemowych Java wymienione w tym temacie. Nie jest konieczna zmiana dotychczasowych konfiguracji. Dotychczasowe nazwy właściwości systemowych również będą działać.

Należy upewnić się, że instalacja produktu WebSphere Application Server zawiera wersję adaptera zasobów IBM MQ , która obsługuje opcję allowlisting.

Więcej informacji na temat korzystania z tych dwóch produktów zawiera sekcja “Używanie produktów IBM MQ i WebSphere Application Server razem” na stronie 517 .

Począwszy od wersji IBM MQ 9.0.0 Fix Pack 1 dostępne są odpowiednie funkcje.

Po zaktualizowaniu serwera aplikacji można użyć właściwości systemowych Java :

- `-Dcom.ibm.mq.jms.allowlist`
- `-Dcom.ibm.mq.jms.allowlist.discover`

opisano w sekcji “Konfigurowanie i używanie listy zaakceptowanych JMS lub Jakarta Messaging” na stronie 140.

**Uwaga:** Aby zmiany odniosły skutek, należy ustawić właściwości systemowe Java jako ogólne argumenty maszyny JVM na serwerze Java Virtual Machine używanym do uruchamiania serwera aplikacji i zrestartować serwer aplikacji.

Więcej informacji na ten temat zawiera sekcja *Ogólne argumenty wirtualnej maszyny języka Java* w sekcji Ustawienia wirtualnej maszyny językaJava .

Aby ustawić właściwości, przejdź do okna Java Virtual Machine w sekcji *Definicje procesów* i wprowadź odpowiedni argument.

Następujące ustawienie:

```
-Dcom.ibm.mq.jms.allowlist=<youruserId>_MyObject
```

powoduje, że serwer aplikacji używa listy zaakceptowanych *id_użytkownika_MyObject*. Tylko obiekty tego typu są przetwarzane przez serwer aplikacji.

Następujące ustawienia:

```
-Dcom.ibm.mq.jms.allowlist.discover=true
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

Skonfigurować serwer aplikacji w taki sposób, aby używał trybu *Wykryj* , i zapisać szczegóły komunikatów JMS ObjectMessagesprzetwarzanych przez serwer aplikacji w pliku `C:\allowlist.txt` .

Następujące ustawienie:

```
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

powoduje załadowanie pliku `C:/allowlist.txt`przez serwer aplikacji i użycie informacji zawartych w tym pliku do określenia listy zaakceptowanych.

**Pojęcia pokrewne**

“Uruchamianie aplikacji IBM MQ classes for JMS w katalogu Java security manager” na stronie 111
Program IBM MQ classes for JMS można uruchomić z włączoną opcją Java security manager .
Aby pomyślnie uruchomić aplikacje z włączoną opcją Java security manager , należy skonfigurować środowisko Java Virtual Machine (JVM) przy użyciu odpowiedniego pliku konfiguracyjnego strategii.

## Konwersje łańcuchów znaków w programie IBM MQ classes for JMS

IBM MQ classes for JMS CharsetEncoders i CharsetDecoders są używane bezpośrednio do konwersji łańcuchów znaków. Domyślne zachowanie konwersji łańcucha znaków można skonfigurować przy użyciu dwóch właściwości systemowych. Obsługę komunikatów, które zawierają znaki niemożliwe do odwzorowania, można skonfigurować za pomocą właściwości komunikatu w celu ustawienia działania UnmappableCharacteri bajtów zastępujących.

Przed wersją IBM MQ 8.0konwersje łańcuchów w pliku IBM MQ classes for JMS były wykonywane przez wywołanie metod `java.nio.charset.Charset.decode(ByteBuffer)` i `Charset.encode(CharBuffer)` .

Użycie jednej z tych metod powoduje domyślne zastąpienie ( REPLACE) zniekształconych lub niemożliwych do przetłumaczenia danych. Takie zachowanie może przesłaniać błędy w aplikacjach i prowadzić do nieoczekiwanych znaków, na przykład ?, w przetłumaczonych danych.

W programie IBM MQ 8.0, aby wykryć takie problemy wcześniej i bardziej efektywnie, IBM MQ classes for JMS należy użyć CharsetEncoders i CharsetDecoders bezpośrednio i skonfigurować obsługę zniekształconych i niemożliwych do przetłumaczenia danych w sposób jawny. Domyślnym zachowaniem jest REPORT takie problemy przez zgłoszenie odpowiedniego `MQException`.

## Konfigurowanie

Translacja z formatu UTF-16 (reprezentacja znaków używana w Java) na rodzimy zestaw znaków, taki jak UTF-8, jest zwana *kodowaniem*, natomiast translacja w przeciwnym kierunku jest zwana *dekodowaniem*.

Dekodowanie przyjmuje domyślne zachowanie programu `CharsetDecoders`i zgłasza błędy, zgłaszając wyjątek.

Jedno ustawienie służy do określenia parametru `java.nio.charset.CodingErrorAction` , który steruje obsługą błędów zarówno przy kodowaniu, jak i dekodowaniu. Inne ustawienie jest używane do sterowania bajtem zastępującym (lub bajtami) podczas kodowania. W operacjach dekodowania zostanie użyty domyślny łańcuch zastępujący Java .

## UnmappableCharacterUstawienia działania i zastępowania bajtów w IBM MQ classes for JMS

W produkcie IBM MQ 8.0dostępne są dwie następujące właściwości umożliwiające ustawienie działania UnmappableCharacteri bajtów zastępujących. Odpowiednie definicje stałych znajdują się w pliku `com.ibm.msg.client.wmq.WMQConstants`.

**JMS_IBM_UNMAPPABLE_ACTION**
Ustawia lub pobiera wartość `CodingErrorAction` , która ma być stosowana, gdy znak nie może zostać odwzorowany w operacji kodowania lub dekodowania.

Należy ustawić wartość `CodingErrorAction.{REPLACE|REPORT|IGNORE}.toString()` w następujący sposób:

```
public static final String JMS_IBM_UNMAPPABLE_ACTION = "JMS_IBM_Unmappable_Action";
```

**JMS_IBM_UNMAPPABLE_REPLACEMENT**
Ustawia lub pobiera bajty zastępujące, które mają zostać zastosowane, gdy znak nie może zostać odwzorowany w operacji kodowania.

W operacjach dekodowania używany jest domyślny łańcuch zastępujący Java .

```
public static final String JMS_IBM_UNMAPPABLE_REPLACEMENT = "JMS_IBM_Unmappable_Replacement";
```

Właściwości `JMS_IBM_UNMAPPABLE_ACTION` i `JMS_IBM_UNMAPPABLE_REPLACEMENT` można ustawić w miejscach docelowych lub komunikatach. Wartość ustawiona w komunikacie nadpisuje wartość ustawioną w miejscu docelowym, do którego wysyłany jest komunikat.

Należy zauważyć, że parametr `JMS_IBM_UNMAPPABLE_REPLACEMENT` musi być ustawiony jako jednobajtowy.

## Właściwości systemowe służące do ustawiania systemowych wartości domyślnych

W produkcie IBM MQ 8.0dostępne są następujące dwie właściwości systemowe Java umożliwiające skonfigurowanie domyślnego zachowania w odniesieniu do konwersji łańcuchów znaków.

**com.ibm.mq.cfg.jmqi.UnmappableCharacterAction**
Określa działanie, które ma zostać podjęte dla niepodlegających tłumaczeniu danych przy kodowaniu i dekodowaniu. Możliwe wartości to REPORT, REPLACElub IGNORE.

**com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement**
Ustawia lub pobiera bajty zastępujące, które mają być stosowane, gdy nie można odwzorować znaku w operacji kodowania. W operacjach dekodowania używany jest domyślny łańcuch zastępujący Java .

Aby uniknąć pomyłek między reprezentacjami znaków Java i bajtów rodzimych, należy określić wartość `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement` jako liczbę dziesiętną reprezentującą bajt zastępujący w rodzimym zestawie znaków.

Na przykład wartość dziesiętna ?, jako bajt rodzimy, wynosi 63, jeśli rodzimy zestaw znaków jest oparty na kodzie ASCII, na przykład ISO-8859-1, a wartość 111, jeśli rodzimy zestaw znaków to EBCDIC.

**Uwaga:** Należy zauważyć, że jeśli obiekt MQMD lub MQMessage ma ustawione pola **unmappableAction** lub **unMappableReplacement** , to wartości tych pól mają pierwszeństwo przed właściwościami systemu Java . Umożliwia to nadpisanie wartości określonych przez właściwości systemowe Java dla każdego komunikatu, jeśli jest to wymagane.

**Pojęcia pokrewne**
“Konwersje łańcuchów znaków w programie IBM MQ classes for Java” na stronie 362
IBM MQ classes for Java CharsetEncoders i CharsetDecoders są używane bezpośrednio do konwersji łańcuchów znaków. Domyślne zachowanie konwersji łańcucha znaków można skonfigurować przy użyciu dwóch właściwości systemowych. Obsługę komunikatów, które zawierają znaki niemożliwe do odwzorowania, można skonfigurować za pomocą programu `com.ibm.mq.MQMD`.

## Pisanie aplikacji IBM MQ classes for JMS/Jakarta Messaging

Po krótkim wprowadzeniu do modelu JMS ta sekcja zawiera szczegółowe wskazówki dotyczące pisania aplikacji IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging .

### O tym zadaniu

V 9.3.0 JM 3.0 V 9.3.0 W produkcie IBM MQ 9.3.0produkt Jakarta Messaging 3.0 jest obsługiwany na potrzeby tworzenia nowych aplikacji. Produkt IBM MQ 9.3.0 nadal obsługuje produkt JMS 2.0 dla istniejących aplikacji. Nie jest obsługiwane używanie zarówno interfejsu API Jakarta Messaging 3.0 , jak i interfejsu API JMS 2.0 w tej samej aplikacji. Więcej informacji na ten temat zawiera sekcja Używanie klas IBM MQ dla przesyłania komunikatów JMS/Jakarta.

**Pojęcia pokrewne**
IBM MQ classes for Jakarta Messaging: przegląd

#### *Model JMS i Jakarta Messaging*

Model JMS i Jakarta Messaging definiują zestaw interfejsów, które mogą być używane przez aplikacje Java do wykonywania operacji przesyłania komunikatów. IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging są dostawcami przesyłania komunikatów. Definiują one sposób, w jaki obiekty JMS i Jakarta Messaging są powiązane z pojęciami IBM MQ . Specyfikacje JMS i Jakarta Messaging oczekują pewnych obiektów JMS i Jakarta Messaging , które będą administrowane.

JMS 2.0 Firma IBM MQ 8.0 dodała obsługę wersji JMS 2.0 standardu JMS , w której wprowadzono uproszczony interfejs API, zachowując jednocześnie klasyczny interfejs API z poziomu produktu JMS 1.1.

JM 3.0 V 9.3.0 V 9.3.0 W produkcie IBM MQ 9.3.0produkt Jakarta Messaging 3.0 jest obsługiwany na potrzeby tworzenia nowych aplikacji. Produkt IBM MQ 9.3.0 nadal obsługuje produkt JMS 2.0 dla istniejących aplikacji. Nie jest obsługiwane używanie zarówno interfejsu API JMS 2.0 , jak i interfejsu API Jakarta Messaging 3.0 w tej samej aplikacji.

**Uwaga:** W przypadku systemu Jakarta Messaging 3.0sterowanie specyfikacją JMS jest przenoszone z Oracle do Java Community Process. Jednak środowisko Oracle zachowuje kontrolę nad nazwą javax, która jest używana w innych technologiach Java , które nie zostały przeniesione do procesu społeczności

Java . Tak więc Jakarta Messaging 3.0 jest funkcjonalnie równoważne JMS 2.0 istnieją pewne różnice w nazewnictwie:

- Oficjalna nazwa wersji 3.0 to Jakarta Messaging , a nie Java Message Service.
- Nazwy pakietów i stałych są poprzedzone przedrostkiem `jakarta` , a nie `javax`. Na przykład w systemie JMS 2.0 początkowe połączenie z dostawcą przesyłania komunikatów to obiekt `javax.jms.Connection` , a w systemie Jakarta Messaging 3.0 jest to obiekt `jakarta.jms.Connection` .

JMS 2.0 Pakiety javax.jms definiują interfejsy JMS , a dostawca JMS implementuje te interfejsy dla konkretnego produktu przesyłania komunikatów. IBM MQ classes for JMS jest dostawcą JMS , który implementuje interfejsy JMS dla IBM MQ.

JM 3.0 Pakiety jakarta.jms definiują interfejsy języka Jakarta Messaging , a dostawca Jakarta Messaging implementuje te interfejsy dla konkretnego produktu przesyłania komunikatów. IBM MQ classes for Jakarta Messaging jest dostawcą Jakarta Messaging , który implementuje interfejsy Jakarta Messaging dla IBM MQ.

Ponieważ JMS i Jakarta Messaging współużytkują wiele wspólnych elementów, dalsze odwołania do JMS w tym temacie mogą być traktowane jako odwołania do obu tych elementów. Wszelkie różnice są podświetlane w razie potrzeby.

## Uproszczony interfejs API

Firma JMS 2.0 wprowadziła uproszczony interfejs API, zachowując jednocześnie specyficzne dla domeny i niezależne od domeny interfejsy produktu JMS 1.1. Uproszczony interfejs API zmniejsza liczbę obiektów potrzebnych do wysyłania i odbierania komunikatów i składa się z następujących interfejsów:

**ConnectionFactory**
ConnectionFactory to obiekt administrowany, który jest używany przez klienta JMS do tworzenia połączenia. Ten interfejs jest również używany w klasycznym interfejsie API.

**JMSKontekst**
Ten obiekt łączy obiekty połączenia i sesji klasycznego interfejsu API. JMSObiekty kontekstowe można tworzyć na podstawie innych obiektów JMSContext, przy czym połączenie bazowe jest duplikowane.

**JMSProducent**
Producent JMSjest tworzony przez kontekst JMSi jest używany do wysyłania komunikatów do kolejki lub tematu. Obiekt producenta JMSpowoduje utworzenie obiektów wymaganych do wysłania komunikatu.

**JMSKonsument**
Konsument JMSjest tworzony przez kontekst JMSi używany do odbierania komunikatów z tematu lub kolejki.

Uproszczony interfejs API ma wiele efektów:

- Obiekt kontekstu JMSzawsze automatycznie uruchamia połączenie bazowe.
- JMSProducenci i JMSKonsumenci mogą teraz pracować bezpośrednio z treścią komunikatu bez konieczności pobierania całego obiektu komunikatu przy użyciu metody getBody komunikatu.
- Właściwości komunikatu można ustawić w obiekcie producenta JMSprzy użyciu łańcucha metod przed wysłaniem treści komunikatu. Producent JMSbędzie obsługiwać tworzenie wszystkich obiektów, które są potrzebne do wysłania komunikatu. Za pomocą parametru JMS 2.0można ustawić właściwości i wysłać komunikat w następujący sposób:

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

W produkcie JMS 2.0 wprowadzono również subskrypcje współużytkowane, w których komunikaty mogą być współużytkowane przez wielu konsumentów. Wszystkie subskrypcje programu JMS 1.1 są traktowane jako subskrypcje niewspółużytkowane.

## Klasyczny interfejs API

Poniższa lista zawiera podsumowanie głównych interfejsów JMS klasycznego interfejsu API:

**Miejsce docelowe**
Miejsce docelowe to miejsce, w którym aplikacja wysyła komunikaty lub jest źródłem, z którego aplikacja odbiera komunikaty, lub oba te elementy.

**ConnectionFactory**
Obiekt ConnectionFactory hermetyzuje zestaw właściwości konfiguracyjnych połączenia. Aplikacja używa fabryki połączeń do utworzenia połączenia.

**Połączenie**
Obiekt połączenia hermetyzuje aktywne połączenie aplikacji z serwerem przesyłania komunikatów. Aplikacja używa połączenia do tworzenia sesji.

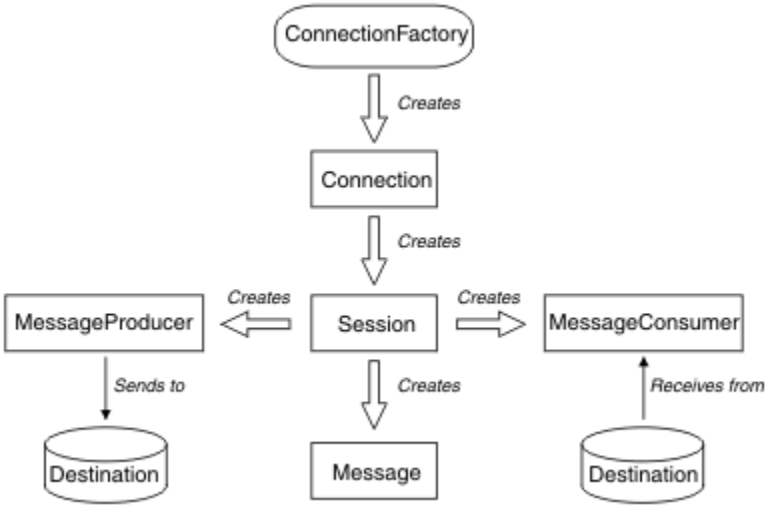
**Sesja**
Sesja jest jednowątkowym kontekstem do wysyłania i odbierania komunikatów. Aplikacja używa sesji do tworzenia komunikatów, producentów komunikatów i konsumentów komunikatów. Sesja jest transakcją lub nie jest transakcją.

**Komunikat**
Obiekt Message hermetyzuje komunikat wysyłany lub odbierany przez aplikację.

**MessageProducer**
Aplikacja używa producenta komunikatów do wysyłania komunikatów do miejsca docelowego.

**MessageConsumer**
Aplikacja używa konsumenta komunikatów do odbierania komunikatów wysyłanych do miejsca docelowego.

Rysunek 9 na stronie 147 przedstawia te obiekty i ich relacje.




*Rysunek 9. Obiekty JMS i ich relacje*

Obiekt docelowy, obiekt ConnectionFactorylub obiekt Connection mogą być używane współbieżnie przez różne wątki aplikacji wielowątkowej, ale obiekt Session, MessageProducerlub MessageConsumer nie może być używany współbieżnie przez różne wątki. Najprostszym sposobem zapewnienia, że obiekt Session, MessageProducerlub MessageConsumer nie jest używany współbieżnie, jest utworzenie osobnego obiektu Session dla każdego wątku.

## Domeny przesyłania komunikatów

Produkt JMS obsługuje dwa style przesyłania komunikatów:

- Przesyłanie komunikatów w modelu punkt-punkt
- Przesyłanie komunikatów w trybie publikowania/subskrypcji

Te style przesyłania komunikatów są również nazywane *domenami przesyłania komunikatów*i można łączyć oba style przesyłania komunikatów w aplikacji. W domenie typu punkt z punktem miejsce docelowe jest kolejką, a w domenie typu publikowanie/subskrypcja miejscem docelowym jest temat.

W wersjach JMS wcześniejszych niż JMS 1.1programowanie w domenie typu punkt z punktem używa jednego zestawu interfejsów i metod, a programowanie w domenie publikowania/subskrypcji używa innego zestawu. Dwa zestawy są podobne, ale oddzielne. W produkcie JMS 1.1można używać wspólnego zestawu interfejsów i metod, które obsługują obie domeny przesyłania komunikatów. Wspólne interfejsy udostępniają niezależny od domeny widok każdej domeny przesyłania komunikatów. Tabela 15 na stronie 148 zawiera listę niezależnych interfejsów domeny JMS i odpowiadających im interfejsów specyficznych dla domeny.

*Tabela 15. Interfejsy niezależne od domeny JMS i specyficzne dla domeny*

| **Interfejsy niezależne od domeny** | **Interfejsy specyficzne dla domeny typu punkt z punktem** | **Interfejsy specyficzne dla domeny publikowania/ subskrypcji** |
|---|---|---|
| ConnectionFactory | Fabryka QueueConnection | Fabryka TopicConnection |
| Połączenie | QueueConnection | TopicConnection |
| Miejsce docelowe | Kolejka | Temat |
| Sesja | QueueSession | TopicSession |
| MessageProducer | QueueSender | TopicPublisher |
| MessageConsumer | QueueReceiver<br>QueueBrowser | TopicSubscriber |

JMS 2.0 Produkt IBM MQ classes for JMS 2.0 obsługuje zarówno wcześniejsze interfejsy specyficzne dla domeny JMS 1.1 , jak i uproszczony interfejs API produktu JMS 2.0. Produkt IBM MQ classes for JMS 2.0 może być używany do obsługi istniejących aplikacji, w tym do tworzenia nowych funkcji w istniejących aplikacjach.

JM 3.0 Produkt IBM MQ classes for Jakarta Messaging 3.0 obsługuje wersje Jakarta Messaging tych samych interfejsów i jest zalecany do tworzenia nowych aplikacji.

W systemach IBM MQ classes for JMS i IBM MQ classes for Jakarta Messagingobiekty JMS są powiązane z pojęciami IBM MQ w następujący sposób:

- Obiekt połączenia ma właściwości, które pochodzą z właściwości fabryki połączeń użytej do utworzenia połączenia. Te właściwości sterują sposobem, w jaki aplikacja nawiązuje połączenie z menedżerem kolejek. Przykładami tych właściwości są nazwa menedżera kolejek oraz, w przypadku aplikacji nawiązującej połączenie z menedżerem kolejek w trybie klienta, nazwa hosta lub adres IP systemu, w którym działa menedżer kolejek.
- Obiekt Session hermetyzuje uchwyt połączenia IBM MQ , który definiuje zasięg transakcyjny sesji.
- Każdy z obiektów MessageProducer i MessageConsumer hermetyzuje uchwyt obiektu IBM MQ .

W przypadku korzystania z systemu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messagingmają zastosowanie wszystkie normalne reguły IBM MQ . W szczególności należy zauważyć, że aplikacja może wysłać komunikat do kolejki zdalnej, ale może odebrać komunikat tylko z kolejki, której właścicielem jest menedżer kolejek, z którym połączona jest aplikacja.

Specyfikacja JMS oczekuje, że obiekty ConnectionFactory i Destination będą administrowane. Administrator tworzy i obsługuje obiekty administrowane w centralnym repozytorium, a aplikacja JMS pobiera te obiekty za pomocą interfejsu JNDI ( Java Naming and Directory Interface).

W systemach IBM MQ classes for JMS i IBM MQ classes for Jakarta Messagingimplementacja interfejsu miejsca docelowego jest abstrakcyjną nadklasą klasy Queue i Topic, a więc instancją klasy Destination jest obiekt kolejki lub obiekt tematu. Niezależne od domeny interfejsy traktują kolejkę lub temat jako miejsce docelowe. Domena przesyłania komunikatów dla obiektu MessageProducer lub MessageConsumer jest określana na podstawie tego, czy miejsce docelowe jest kolejką, czy tematem.

Dlatego w systemach IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging można administrować obiektami następujących typów:

- ConnectionFactory
- Fabryka QueueConnection
- Fabryka TopicConnection
- Kolejka
- Temat
- XAConnectionFactory
- Fabryka XAQueueConnection
- Fabryka XATopicConnection

**Pojęcia pokrewne**

Interfejsy języka IBM MQ Java

“Tworzenie i konfigurowanie fabryk połączeń i miejsc docelowych” na stronie 211

Aplikacje IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging mogą tworzyć fabryki połączeń i miejsca docelowe, pobierając je jako obiekty administrowane z przestrzeni nazw JNDI ( Java Naming and Directory Interface), używając rozszerzeń IBM JMS lub rozszerzeń IBM MQ JMS . Aplikacja może również używać rozszerzeń IBM JMS lub rozszerzeń IBM MQ JMS do ustawiania właściwości fabryk połączeń i miejsc docelowych.

## *Komunikaty produktu JMS*

Komunikaty produktu JMS składają się z nagłówka, właściwości i treści. JMS definiuje pięć typów treści komunikatu.

Komunikaty programu JMS składają się z następujących części:

**Nagłówek**
Wszystkie komunikaty obsługują ten sam zestaw pól nagłówka. Pola nagłówka zawierają wartości, które są używane zarówno przez klientów, jak i dostawców do identyfikowania i kierowania komunikatów.

**Właściwości**
Każdy komunikat zawiera wbudowane narzędzie do obsługi wartości właściwości zdefiniowanych przez aplikację. Właściwości zapewniają wydajny mechanizm filtrowania komunikatów zdefiniowanych przez aplikację.

**Treść**
JMS definiuje pięć typów treści komunikatu, które obejmują większość obecnie używanych stylów przesyłania komunikatów:

**Strumień**
Strumień wartości podstawowych Java . Jest wypełniona i odczytywana sekwencyjnie.

**Odwzoruj**
Zestaw par nazwa-wartość, w których nazwy są łańcuchami, a wartości są typami podstawowymi Java . Dostęp do pozycji można uzyskać sekwencyjnie lub losowo według nazwy. Kolejność pozycji jest niezdefiniowana.

**Tekst**
Komunikat zawierający klasę java.lang.String.

**Obiekt**
Komunikat zawierający przekształcalny do postaci szeregowej obiekt Java .

**Bajty**
Strumień nieterpretowanych bajtów. Ten typ komunikatu jest przeznaczony do dosłownego kodowania treści w taki sposób, aby była zgodna z istniejącym formatem komunikatu.

Pole nagłówka JMSCorrelationID służy do łączenia jednego komunikatu z innym. Zwykle łączy on komunikat odpowiedzi z komunikatem żądającym. JMSCorrelationID może zawierać identyfikator komunikatu specyficzny dla dostawcy, łańcuch specyficzny dla aplikacji lub wartość typu byte [] rodzima dla dostawcy.

*Selektory komunikatów w produkcie JMS*
Komunikaty mogą zawierać wartości właściwości zdefiniowane przez aplikację. Aplikacja może używać selektorów komunikatów do filtrowania komunikatów przez dostawcę JMS .

Komunikat zawiera wbudowane narzędzie do obsługi wartości właściwości zdefiniowanych przez aplikację. W rezultacie udostępnia mechanizm dodawania do komunikatu pól nagłówka specyficznych dla aplikacji. Właściwości umożliwiają aplikacji korzystającej z selektorów komunikatów wybieranie lub filtrowanie komunikatów w jej imieniu przez dostawcę JMS przy użyciu kryteriów specyficznych dla aplikacji. Właściwości definiowane przez aplikację muszą być zgodne z następującymi regułami:

- Nazwy właściwości muszą być zgodnie z regułami dotyczącymi identyfikatora selektora komunikatów.
- Wartości właściwości mogą być typu Boolean, byte, short, int, long, float, double i string.
- Przedrostki nazw JMSX i JMS_ są zastrzeżone.

Wartości właściwości są ustawiane przed wysłaniem komunikatu. Gdy klient odbiera komunikat, właściwości komunikatu są dostępne tylko do odczytu. Jeśli klient próbuje w tym momencie ustawić właściwości, zgłaszany jest wyjątek MessageNotWriteableException . Jeśli zostanie wywołana komenda clearProperties , właściwości mogą być teraz zarówno odczytywane, jak i zapisywane.

Wartość właściwości może być duplikatem wartości w treści komunikatu. JMS nie definiuje strategii dla tego, co może być przekształcane w właściwość. Twórcy aplikacji muszą jednak mieć świadomość, że dostawcy JMS prawdopodobnie obsługują dane w treści komunikatu bardziej efektywnie niż dane we właściwościach komunikatu. Aby uzyskać najlepszą wydajność, aplikacje muszą używać właściwości komunikatu tylko wtedy, gdy muszą dostosować nagłówek komunikatu. Podstawową przyczyną takiej sytuacji jest obsługa niestandardowego wyboru komunikatów.

Selektor komunikatów JMS umożliwia klientowi określenie interesujących go komunikatów przy użyciu nagłówka komunikatu. Dostarczane są tylko komunikaty z nagłówkami zgodnymi z selektorem.

Selektory komunikatów nie mogą odwoływać się do wartości treści komunikatu.

Selektor komunikatu jest zgodny z komunikatem, gdy selektor przyjmuje wartość true, gdy pola nagłówka komunikatu i wartości właściwości są zastępowane odpowiednimi identyfikatorami w selektorze.

Selektor komunikatów jest łańcuchem, którego składnia jest oparta na podzbiorze składni wyrażenia warunkowego SQL92 . Selektor komunikatów jest wartościowany w kolejności od lewej do prawej w obrębie poziomu kolejności wykonywania. Aby zmienić tę kolejność, można użyć nawiasów. Predefiniowane literały selektorów i nazwy operatorów są tutaj pisane wielkimi literami, jednak nie jest w nich rozróżniana wielkość liter.

## Treść selektora komunikatów

Selektor komunikatów może zawierać:

- Literały
  - Literał łańcuchowy jest ujęty w cudzysłów. Podwójny cudzysłów reprezentuje cudzysłów. Przykłady to 'literal' i 'literal''s '. Podobnie jak literały łańcuchowe Java , używają one kodowania znaków Unicode.
  - Dokładny literał liczbowy jest wartością liczbową bez przecinka dziesiętnego, np. 57, -957 i +62. Obsługiwane są liczby z zakresu Java long.

  - Przybliżony literał liczbowy jest wartością liczbową w notacji naukowej, taką jak 7E3 lub -57.9E2, lub wartością liczbową z liczbą dziesiętną, taką jak 7., -95.7lub +6.2. Obsługiwane są liczby z zakresu Java double.
  - Literały boolowskie TRUE i FALSE.
- Identyfikatory:
  - Identyfikator to sekwencja o nieograniczonej długości, składająca się z Java liter i Java cyfr, z których pierwsza musi być literą Java . Litera jest dowolnym znakiem, dla którego metoda Character.isJavaLetter zwraca wartość true. Obejmuje to znaki _ i $. Litera lub cyfra to dowolny znak, dla którego metoda Character.isJavaLetterOrDigit zwraca wartość true.
  - Identyfikatory nie mogą być nazwami NULL, TRUE ani FALSE.
  - Identyfikatory nie mogą być identyfikatorami NOT, AND, OR, BETWEEN, LIKE, IN ani IS.
  - Identyfikatory są odwołaniami do pól nagłówków lub odwołaniami do właściwości.
  - W identyfikatorach rozróżniana jest wielkość liter.
  - Odwołania do pól nagłówka komunikatu są ograniczone do:
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
    - JMSType

    JMSMessageID, JMSTimrelationID, JMSCorrelationIDi JMSType mogą mieć wartość NULL, a jeśli tak, to są traktowane jako wartość NULL.
  - Każda nazwa rozpoczynająca się od łańcucha JMSX jest nazwą właściwości zdefiniowaną przez JMS.
  - Każda nazwa rozpoczynająca się od łańcucha JMS_ jest nazwą właściwości specyficzną dla dostawcy.
  - Każda nazwa, która nie rozpoczyna się od łańcucha JMS , jest nazwą właściwości specyficzną dla aplikacji. Jeśli istnieje odwołanie do właściwości, która nie istnieje w komunikacie, jej wartością jest NULL. Jeśli istnieje, jego wartością jest odpowiednia wartość właściwości.
- Biały znak jest taki sam, jak zdefiniowany dla Java: spacja, tabulacja pozioma, znak nowego formularza i znak końca wiersza.
- Wyrażenia:
  - Selektor jest wyrażeniem warunkowym. Selektor, który przyjmuje wartość true, jest zgodny. Selektor, który przyjmuje wartość false lub unknown, nie jest zgodny.
  - Wyrażenia arytmetyczne składają się z samych siebie, operacji arytmetycznych, identyfikatorów (z wartością, która jest traktowana jako literał liczbowy) i literałów liczbowych.
  - Wyrażenia warunkowe składają się z samych siebie, operacji porównania i operacji logicznych.
- Obsługiwana jest standardowa funkcja bracketing () do ustawiania kolejności, w jakiej wyrażenia są wartościowane.
- Operatory logiczne w kolejności wykonywania operacji: NOT, AND, OR.
- Operatory porównania: =, >, > =, <, < =, < > (różne).
  - Można porównywać tylko wartości tego samego typu. Jednym z wyjątków jest to, że poprawne jest porównanie dokładnych wartości liczbowych i przybliżonych wartości liczbowych. (Wymagana konwersja typów jest zdefiniowana przez reguły promocji numerycznej Java ). Jeśli podejmowana jest próba porównania różnych typów, selektor zawsze ma wartość false.
  - Porównywanie łańcuchów i wartości boolowskich jest ograniczone do znaków = i < >. Dwa łańcuchy są równe tylko wtedy, gdy zawierają taką samą sekwencję znaków.
- Operatory arytmetyczne w kolejności wykonywania operacji:
  - +,-jednoargumentowy.

  - *,/, mnożenie i dzielenie.
  - +,-, dodawanie i odejmowanie.
  - Operacje arytmetyczne na wartości NULL nie są obsługiwane. Jeśli zostaną podjęte próby, pełny selektor zawsze będzie mieć wartość false.
  - W operacjach arytmetycznych musi być używana promocja liczbowa Java .
- Operator porównania arithmetic-expr1 [ NOT] BETWEEN arithmetic-expr2 i arithmetic-expr3 :
  - Wiek POMIĘDZY 15 a 19 rokiem życia jest równoważny wiekowi > = 15 I wiekowi < = 19.
  - Wiek nie pomiędzy 15 a 19 rokiem życia jest odpowiednikiem wieku < 15 LUB wieku > 19 lat.
  - Jeśli dowolne wyrażenie operacji BETWEEN ma wartość NULL, wartością operacji jest false. Jeśli dowolne wyrażenie operacji NOT BETWEEN ma wartość NULL, wartością tej operacji jest true.
- identyfikator [ NOT] IN (string-literal1, string-literal2, ...) operator porównania, w którym identyfikator ma wartość typu String lub NULL.
  - Kraj IN ("UK", "US", "France") jest prawdziwy dla "UK" i fałszywy dla "Peru". Jest to odpowiednik wyrażenia (Country = 'UK ') LUB (Country = 'US') LUB (Country = 'France ').
  - Kraj NOT IN ("UK", "US", "France") jest fałszywy dla "UK" i prawdziwy dla "Peru". Jest to odpowiednik wyrażenia NOT ((Country = 'UK ') OR (Country = 'US') OR (Country = 'France ')).
  - Jeśli identyfikator operacji IN lub NOT IN ma wartość NULL, wartość operacji jest nieznana.
- identyfikator [ NOT] LIKE wartość_wzorca [ ESCAPE znak_zmiany znaczenia] operator porównania, w którym identyfikator ma wartość łańcuchową. wartość wzorca jest literałem łańcuchowym, gdzie _ oznacza dowolny pojedynczy znak, a% oznacza dowolną sekwencję znaków (łącznie z pustą sekwencją). Wszystkie inne znaki są dla siebie. Opcjonalny znak zmiany znaczenia jest pojedynczym znakowym literałem łańcuchowym, którego znak jest używany do zmiany znaczenia specjalnego znaczenia znaków _ i% w wartości wzorca.
  - phone LIKE '12%3' is true for 123 and 12993 and false for 1234.
  - słowo LIKE 'l_se' ma wartość true w przypadku "lose" i wartość false w przypadku "loose".
  - podkreślenie LIKE '_ %' ESCAPE' \ ' ma wartość true dla "_foo" i false dla "bar".
  - phone NOT LIKE '12%3' is false for 123 and 12993 and true for 1234.
  - Jeśli identyfikator operacji LIKE lub NOT LIKE ma wartość NULL, wartość operacji jest nieznana.
- Operator porównania identyfikatora IS NULL sprawdza wartość pola nagłówka pustego lub brakującą wartość właściwości.
  - prop_name IS NULL.
- Operator porównania identyfikatora IS NOT NULL sprawdza, czy istnieje wartość pola nagłówka inna niż NULL lub wartość właściwości.
  - prop_name ma wartość inną niż NULL.

## Przykład selektora komunikatów

Następujący selektor komunikatów wybiera komunikaty o typie komunikatu samochód, kolorze niebieskim i wadze większej niż 2500 funtów:

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

## Wartości właściwości NULL

Jak wspomniano na poprzedniej liście, wartości właściwości mogą mieć wartość NULL. Wartościowanie wyrażeń selektorów zawierających wartości NULL jest definiowane przez semantykę NULL języka SQL 92. Poniższa lista zawiera krótki opis tej semantyki:

- SQL traktuje wartość NULL jako nieznaną.

- Porównanie lub arytmetyka z nieznaną wartością zawsze zwraca nieznaną wartość.
- Operator IS NULL przekształca nieznaną wartość w wartość TRUE.
- Operator IS NOT NULL przekształca nieznaną wartość w wartość FALSE.

## Specjalne zachowanie JMSMessageID i JMSCorrelationID

Klasy IBM MQ dla JMS zawierają optymalizacje podczas wybierania komunikatów z kolejki na podstawie atrybutu JMSMessageID lub JMSCorrelationID.

Jeśli aplikacja określa selektor formularza:

JMSMessageID= 'ID:*id_komunikatu*'

gdzie *id_komunikatu* jest łańcuchem zawierającym standardowy identyfikator komunikatu IBM MQ , klasy IBM MQ dla usługi JMS używają identyfikatora **MatchOption** MQMO_MATCH_MSG_ID do pobrania komunikatu z określonym identyfikatorem komunikatu.

Aby na przykład pobrać z kolejki komunikat o identyfikatorze 414D51207061756C745639314C545320C57C1A5F25ECE602 , aplikacja powinna użyć następującego selektora komunikatów:

```
JMSMessageID='ID:414D51207061756C745639314C545320C57C1A5F25ECE602'
```

Podobnie, jeśli aplikacja określa selektor, który ma format:

JMSCorrelationID = 'ID:*id_korelacji*'

gdzie *correlation_id* jest łańcuchem zawierającym standardowy identyfikator korelacji IBM MQ , klasy IBM MQ dla usługi JMS używają identyfikatora **MatchOption** MQMO_MATCH_CORREL_ID do pobrania komunikatu o podanym identyfikatorze korelacji z kolejki.

W poniższym przykładzie selektor komunikatów jest używany do pobrania komunikatu, który ma identyfikator korelacji 414D51207061756C745639314C545320846E5B5F25B1CC02:

```
JMSCorrelationID='ID:414D51207061756C745639314C545320846E5B5F25B1CC02'
```

Jeśli selektor komunikatów zawiera wartość zerową dla atrybutu *message_id* lub *correlation_id*, to jest on zgodny z dowolnym komunikatem w kolejce. Jeśli na przykład aplikacja używa selektora:

```
JMSMessageID='ID:000000000000000000000000000000000000000000000000'
```

wszystkie komunikaty w kolejce są traktowane jako zgodne i zwracane do aplikacji.

Więcej informacji na temat MQMO_MATCH_MSG_ID i MQMO_MATCH_CORREL_ID **MatchOptions**zawiera sekcja MatchOptions (MQLONG).

## Ograniczenia

Chociaż SQL obsługuje stałe porównania dziesiętne i arytmetyczne, selektory komunikatów JMS nie obsługują tego porównania. Dlatego dokładne literały liczbowe są ograniczone do tych, które nie mają liczby dziesiętnej. Jest to również powód, dla którego istnieją liczby z liczbą dziesiętną jako alternatywną reprezentację przybliżonej wartości liczbowej.

Komentarze SQL nie są obsługiwane.

*Odwzorowywanie komunikatów JMS na komunikaty IBM MQ*
Komunikaty produktu IBM MQ składają się z deskryptora komunikatu, opcjonalnego nagłówka MQRFH2 i treści. Treść komunikatu JMS jest częściowo odwzorowana, a częściowo kopiowana do komunikatu IBM MQ .

W tym temacie opisano, w jaki sposób struktura komunikatu JMS opisana w pierwszej części tej sekcji jest odwzorowywana na komunikat IBM MQ . Jest to interesujące dla programistów, którzy chcą przesyłać komunikaty między aplikacjami JMS i tradycyjnymi IBM MQ . Dotyczy to również osób, które chcą

manipulować komunikatami przesyłanymi między dwiema aplikacjami JMS , na przykład w implementacji IBM Integration Bus .

Ta sekcja nie ma zastosowania, jeśli aplikacja używa połączenia w czasie rzeczywistym z brokerem. Gdy aplikacja korzysta z połączenia w czasie rzeczywistym, cała komunikacja jest realizowana bezpośrednio za pośrednictwem protokołu TCP/IP; nie są używane żadne kolejki ani komunikaty systemu IBM MQ .

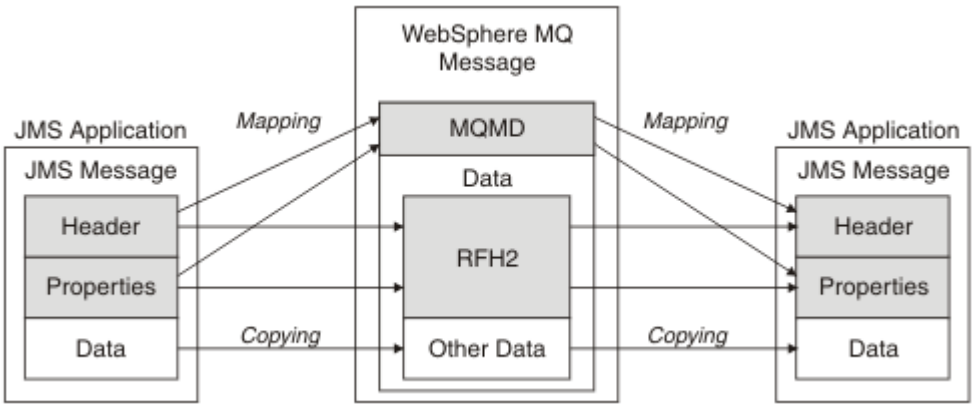
Komunikaty IBM MQ składają się z trzech komponentów:

- Deskryptor komunikatu IBM MQ (MQMD)
- Nagłówek IBM MQ MQRFH2
- Treść komunikatu.

Nagłówek MQRFH2 jest opcjonalny, a jego uwzględnienie w komunikacie wychodzącym jest zarządzane przez opcję TARGCLIENT w klasie docelowej JMS . Opcję tę można ustawić za pomocą narzędzia administracyjnego IBM MQ JMS . Ponieważ nagłówek MQRFH2 zawiera informacje specyficzne dla produktu JMS, należy zawsze dołączać je do komunikatu, gdy nadawca wie, że miejsce docelowe odbierania jest aplikacją JMS . Zwykle należy pominąć MQRFH2 podczas wysyłania komunikatu bezpośrednio do aplikacji innej niżJMS . Wynika to z faktu, że taka aplikacja nie oczekuje nagłówka MQRFH2 w swoim komunikacie IBM MQ .

Jeśli komunikat przychodzący nie ma nagłówka MQRFH2 , obiekt kolejki lub tematu pochodzący z pola nagłówka JMSReplyTo komunikatu domyślnie ma ustawioną tę opcję, aby komunikat odpowiedzi wysłany do kolejki lub tematu nie miał również nagłówka MQRFH2 . To zachowanie można wyłączyć, umieszczając nagłówek MQRFH2 w komunikacie odpowiedzi tylko wtedy, gdy oryginalny komunikat ma nagłówek MQRFH2 . W tym celu należy ustawić właściwość TARGCLIENTMATCHING fabryki połączeń na wartość NO.

Rysunek 10 na stronie 154 pokazuje, w jaki sposób struktura komunikatu JMS jest transformowana w komunikat IBM MQ i z powrotem:




*Rysunek 10. Sposób transformowania komunikatów między produktem JMS i produktem IBM MQ przy użyciu nagłówka MQRFH2*

Struktury są przekształcane na dwa sposoby:

**Odwzorowanie**
Jeśli struktura MQMD zawiera pole, które jest równoważne polu JMS , pole JMS jest odwzorowywane na pole MQMD. Dodatkowe pola MQMD są prezentowane jako właściwości JMS , ponieważ aplikacja JMS może wymagać pobrania lub ustawienia tych pól podczas komunikacji z aplikacją inną niżJMS .

**Kopiowanie**
Jeśli nie ma odpowiednika MQMD, jako pole wewnątrz MQRFH2jest przekazywane pole nagłówka JMS lub właściwość, która może być transformowana.

*Nagłówek MQRFH2 i JMS*
Ta kolekcja tematów zawiera opis nagłówka MQRFH 2, który zawiera dane specyficzne dla produktu JMSpowiązane z treścią komunikatu. Nagłówek MQRFH w wersji 2 jest rozszerzalny i może również zawierać dodatkowe informacje, które nie są bezpośrednio powiązane z produktem JMS. Jednak ta sekcja

dotyczy tylko jej użycia przez produkt JMS.Pełny opis znajduje się w sekcji MQRFH2 -reguły i nagłówek formatowania 2.

Nagłówek składa się z dwóch części: części stałej i części zmiennej.

**Stała część**
Stała część jest modelowana na podstawie *standardowego* IBM MQ wzorca nagłówka i składa się z następujących pól:

**StrucId (MQCHAR4)**
Identyfikator struktury.

Musi mieć wartość MQRFH_STRUC_ID (wartość: "RFH " ) (wartość początkowa).

MQRFH_STRUC_ID_ARRAY (wartość: "R", "F", "H", " " ) jest również zdefiniowana.

**Wersja (MQLONG)**
Numer wersji struktury.

Musi mieć wartość MQRFH_VERSION_2 (wartość: 2) (wartość początkowa).

**StrucLength (MQLONG)**
Łączna długość nagłówka MQRFH2z uwzględnieniem pól danych NameValue.

Wartość ustawiona w polu StrucLength musi być wielokrotnością 4 (w celu osiągnięcia tego celu dane w polach danych NameValuemogą być dopełniane spacjami).

**Kodowanie (MQLONG)**
Kodowanie danych.

Kodowanie dowolnych danych liczbowych w części komunikatu następującej po nagłówku MQRFH2 (następny nagłówek lub dane komunikatu następujące po tym nagłówku).

**CodedCharSetId (MQLONG)**
Identyfikator kodowanego zestawu znaków.

Reprezentacja dowolnych danych znakowych w części komunikatu następującej po nagłówku MQRFH2 (następny nagłówek lub dane komunikatu następujące po tym nagłówku).

**Format (MQCHAR8)**
Nazwa formatu.

Nazwa formatu części komunikatu następującej po MQRFH2.

**Flagi (MQLONG)**
Flagi.

MQRFH_NO_FLAGS = 0. Nie ustawiono flag.

**Identyfikator CCSID NameValue(MQLONG)**
Identyfikator kodowanego zestawu znaków (CCSID) dla łańcuchów znaków danych NameValuezawartych w tym nagłówku. Dane NameValuemogą być kodowane w zestawie znaków, który różni się od innych łańcuchów znaków zawartych w nagłówku (StrucID i Format).

Jeśli identyfikator CCSID NameValuejest 2-bajtowym identyfikatorem CCSID Unicode (1200, 13488 lub 17584), kolejność bajtów w kodzie Unicode jest taka sama, jak kolejność bajtów w polach liczbowych w MQRFH2. (Na przykład wersja, identyfikator CCSID StrucLengthi NameValue).

*Tabela 16. Możliwe wartości pola identyfikatora CCSID NameValue*

| **CCSID** | **Znaczenie** |
|---|---|
| 1200 | UTF-16, najnowsza obsługiwana wersja Unicode |
| 13488 | UTF-16, podzbiór Unicode w wersji 2.0 |
| 17584 | Podzbiór UTF-16, wersja Unicode 3.0 (zawiera symbol euro) |

| *Tabela 16. Możliwe wartości pola identyfikatora CCSID NameValue (kontynuacja)* | |
|---|---|
| **CCSID** | **Znaczenie** |
| 1208 | UTF-8, najnowsza obsługiwana wersja Unicode |

**Część zmiennej**

Zmienna część następuje po stałej części. Część zmiennej zawiera liczbę folderów MQRFH2 . Każdy folder zawiera zmienną liczbę elementów lub właściwości. Właściwości powiązane z grupą folderów. Nagłówki MQRFH2 utworzone przez JMS mogą zawierać dowolny z następujących folderów:

**Folder `mcd`**

`mcd` zawiera właściwości opisujące format komunikatu. Na przykład właściwość domeny usługi komunikatu `Msd` identyfikuje komunikat JMS jako `JMSTextMessage`, `JMSBytesMessage`, `JMSStreamMessage`, `JMSMapMessage`, `JMSObjectMessage` lub wartość NULL.

Folder `mcd` jest zawsze obecny w komunikacie usługi Java Message Service zawierającym `MQRFH2`.

Jest on zawsze obecny w komunikacie zawierającym element `MQRFH2` wysłanym z programu IBM Integration Bus. Opisuje on domenę, format, typ i zestaw komunikatu.

| *Tabela 17. `mcd` - nazwa właściwości, synonim, typ danych i folder* | | | |
|---|---|---|---|
| **Synonim właściwości** | **Nazwa właściwości** | **Typ danych** | **Folder** |
| | `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>` |

Nie należy dodawać własnych właściwości w folderze `mcd`.

**Folder `jms`**

`jms` zawiera pola nagłówka JMS oraz właściwości JMSX, które nie mogą być w pełni wyrażone w produkcie `MQMD`. Folder `jms` jest zawsze obecny w produkcie `MQRFH2` usługi Java Message Service.

**Folder `usr`**

`usr` zawiera zdefiniowane przez aplikację właściwości JMS powiązane z komunikatem. Folder `usr` jest obecny tylko wtedy, gdy w aplikacji ustawiono właściwość definiowaną przez aplikację.

**Folder `mqext`**

Produkt `mqext` zawiera następujące typy właściwości:

- Właściwości, które są używane tylko przez produkt WebSphere Application Server.
- Właściwości związane z opóźnionym dostarczaniem komunikatów.

Folder jest obecny, jeśli w przypadku aplikacji ustawiono co najmniej jedną właściwość zdefiniowaną przez IBM albo używane jest opóźnienie dostawy.

| *Tabela 18. mqext - nazwa właściwości, synonim, typ danych i folder* | | | |
|---|---|---|---|
| **Synonim właściwości** | **Nazwa właściwości** | **Typ danych** | **Folder** |
| 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> |

Nie należy dodawać własnych właściwości w folderze mqext.

**Folder mqps**

mqps zawiera właściwości, które są używane tylko przez proces publikowania/subskrybowania produktu IBM MQ. Folder jest obecny tylko wtedy, gdy w przypadku aplikacji ustawiono co najmniej jedną zintegrowaną właściwość publikowania/subskrybowania.

| *Tabela 19. mqps - nazwa właściwości, synonim, typ danych i folder* | | | |
|---|---|---|---|
| **Synonim właściwości** | **Nazwa właściwości** | **Typ danych** | **Folder** |
| 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> |

Nie należy dodawać własnych właściwości w folderze mqps.

Tabela 20 na stronie 157 przedstawia pełną listę nazw właściwości.

| *Tabela 20. Foldery i właściwości MQRFH2 używane przez produkt JMS* | | | | |
|---|---|---|---|---|
| **JMS nazwa pola** | **Java typ** | **Nazwa folderu MQRFH2** | **Nazwa właściwości** | **Typ/wartości** |
| JMSDestination | Miejsce docelowe | jms | Dst. | string (łańcuch) |

| *Tabela 20. Foldery i właściwości MQRFH2 używane przez produkt JMS (kontynuacja)* | | | | |
|---|---|---|---|---|
| **JMS nazwa pola** | **Java typ** | **Nazwa folderu MQRFH2** | **Nazwa właściwości** | **Typ/wartości** |
| JMSExpiration | long | jms | Wyg. | i8 |
| JMSPriority | int | jms | PRI | i4 |
| JMSDeliveryMode | int | jms | Dłw | i4 |
| JMSCorrelationID | Łańcuch | jms | CID | string (łańcuch) |
| JMSReplyTo | Miejsce docelowe | jms | Rto (do) | string (łańcuch) |
| JMSTimestamp | long | jms | tms | i8 |
| JMSType | Łańcuch | MCD | Typ, Zestaw, Fmt | string (łańcuch) |
| JMSXGroupID | Łańcuch | jms | Identyfikator grupy | string (łańcuch) |
| JMSXGroupSeq | int | jms | Sekw | i4 |
| xxx (zdefiniowane przez użytkownika) | Dowolna | USR | xxx | dowolne |
| | | MCD | MSD | jms_none, tekst_jms jms_bytes, mapa jms Strumień jms Obiekt jms |

**NameValueDługość (MQLONG)**
Długość w bajtach łańcucha danych NameValue, który znajduje się bezpośrednio po tym polu (nie zawiera własnej długości).

**Dane NameValue(MQCHARn)**
Pojedynczy łańcuch znaków, którego długość w bajtach jest określona przez poprzedzające pole długości NameValue. Zawiera on folder zawierający sekwencję właściwości. Każda właściwość to tercet nazwa/typ/wartość, zawarty w elemencie XML, którego nazwa jest nazwą folderu, w następujący sposób:

```
<foldername>
triplet1 triplet2 .....  tripletn </foldername>
```

Po zamykającym znaczniku `</foldername>` mogą występować spacje jako znaki dopełniające. Każdy tercet jest zakodowany przy użyciu składni podobnej do XML:

```
<name dt='datatype'>value</name>
```

Element `dt='datatype'` jest opcjonalny i jest pomijany w przypadku wielu właściwości, ponieważ typ danych jest predefiniowany. Jeśli znacznik `dt=` zawiera co najmniej jeden znak spacji, należy go umieścić przed nim.

**name**
to nazwa właściwości; patrz Tabela 20 na stronie 157.

**datatype**
musi być zgodny, po zawijaniu, z jednym z typów danych wymienionych w sekcji Tabela 21 na stronie 159.

**value**
jest łańcuchową reprezentacją wartości do przekazania, przy użyciu definicji w pliku Tabela 21 na stronie 159.

Wartość NULL jest kodowana przy użyciu następującej składni:

```
<name dt='datatype' xsi:nil='true'></name>
```

Nie należy używać opcji `xsi:nil='false'`.

*Tabela 21. Typy danych właściwości*

| **Typ danych** | **Definicja** |
|---|---|
| string (łańcuch) | Dowolna sekwencja znaków z wyjątkiem < i & |
| boolean (boolowskie) | Znak `0` lub `1` ( `0` = false, `1` = true) |
| bin.hex | Cyfry szesnastkowe reprezentujące oktety |
| i1 | Liczba wyrażona za pomocą cyfr `0..9`, z opcjonalnym znakiem (bez ułamków lub wykładnika). Musi mieścić się w zakresie od -128 do 127 włącznie |
| i2 | Liczba wyrażona za pomocą cyfr `0..9`, z opcjonalnym znakiem (bez ułamków lub wykładnika). Musi być z zakresu od -32768 do 32767 włącznie |
| i4 | Liczba wyrażona za pomocą cyfr `0..9`, z opcjonalnym znakiem (bez ułamków lub wykładnika). Musi należeć do zakresu od -2147483648 do 2147483647 włącznie |
| i8 | Liczba wyrażona za pomocą cyfr `0..9`, z opcjonalnym znakiem (bez ułamków lub wykładnika). Musi mieścić się w zakresie od -9223372036854775808 do 9223372036854775807 włącznie |
| int | Liczba wyrażona za pomocą cyfr `0..9`, z opcjonalnym znakiem (bez ułamków lub wykładnika). Musi należeć do tego samego zakresu co i8. Można go użyć zamiast jednego z typów i *, jeśli nadawca nie chce powiązać określonej precyzji z właściwością. |
| r4 | Liczba zmiennopozycyjna, wielkość < = 3.40282347E+38, > = 1.175E-37 wyrażona za pomocą cyfr `0..9`, opcjonalny znak, opcjonalne cyfry ułamkowe, opcjonalny wykładnik |
| r8 | Liczba zmiennopozycyjna, wielkość < = 1.7976931348623E+308, > = 2.225E-307 wyrażona za pomocą cyfr `0..9`, opcjonalny znak, opcjonalne cyfry ułamkowe, opcjonalny wykładnik |

Wartość łańcuchowa może zawierać spacje. W wartości łańcuchowej należy użyć następujących sekwencji o zmienionym znaczeniu:

- `&` dla znaku &
- `<` dla znaku <

Można użyć następujących sekwencji o zmienionym znaczeniu, ale nie są one wymagane:

- `>` dla znaku >
- `&apos;` dla znaku '
- `&quot;` dla znaku "

*Pola i właściwości JMS z odpowiednimi polami MQMD*
W poniższych tabelach przedstawiono pola MQMD równoważne polom nagłówka JMS , właściwościom JMS i właściwościom specyficznym dla dostawcy JMS .

Tabela 22 na stronie 160 zawiera listę pól nagłówka JMS , a Tabela 23 na stronie 160 zawiera listę właściwości JMS , które są odwzorowane bezpośrednio na pola MQMD. Tabela 24 na stronie 160 zawiera listę właściwości specyficznych dla dostawcy i pól MQMD, na które są one odwzorowane.

*Tabela 22. Odwzorowanie pól nagłówka JMS na pola MQMD*

| **JMS Pole nagłówka** | **Java typ** | **Pole MQMD** | **Typ C** |
|---|---|---|---|
| JMSDeliveryMode | int | Trwałość | MQLONG |
| JMSExpiration | long | Utrata ważności | MQLONG |
| JMSPriority | int | Priorytet | MQLONG |
| JMSMessageID | Łańcuch | MsgID | MQBYTE24 |
| JMSTimestamp | long | PutDate<br>PutTime | MQCHAR8<br>MQCHAR8 |
| JMSCorrelationID | Łańcuch | CorrelId | MQBYTE24 |

*Tabela 23. Odwzorowanie właściwości JMS na pola MQMD*

| **JMS właściwość** | **Java typ** | **Pole MQMD** | **Typ C** |
|---|---|---|---|
| JMSXUserID | Łańcuch | UserIdentifier | MQCHAR12 |
| JMSXAppID | Łańcuch | Nazwa_aplikacji_wstawiającej | MQCHAR28 |
| JMSXDeliveryCount | int | BackoutCount | MQLONG |
| JMSXGroupID | Łańcuch | GroupId | MQBYTE24 |
| JMSXGroupSeq | int | Numer_kolejny_komunikatu | MQLONG |

*Tabela 24. Odwzorowanie właściwości specyficznych dla dostawcy JMS na pola MQMD*

| **Właściwość specyficzna dla dostawcy JMS** | **Java typ** | **Pole MQMD** | **Typ C** |
|---|---|---|---|
| Wyjątek JMS_IBM_Report_Exception | int | Raport | MQLONG |
| Utrata ważności raportu JMS_IBM_Report_Expiration | int | Raport | MQLONG |
| JMS_IBM_Report_COA | int | Raport | MQLONG |
| JMS_IBM_Report_COD | int | Raport | MQLONG |
| Raport JMS_IBM_Report_PAN | int | Raport | MQLONG |
| JMS_IBM_Report_NAN | int | Raport | MQLONG |
| JMS_IBM_Report_Pass_Msg_ID | int | Raport | MQLONG |
| JMS_IBM_Report_Pass_Correl_ID (Identyfikator korelacji) | int | Raport | MQLONG |

| *Tabela 24. Odwzorowanie właściwości specyficznych dla dostawcy JMS na pola MQMD (kontynuacja)* | | | |
|---|---|---|---|
| **Właściwość specyficzna dla dostawcy JMS** | **Java typ** | **Pole MQMD** | **Typ C** |
| JMS_IBM_Report_Discard_Msg, | int | Raport | MQLONG |
| JMS_IBM_MsgType | int | MsgType | MQLONG |
| JMS_IBM_Feedback, | int | Opinie | MQLONG |
| JMS_IBM_Format | Łańcuch | Formatowanie "1" na stronie 161 | MQCHAR8 |
| Typ JMS_IBM_PutAppl | int | Typ_aplikacji_wstawiającej | MQLONG |
| JMS_IBM_Encoding | int | Kodowanie | MQLONG |
| Zestaw znaków JMS_IBM_Character_Set | Łańcuch | CodedCharacterSetId "2" na stronie 161 | MQLONG |
| JMS_IBM_PutDate | Łańcuch | PutDate | MQCHAR8 |
| JMS_IBM_PutTime | Łańcuch | PutTime | MQCHAR8 |
| JMS_IBM_Last_Msg_In_Group | boolean (boolowskie) | MsgFlags | MQLONG |

**Uwaga:**

1. JMS_IBM_Format reprezentuje format treści komunikatu. Można to zdefiniować przez aplikację, ustawiając właściwość JMS_IBM_Format komunikatu (należy zauważyć, że obowiązuje limit 8 znaków), lub można ustawić wartość domyślną dla formatu IBM MQ treści komunikatu odpowiedniego dla typu komunikatu JMS . JMS_IBM_Format jest odwzorowywany na pole formatu MQMD tylko wtedy, gdy komunikat nie zawiera sekcji RFH lub RFH2 . W typowym komunikacie jest on odwzorowywany na pole Format nagłówka RFH2 bezpośrednio poprzedzającego treść komunikatu.
2. Wartość właściwości JMS_IBM_Character_Set jest wartością łańcuchową, która zawiera Java odpowiednik zestawu znaków dla wartości liczbowej CodedCharacterSetId . Pole MQMD CodedCharacterSetId jest wartością liczbową, która zawiera odpowiednik łańcucha zestawu znaków Java określonego przez właściwość JMS_IBM_Character_Set.

*Odwzorowanie pól JMS na pola IBM MQ (komunikaty wychodzące)*
Te tabele przedstawiają sposób odwzorowania pól nagłówka i właściwości JMS na pola MQMD i MQRFH2 w czasie wysyłania () lub publikowania ().

Tabela 25 na stronie 162 pokazuje, w jaki sposób pola nagłówka JMS są odwzorowywane na pola MQMD/RFH2 w czasie wysyłania () lub publikowania (). Tabela 26 na stronie 162 pokazuje, w jaki sposób właściwości JMS są odwzorowywane na pola MQMD/RFH2 w czasie wysyłania () lub publikowania (). Tabela 27 na stronie 163 pokazuje, w jaki sposób właściwości specyficzne dla dostawcy JMS są odwzorowywane na pola MQMD w czasie wysyłania () lub publikowania (),

W przypadku pól oznaczonych jako Ustaw obiekt komunikatu wartość przesyłana jest wartością przechowaną w komunikacie JMS bezpośrednio przed operacją send () lub publish (). Wartość w komunikacie JMS pozostaje niezmieniona przez operację.

W przypadku pól oznaczonych jako Ustaw wg metody wysyłania wartość jest przypisywana podczas wykonywania metody send () lub publish () (dowolna wartość przechowywana w komunikacie JMS jest ignorowana). Wartość w komunikacie JMS zostanie zaktualizowana w celu wyświetlenia użytej wartości.

Pola oznaczone jako Tylko odbieranie nie są przesyłane i pozostają niezmienione w komunikacie przez funkcję send () lub publish ().

*Tabela 25. Odwzorowanie pola komunikatu wychodzącego*

| Nazwa pola nagłówka JMS | Pole MQMD używane do transmisji | Nagłówek | Ustawione przez |
|---|---|---|---|
| JMSDestination | | MQRFH2 | Metoda wysyłania |
| JMSDeliveryMode | Trwałość | MQRFH2 | Metoda wysyłania |
| JMSExpiration | Utrata ważności | MQRFH2 | Metoda wysyłania |
| JMSPriority | Priorytet | MQRFH2 | Metoda wysyłania |
| JMSMessageID | MsgID | | Metoda wysyłania |
| JMSTimestamp | PutDate/PutTime | | Metoda wysyłania |
| JMSCorrelationID | CorrelId | MQRFH2 | Obiekt komunikatu |
| JMSReplyTo | Menedżer kolejek ReplyToQ/ ReplyTo | MQRFH2 | Obiekt komunikatu |
| JMSType | | MQRFH2 | Obiekt komunikatu |
| JMSRedelivered | | | Tylko odbiór |

**Uwaga:**

1. Pole MQMD CodedCharacterSetId jest wartością liczbową, która zawiera odpowiednik łańcucha zestawu znaków Java określonego przez właściwość JMS_IBM_Character_Set.

*Tabela 26. Odwzorowanie właściwości JMS komunikatu wychodzącego*

| JMS Nazwa właściwości | Pole MQMD używane do transmisji | Nagłówek | Ustawione przez |
|---|---|---|---|
| JMSXUserID | UserIdentifier | | Metoda wysyłania |
| JMSXAppID | Nazwa_aplikacji_wstawiające j | | Metoda wysyłania |
| JMSXDeliveryCount | | | Tylko odbiór |
| JMSXGroupID | GroupId | MQRFH2 | Obiekt komunikatu |
| JMSXGroupSeq | Numer_kolejny_komunikatu | MQRFH2 | Obiekt komunikatu |

**Uwaga:**

Te właściwości są definiowane jako tylko do odczytu przez specyfikację JMS i są ustawiane (w niektórych przypadkach opcjonalnie) przez dostawcę JMS.

W produkcie IBM MQ classes for JMS dwie z tych właściwości mogą zostać przesłonięte przez aplikację. W tym celu upewnij się, że miejsce docelowe zostało odpowiednio skonfigurowane, ustawiając następujące właściwości:

1. Ustaw właściwość `WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT` na wartość `WMQConstants.WMQ_MDCTX_SET_ALL_CONTEXT`.
2. Ustaw właściwość `WMQConstants.WMQ_MQMD_WRITE_ENABLED` na wartość `true`.

Aplikacja może przesłonić następujące właściwości:

**JMSXAppID**
Tę właściwość można przesłonić, ustawiając właściwość `WMQConstants.JMS_IBM_MQMD_PUTAPPLNAME` w komunikacie-wartość powinna być łańcuchem Java .

**JMSXGroupID**
Tę właściwość można przesłonić, ustawiając właściwość `WMQConstants.JMS_IBM_MQMD_GROUPID` w komunikacie-wartość powinna być tablicą bajtów.

*Tabela 27. Odwzorowanie właściwości specyficzne dla dostawcy JMS komunikatu wychodzącego*

| Nazwa właściwości specyficznej dla dostawcy JMS | Pole MQMD używane do transmisji | Nagłówek | Ustawione przez |
|---|---|---|---|
| Wyjątek JMS_IBM_Report_Exception | Raport | | Obiekt komunikatu |
| Utrata ważności raportu JMS_IBM_Report_Expiration | Raport | | Obiekt komunikatu |
| JMS_IBM_Report_COA/COD | Raport | | Obiekt komunikatu |
| JMS_IBM_Report_NAN/PAN | Raport | | Obiekt komunikatu |
| JMS_IBM_Report_Pass_Msg_ID | Raport | | Obiekt komunikatu |
| JMS_IBM_Report_Pass_Correl_ID (Identyfikator korelacji) | Raport | | Obiekt komunikatu |
| JMS_IBM_Report_Discard_Msg, | Raport | | Obiekt komunikatu |
| JMS_IBM_MsgType | MsgType | | Obiekt komunikatu |
| JMS_IBM_Feedback, | Opinie | | Obiekt komunikatu |
| JMS_IBM_Format | Format | | Obiekt komunikatu |
| Typ JMS_IBM_PutAppl | Typ_aplikacji_wstawiającej | | Metoda wysyłania |
| JMS_IBM_Encoding | Kodowanie | | Obiekt komunikatu |
| Zestaw znaków JMS_IBM_Character_Set | CodedCharacterSetId | | Obiekt komunikatu |
| JMS_IBM_PutDate | PutDate | | Metoda wysyłania |

| *Tabela 27. Odwzorowanie właściwości specyficzne dla dostawcy JMS komunikatu wychodzącego (kontynuacja)* | | | |
|---|---|---|---|
| **Nazwa właściwości specyficznej dla dostawcy JMS** | **Pole MQMD używane do transmisji** | **Nagłówek** | **Ustawione przez** |
| JMS_IBM_PutTime | PutTime | | Metoda wysyłania |
| JMS_IBM_Last_Msg_In_Group | MsgFlags | | Obiekt komunikatu |

*Odwzorowywanie pól nagłówka JMS w funkcji send () lub publish ()*
Te uwagi dotyczą odwzorowania pól JMS w send () lub publish ().

**Miejsce JMSDestination- MQRFH2**
Jest on przechowywany w postaci łańcucha, który przekształca do postaci szeregowej właściwości obiektu docelowego, aby odbierający obiekt JMS mógł odtworzyć równoważny obiekt docelowy. Pole MQRFH2 jest zakodowane jako identyfikator URI (szczegółowe informacje na temat notacji identyfikatora URI zawiera sekcja "Identyfikatory URI (Uniform Resource Identifier)" na stronie 229 ).

**JMSReplyTo to MQMD.ReplyToQ, ReplyTo, MQRFH2**
Nazwa kolejki jest kopiowana do deskryptora MQMD produktu MQMD.ReplyToQ i nazwa menedżera kolejek są kopiowane do pól menedżera kolejek ReplyTo. Informacje o rozszerzeniu miejsca docelowego (inne przydatne szczegóły przechowywane w obiekcie docelowym) są kopiowane do pola MQRFH2 . Pole MQRFH2 jest zakodowane jako identyfikator URI (szczegółowe informacje na temat notacji identyfikatora URI zawiera sekcja "Identyfikatory URI (Uniform Resource Identifier)" na stronie 229 ).

**JMSDeliveryMode to MQMD.Persistence**
Wartość JMSDeliveryMode jest ustawiana przez metodę send () lub publish () albo MessageProducer, chyba że obiekt docelowy ją nadpisuje. Wartość JMSDeliveryMode jest odwzorowywana na deskryptor MQMD.Persistence trwałości w następujący sposób:

- Wartość JMS PERSISTENT jest równoważna wartości MQPER_PERSISTENT
- JMS wartość NON_PERSISTENT jest odpowiednikiem wartości MQPER_NOT_PERSISTENT

Jeśli właściwość trwałości MQQueue nie jest ustawiona na wartość WMQConstants.WMQ_PER_QDEF, wartość trybu dostarczania jest również kodowana w nagłówku MQRFH2.

**JMSExpiration do/z programu MQMD.Expiry, MQRFH2**
JMSExpiration przechowuje czas utraty ważności (suma bieżącego czasu i czasu życia), podczas gdy MQMD zapisuje czas życia. Ponadto atrybut JMSExpiration jest podany w milisekundach, ale ma wartość MQMD.Expiry jest w dziesiątych częściach sekundy.

- Jeśli metoda send () ustawia nieograniczony czas życia, MQMD.Expiry jest ustawiony na wartość MQEI_UNLIMITED, a w nagłówku MQRFH2nie jest zakodowany żaden atrybut JMSExpiration.
- Jeśli metoda send () ustawi czas życia krótszy niż 214748364.7 sekund (około 7 lat), czas życia jest przechowywany w programie MQMD.Expiryi czas utraty ważności (w milisekundach) są zakodowane jako wartość i8 w MQRFH2.
- Jeśli metoda send () ustawi czas życia dłuższy niż 214748364.7 sekund, MQMD.Expiry jest ustawiony na wartość MQEI_UNLIMITED. Prawdziwy czas utraty ważności w milisekundach jest zakodowany jako wartość i8 w MQRFH2.

**JMSPriority MQMD.Priority**
Bezpośrednio odwzoruj wartość JMSPriority (0-9) na wartość priorytetu MQMD (0-9). Jeśli atrybut JMSPriority ma wartość inną niż domyślna, poziom priorytetu jest również kodowany w nagłówku MQRFH2.

**JMSMessageID z MQMD.MessageID**
Wszystkie komunikaty wysłane z programu JMS mają unikalne identyfikatory przypisane przez program IBM MQ. Przypisana wartość jest zwracana w strukturze MQMD.MessageId po wywołaniu MQPUT i jest przekazywane z powrotem do aplikacji w polu JMSMessageID . IBM MQ messageId jest 24-bajtową wartością binarną, a JMSMessageID jest łańcuchem. Identyfikator JMSMessageID

składa się z binarnej wartości messageId przekształconej w sekwencję 48 znaków szesnastkowych z przedrostkiem w postaci identyfikatora znaków:. Produkt JMS udostępnia wskazówkę, którą można ustawić, aby wyłączyć tworzenie identyfikatorów komunikatów. Ta wskazówka jest ignorowana, a we wszystkich przypadkach przypisywany jest unikalny identyfikator. Każda wartość ustawiona w polu JMSMessageID przed nadpisaniem operacji send ().

Jeśli wymagana jest możliwość określenia deskryptora MQMD produktu MQMD.MessageID, można to zrobić za pomocą jednego z rozszerzeń produktu IBM MQ JMS opisanych w sekcji "Odczytywanie i zapisywanie deskryptora komunikatu z aplikacji IBM MQ classes for JMS" na stronie 251.

**JMSTim/To ( MQRFH2 )**
Podczas wysyłania pole JMSTimod_właściwości jest ustawiane zgodnie z zegarem maszyny JVM. Ta wartość jest ustawiana w nagłówku MQRFH2. Dowolna wartość ustawiona w polu JMSTim.właściwość przed nadpisaniem metody send (). Patrz także właściwości JMS_IBM_PutDate i JMS_IBM_PutTime .

**JMSType do MQRFH2**
Ten łańcuch jest ustawiany w polu MQRFH2 mcd.Type . Jeśli jest on w formacie identyfikatora URI, może mieć również wpływ na pola mcd.Set i mcd.Fmt .

**JMSCorrelationID do MQMD.CorrelId, MQRFH2**
JMSCorrelationID może zawierać jedną z następujących informacji:

**Identyfikator komunikatu specyficzny dla dostawcy**
Jest to identyfikator komunikatu z wcześniej wysłanego lub odebranego komunikatu, więc powinien to być łańcuch składający się z 48 małych cyfr szesnastkowych z przedrostkiem `ID:` . Przedrostek zostanie usunięty, pozostałe znaki zostaną przekształcone w znaki binarne, a następnie zostaną ustawione w deskryptorze MQMD MQMD.CorrelId .

**Wartość typu byte [] rodzima dla dostawcy**
Wartość jest kopiowana do deskryptora MQMD produktu MQMD.CorrelId dopełnione wartościami pustymi lub obcięte do 24 bajtów, jeśli jest to konieczne. Żadna wartość CorrelId nie jest zakodowana w nagłówku MQRFH2.

**Łańcuch specyficzny dla aplikacji**
Wartość jest kopiowana do MQRFH2. Pierwsze 24 bajty łańcucha, w formacie UTF8 , są zapisywane w strukturze MQMD.CorrelID.

*Odwzorowywanie pól właściwości JMS*
Te uwagi dotyczą odwzorowania pól właściwości JMS w komunikatach produktu IBM MQ .

**JMSXUserID z MQMD UserIdentifier**
Właściwość JMSXUserID jest ustawiana w przypadku powrotu z wywołania wysyłania.

**JMSXAppID z MQMD PutApplNazwa**
Wartość JSMXAppID jest ustawiana w przypadku powrotu z wywołania wysyłania.

**JMSXGroupID do MQRFH2 (punkt z punktem)**
W przypadku komunikatów typu punkt z punktem identyfikator JMSXGroupID jest kopiowany do pola GroupID MQMD. Jeśli nazwa JMSXGroupID rozpoczyna się od przedrostka ID:, jest ona przekształcana w wartość binarną. W przeciwnym razie jest on kodowany jako łańcuch UTF8 . Wartość jest dopełniana lub obcinana, jeśli jest to konieczne, do długości 24 bajtów. Flaga MQMF_MSG_IN_GROUP jest ustawiona.

**JMSXGroupID to MQRFH2 (publikowanie/subskrypcja)**
W przypadku komunikatów publikowania/subskrypcji identyfikator JMSXGroupID jest kopiowany do MQRFH2 jako łańcuch.

**JMSXGroupSeq MQMD MsgSeqNumer (punkt-punkt)**
W przypadku komunikatów typu punkt z punktem nazwa JMSXGroupSeq jest kopiowana do pola numeru MQMD MsgSeq. Flaga MQMF_MSG_IN_GROUP jest ustawiona.

**JMSXGroupSeq MQMD MsgSeqNumer (publikowania/subskrypcji)**
W przypadku komunikatów publikowania/subskrypcji kolejka JMSXGroupSeq jest kopiowana do struktury MQRFH2 jako i4.

*Odwzorowywanie pól specyficznych dla dostawcy JMS*
Poniższe uwagi dotyczą odwzorowania pól specyficznych dla dostawcy JMS na komunikaty IBM MQ .

**JMS_IBM_Report_*XXX* do raportu MQMD**
Aplikacja JMS może ustawić opcje raportu MQMD za pomocą następujących właściwości JMS_IBM_Report_*XXX* . Pojedyncza struktura MQMD jest odwzorowana na kilka właściwości JMS_IBM_Report_*XXX* .

Stałe JMS_IBM_Report_*XXX* znajdują się w pliku `com.ibm.msg.client.jakarta.wmq.WMQConstants` lub `com.ibm.msg.client.wmq.WMQConstants`.

**Wyjątek JMS_IBM_Report_Exception**

MQRO_EXCEPTION lub
MQRO_EXCEPTION_WITH_DATA lub
MQRO_EXCEPTION_WITH_FULL_DATA

**Utrata ważności raportu JMS_IBM_Report_Expiration**

MQRO_EXPIRATION lub
MQRO_EXPIRATION_WITH_DATA lub
MQRO_EXPIRATION_WITH_FULL_DATA

**JMS_IBM_Report_COA**

MQRO_COA lub
MQRO_COA_WITH_DATA lub
MQRO_KOA_WITH_FULL_DATA

**JMS_IBM_Report_COD**

MQRO_COD lub
MQRO_COD_WITH_DATA lub
MQRO_KOD_WITH_FULL_DATA

**Raport 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 (Identyfikator korelacji)**
MQRO_PASS_CORREL_ID (Identyfikator KORELACJI MQ)

**JMS_IBM_Report_Discard_Msg,**
MQRO_DISCARD_MSG

Wartości MQRO znajdują się w pliku `com.ibm.mq.constants.CMQC`.

**JMS_IBM_MsgType na MQMD MsgType**
Wartość jest odwzorowywana bezpośrednio na wartość MQMD MsgType. Jeśli aplikacja nie ustawiła jawnej wartości JMS_IBM_MsgType, zostanie użyta wartość domyślna. Ta wartość domyślna jest określana w następujący sposób:

- Jeśli właściwość JMSReplyTo jest ustawiona na miejsce docelowe kolejki produktu IBM MQ , parametr MSGType jest ustawiany na wartość MQMT_REQUEST.
- Jeśli właściwość JMSReplyTo nie jest ustawiona lub jest ustawiona na wartość inną niż miejsce docelowe kolejki produktu IBM MQ , opcja MsgType jest ustawiona na wartość MQMT_DATAGRAM.

**JMS_IBM_Feedback do MQMD Feedback**
Wartość jest odwzorowywana bezpośrednio na informację zwrotną MQMD.

**Format JMS_IBM_Format do formatu MQMD**
Wartość jest odwzorowywana bezpośrednio na format MQMD.

**Kodowanie JMS_IBM_Encoding do MQMD**
Jeśli ta właściwość jest ustawiona, nadpisuje kodowanie liczbowe kolejki docelowej lub tematu.

**JMS_IBM_Character_Set to MQMD CodedCharacterSetId**
Jeśli ta właściwość jest ustawiona, zastępuje ona właściwość kodowanego zestawu znaków kolejki docelowej lub tematu.

**JMS_IBM_PutDate z MQMD PutDate**
Wartość tej właściwości jest ustawiana podczas wysyłania bezpośrednio z pola PutDate w strukturze MQMD. Każda wartość ustawiona we właściwości JMS_IBM_PutDate przed nadpisaniem operacji wysyłania. To pole zawiera łańcuch składający się z ośmiu znaków, w formacie daty IBM MQ (RRRRMMDD). Ta właściwość może być używana z właściwością JMS_IBM_PutTime w celu określenia czasu umieszczenia komunikatu zgodnie z menedżerem kolejek.

**JMS_IBM_PutTime z MQMD PutTime**
Wartość tej właściwości jest ustawiana podczas wysyłania bezpośrednio z pola PutTime w strukturze MQMD. Dowolna wartość ustawiona we właściwości JMS_IBM_PutTime przed nadpisaniem operacji wysyłania. To pole zawiera łańcuch ośmiu znaków w formacie IBM MQ godziny GGMMSSTH.
Ta właściwość może być używana z właściwością JMS_IBM_PutDate w celu określenia czasu umieszczenia komunikatu zgodnie z menedżerem kolejek.

**JMS_IBM_Last_Msg_In_Group na MQMD MsgFlags**
W przypadku przesyłania komunikatów w trybie punkt z punktem ta wartość boolowska jest odwzorowywana na opcję MQMF_LAST_MSG_IN_GROUP w polu MsgFlags MQMD. Zwykle jest on używany z właściwościami JMSXGroupID i JMSXGroupSeq w celu wskazania starszej aplikacji IBM MQ , że ten komunikat jest ostatnim w grupie. Ta właściwość jest ignorowana w przypadku przesyłania komunikatów w trybie publikowania/subskrypcji.

*Odwzorowanie pól IBM MQ na pola JMS (komunikaty przychodzące)*
W poniższych tabelach przedstawiono sposób odwzorowania pól nagłówka i właściwości JMS na pola MQMD i MQRFH2 w czasie operacji get () lub receive ().

Tabela 28 na stronie 167 pokazuje, w jaki sposób pola nagłówka JMS są odwzorowywane na pola MQMD/MQRFH2 w czasie metody get () lub receive (). Tabela 29 na stronie 168 pokazuje, w jaki sposób pola właściwości JMS są odwzorowywane na pola MQMD/MQRFH2 w czasie metody get () lub receive (). Tabela 30 na stronie 168 pokazuje, w jaki sposób właściwości specyficzne dla dostawcy JMS są odwzorowywane.

*Tabela 28. Odwzorowanie pola nagłówka JMS komunikatu przychodzącego*

| **Nazwa pola nagłówka JMS** | **Pole MQMD pobrane z** | **Pole MQRFH2 pobrane z** |
|---|---|---|
| JMSDestination | | jms.Dst lub mqps.Top "1" na stronie 168 |
| JMSDeliveryMode | Trwałość "2" na stronie 168 | jms.Dlv "2" na stronie 168 |
| JMSExpiration | | jms.Exp |
| JMSPriority | Priorytet | |
| JMSMessageID | MsgID | |
| JMSTimestamp | PutDate "2" na stronie 168<br>PutTime "2" na stronie 168 | jms.Tms "2" na stronie 168 |
| JMSCorrelationID | CorrelId "2" na stronie 168 | jms.Cid "2" na stronie 168 |

| *Tabela 28. Odwzorowanie pola nagłówka JMS komunikatu przychodzącego (kontynuacja)* | | |
|---|---|---|
| **Nazwa pola nagłówka JMS** | **Pole MQMD pobrane z** | **Pole MQRFH2 pobrane z** |
| JMSReplyTo | Kolejka_zwrotna "2" na stronie 168<br>Menedżer_kolejek_zwrotnych "2" na stronie 168 | jms.Rto "2" na stronie 168 |
| JMSType | | mcd.Type, mcd.Set, mcd.Fmt |
| JMSRedelivered | BackoutCount | |

**Uwaga:**

1. Jeśli ustawione są obie opcje jms.Dst i mqps.Top , używana jest wartość z pola jms.Dst .
2. W przypadku właściwości, które mogą mieć wartości pobrane z nagłówka MQRFH2 lub deskryptora MQMD, jeśli są dostępne, używane jest ustawienie w nagłówku MQRFH2 .
3. Wartość właściwości JMS_IBM_Character_Set jest wartością łańcuchową, która zawiera Java odpowiednik zestawu znaków dla wartości liczbowej CodedCharacterSetId .

| *Tabela 29. Odwzorowanie właściwości komunikatu przychodzącego* | | |
|---|---|---|
| **JMS Nazwa właściwości** | **Pole MQMD pobrane z** | **Pole MQRFH2 pobrane z** |
| JMSXUserID | UserIdentifier | |
| JMSXAppID | Nazwa_aplikacji_wstawiającej | |
| JMSXDeliveryCount | BackoutCount | |
| JMSXGroupID | GroupId "1" na stronie 168 | jms.Gid "1" na stronie 168 |
| JMSXGroupSeq | Numer_kolejny_komunikatu "1" na stronie 168 | jms.Seq "1" na stronie 168 |

**Uwaga:**

1. W przypadku właściwości, które mogą mieć wartości pobrane z nagłówka MQRFH2 lub deskryptora MQMD, jeśli są dostępne, używane jest ustawienie w nagłówku MQRFH2 . Właściwości są ustawiane na podstawie wartości MQMD tylko wtedy, gdy ustawione są flagi komunikatu MQMF_MSG_IN_GROUP lub MQMF_LAST_MSG_IN_GROUP.

| *Tabela 30. Odwzorowanie właściwości JMS specyficzne dla dostawcy komunikatów przychodzących* | | |
|---|---|---|
| **JMS Nazwa właściwości** | **Pole MQMD pobrane z** | **Pole MQRFH2 pobrane z** |
| Wyjątek JMS_IBM_Report_Exception | Raport | |
| Utrata ważności raportu JMS_IBM_Report_Expiration | Raport | |
| JMS_IBM_Report_COA | Raport | |
| JMS_IBM_Report_COD | Raport | |
| Raport JMS_IBM_Report_PAN | Raport | |
| JMS_IBM_Report_NAN | Raport | |

*Tabela 30. Odwzorowanie właściwości JMS specyficzne dla dostawcy komunikatów przychodzących (kontynuacja)*

| **JMS Nazwa właściwości** | **Pole MQMD pobrane z** | **Pole MQRFH2 pobrane z** |
|---|---|---|
| JMS_IBM_Report_ Pass_Msg_ID (Identyfikator komunikatu komunikatu) | Raport | |
| JMS_IBM_Report_Pass_Correl_ID (Identyfikator korelacji) | Raport | |
| JMS_IBM_Report_Discard_Msg, | Raport | |
| JMS_IBM_MsgType | MsgType | |
| JMS_IBM_Feedback, | Opinie | |
| JMS_IBM_Format | Format | |
| Typ JMS_IBM_PutAppl | Typ_aplikacji_wstawiającej | |
| Kodowanie JMS_IBM_Encoding "1" na stronie 169 | Kodowanie | |
| Zestaw znaków JMS_IBM_Character_Set "1" na stronie 169 | CodedCharacterSetId | |
| JMS_IBM_PutDate | PutDate | |
| JMS_IBM_PutTime | PutTime | |
| JMS_IBM_Last_Msg_In_Group | MsgFlags | |

1. Ustawiana tylko wtedy, gdy komunikat przychodzący jest komunikatem bajtowym.

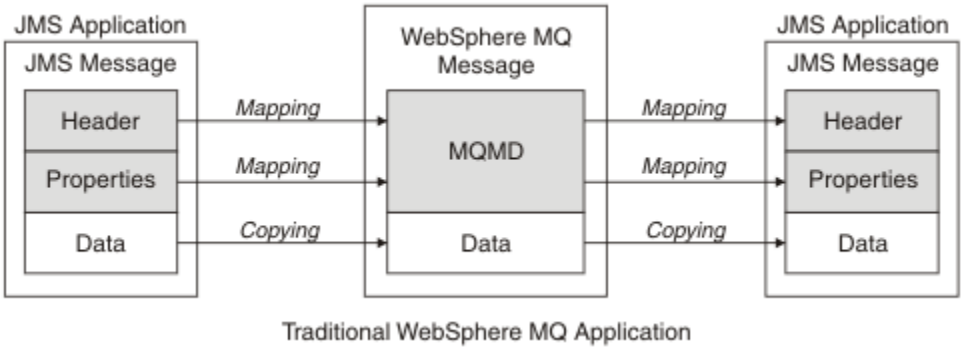
*Wymiana komunikatów między aplikacją JMS a tradycyjną aplikacją IBM MQ*
W tym temacie opisano, co się dzieje, gdy aplikacja JMS wymienia komunikaty z tradycyjną aplikacją IBM MQ , która nie może przetworzyć nagłówka MQRFH2 .

Rysunek 11 na stronie 170 przedstawia odwzorowanie.

Administrator wskazuje, że aplikacja JMS komunikuje się z tradycyjną aplikacją IBM MQ , ustawiając właściwość TARGCLIENT miejsca docelowego na wartość *MQ*. Oznacza to, że nie ma być generowany nagłówek MQRFH2 . W przeciwnym razie aplikacja odbierająca musi być w stanie obsłużyć nagłówek MQRFH2 .

Odwzorowanie z JMS na MQMD docelowe w tradycyjnej aplikacji IBM MQ jest takie samo jak odwzorowanie z JMS na MQMD docelowe w aplikacji JMS . Jeśli program IBM MQ classes for JMS odbierze komunikat IBM MQ z polem MQMD *Format* ustawionym na wartość inną niż MQFMT_RFH2, dane będą odbierane z aplikacji innej niżJMS . Jeśli formatem jest MQFMT_STRING, komunikat jest odbierany jako komunikat tekstowy JMS . W przeciwnym razie jest on odbierany jako komunikat o długości JMS bajtów. Ponieważ nie ma nagłówka MQRFH2, można odtworzyć tylko te właściwości JMS , które są przesyłane w deskryptorze MQMD.

Jeśli produkt IBM MQ classes for JMS odbiera komunikat, który nie ma nagłówka MQRFH2 , właściwość TARGCLIENT obiektu Queue lub Topic pochodzącego z pola nagłówka JMSReplyTo komunikatu jest domyślnie ustawiona na wartość MQ . Oznacza to, że komunikat odpowiedzi wysłany do kolejki lub tematu nie ma również nagłówka MQRFH2 . To zachowanie można wyłączyć, umieszczając nagłówek MQRFH2 w komunikacie odpowiedzi tylko wtedy, gdy oryginalny komunikat ma nagłówek MQRFH2 . W tym celu należy ustawić właściwość TARGCLIENTMATCHING fabryki połączeń na wartość NO.




*Rysunek 11. Sposób transformowania komunikatów produktu JMS w komunikaty produktu IBM MQ bez nagłówka MQRFH2*

*Treść komunikatu JMS*
Ten temat zawiera informacje o kodowaniu treści komunikatu. Kodowanie zależy od typu komunikatu JMS .

**ObjectMessage**
ObjectMessage to obiekt serializowany przez środowisko wykonawcze Java w normalny sposób.

**TextMessage**
TextMessage jest zakodowanym łańcuchem. W przypadku komunikatu wychodzącego łańcuch jest zakodowany w zestawie znaków podanym przez obiekt docelowy. Wartością domyślną jest kodowanie UTF8 (kodowanie UTF8 rozpoczyna się od pierwszego znaku komunikatu; na początku nie ma pola długości). Można jednak określić dowolny inny zestaw znaków obsługiwany przez IBM MQ classes for JMS. Takie zestawy znaków są używane głównie podczas wysyłania komunikatu do aplikacji innej niżJMS .

Jeśli zestaw znaków jest zestawem dwubajtowym (w tym UTF16), kolejność bajtów jest określana przez specyfikację kodowania obiektu docelowego na podstawie liczby całkowitej.

Komunikat przychodzący jest interpretowany przy użyciu zestawu znaków i kodowania, które są określone w samym komunikacie. Te specyfikacje znajdują się w ostatnim nagłówku IBM MQ (lub MQMD, jeśli nie ma nagłówków). W przypadku komunikatów produktu JMS ostatnim nagłówkiem jest zazwyczaj MQRFH2.

**BytesMessage**
Komunikat BytesMessage jest domyślnie sekwencją bajtów zdefiniowaną w specyfikacji JMS 1.0.2 i powiązanej dokumentacji Java .

W przypadku komunikatu wychodzącego złożonego przez samą aplikację właściwość kodowania obiektu docelowego może zostać użyta do przesłonięcia kodowania pól liczb całkowitych i zmiennopozycyjnych zawartych w komunikacie. Można na przykład zażądać, aby wartości zmiennopozycyjne były przechowywane w systemie S/390 , a nie w formacie IEEE).

Komunikat przychodzący jest interpretowany przy użyciu kodowania liczbowego określonego w komunikacie. Ta specyfikacja znajduje się w ostatnim nagłówku IBM MQ (lub MQMD, jeśli nie ma nagłówków). W przypadku komunikatów produktu JMS ostatnim nagłówkiem jest zazwyczaj MQRFH2.

Jeśli komunikat BytesMessage zostanie odebrany i ponownie wysłany bez modyfikacji, jego treść jest przesyłana jako bajt po odebraniu. Właściwość kodowania obiektu docelowego nie ma wpływu na treść. Jedyną jednostką typu łańcuchowego, którą można wysłać jawnie w BytesMessage , jest łańcuch UTF8 . Jest on zakodowany w formacie Java UTF8 i rozpoczyna się od pola o długości 2 bajtów. Właściwość zestawu znaków obiektu docelowego nie ma wpływu na kodowanie wychodzącego komunikatu BytesMessage. Wartość zestawu znaków w przychodzącym komunikacie IBM MQ nie ma wpływu na interpretację tego komunikatu jako JMS BytesMessage.

Jest mało prawdopodobne, aby aplikacje inne niżJava rozpoznawały kodowanie Java UTF8 . Dlatego, aby aplikacja JMS mogła wysłać komunikat BytesMessage zawierający dane tekstowe, sama aplikacja musi przekształcić swoje łańcuchy w tablice bajtów i zapisać te tablice w komunikacie BytesMessage.

**MapMessage**
MapMessage to łańcuch zawierający trójki nazwy/typu/wartości XML zakodowane jako:

```
<map>
  <elt name="elementname1" dt="datatype1">value1</elt>
  <elt name="elementname2" dt="datatype2">value2</elt>
  ...
</map>
```

gdzie `datatype` jest jednym z typów danych wymienionych w sekcji Tabela 21 na stronie 159. Domyślnym typem danych jest `string`, dlatego atrybut `dt="string"` jest pomijany dla elementów łańcuchowych.

Zestaw znaków używany do kodowania lub interpretowania łańcucha XML, który tworzy treść komunikatu odwzorowania, jest określany zgodnie z regułami, które mają zastosowanie do komunikatu tekstowego.

W wersjach systemu IBM MQ classes for JMS wcześniejszych niż 5.3 treść komunikatu odwzorowania była kodowana w następującym formacie:

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
  ...
</map>
```

System IBM MQ classes for JMS 5.3 lub nowszej może interpretować każdy z tych formatów, ale wersje systemu IBM MQ classes for JMS wcześniejsze niż 5.3 nie mogą interpretować bieżącego formatu.

Jeśli aplikacja musi wysłać komunikaty odwzorowania do innej aplikacji, która używa wersji systemu IBM MQ classes for JMS wcześniejszej niż 5.3, aplikacja wysyłająca musi wywołać metodę fabryki połączeń `setMapNameStyle(WMQConstants.WMQ_MAP_NAME_STYLE_COMPATIBLE)` w celu określenia, że komunikaty odwzorowania są wysyłane w poprzednim formacie. Domyślnie wszystkie komunikaty odwzorowania są wysyłane w bieżącym formacie.

**StreamMessage**
Komunikat StreamMessage jest podobny do komunikatu odwzorowania, ale bez nazw elementów:

```
<stream>
  <elt dt="datatype1">value1</elt>
  <elt dt="datatype2">value2</elt>
  ...
</stream>
```

gdzie `datatype` jest jednym z typów danych wymienionych w sekcji Tabela 21 na stronie 159. Domyślnym typem danych jest `string`, dlatego atrybut `dt="string"` jest pomijany dla elementów łańcuchowych.

Zestaw znaków używany do kodowania lub interpretowania łańcucha XML, który tworzy treść komunikatu StreamMessage , jest określany zgodnie z regułami dotyczącymi komunikatu TextMessage.

Pole MQRFH2.format jest ustawione w następujący sposób:

**MQFMT_BRAK**
dla ObjectMessage, BytesMessagelub wiadomości bez treści.

**MQFMT_STRING**
dla komunikatu TextMessage, StreamMessagelub MapMessage.

*JMS Konwersja komunikatu*
Konwersja danych komunikatu w produkcie JMS jest wykonywana podczas wysyłania i odbierania komunikatów. Program IBM MQ wykonuje większość konwersji danych automatycznie. Przekształca

on dane tekstowe i liczbowe podczas przesyłania komunikatu między aplikacjami JMS . Tekst jest przekształcany podczas wymiany JMSTextMessage między aplikacją JMS a aplikacją IBM MQ .

Jeśli planowana jest bardziej złożona wymiana komunikatów, interesujące są następujące tematy. Złożone wymiany komunikatów obejmują:

- Przesyłanie komunikatów innych niż tekstowe między aplikacją IBM MQ a aplikacją JMS .
- Wymiana danych tekstowych w formacie bajtowym.
- Przekształcanie tekstu w aplikacji.

## JMS Dane komunikatów

Konwersja danych jest niezbędna do wymiany danych tekstowych i liczbowych między aplikacjami, nawet między dwiema aplikacjami JMS . Wewnętrzna reprezentacja tekstu i liczb musi być zakodowana, aby można je było przesłać w komunikacie. Kodowanie wymusza decyzję o sposobie reprezentowania liczb i tekstu. IBM MQ zarządza kodowaniem tekstu i liczb w komunikatach JMS , z wyjątkiem JMSObjectMessage, patrz sekcja “JMSObjectMessage” na stronie 178. Używa on trzech atrybutów komunikatu. Te trzy atrybuty to CodedCharacterSetId, Encodingi Format.

Te trzy atrybuty komunikatu są zwykle przechowywane w polach nagłówka JMS ( MQRFH2) komunikatu JMS . Jeśli komunikat jest typu MQ, a nie JMS , atrybuty są zapisywane w deskryptorze komunikatu MQMD. Atrybuty są używane do przekształcania danych komunikatu JMS . Dane komunikatu produktu JMS są przesyłane w części danych komunikatu produktu IBM MQ .

## JMS Właściwości komunikatu

Właściwości komunikatu JMS , takie jak JMS_IBM_CHARACTER_SET, są wymieniane w części nagłówka MQRFH2 komunikatu JMS , chyba że komunikat został wysłany bez MQRFH2. Tylko wartości JMSTextMessage i JMSBytesMessage mogą być wysyłane bez wartości MQRFH2. Jeśli właściwość JMS jest przechowywana jako właściwość komunikatu IBM MQ w deskryptorze komunikatu MQMD, jest przekształcana w ramach konwersji języka MQMD . Jeśli właściwość JMS jest przechowywana w pliku MQRFH2, jest przechowywana w zestawie znaków określonym przez parametr MQRFH2.NameValueCCSID. Po wysłaniu lub odebraniu komunikatu właściwości komunikatu są przekształcane na ich wewnętrzną reprezentację w maszynie JVM i z niej. Konwersja odbywa się do i z zestawu znaków deskryptora komunikatu lub MQRFH2.NameValueCCSID. Dane liczbowe są przekształcane w tekst.

## JMS Konwersja komunikatu

Poniższe tematy zawierają przykłady i zadania, które są przydatne, jeśli planowana jest wymiana bardziej złożonych komunikatów wymagających konwersji.

*Podejścia do konwersji komunikatów JMS*
Wiele podejść do konwersji danych jest otwartych dla projektantów aplikacji JMS . Te podejścia nie są wyłączne; niektóre aplikacje mogą korzystać z ich kombinacji. Jeśli aplikacja wymienia tylko tekst lub komunikaty tylko z innymi aplikacjami JMS , zwykle nie jest rozważana konwersja danych. Konwersja danych jest wykonywana automatycznie przez IBM MQ.

Można zadać kilka pytań dotyczących sposobu podejścia do konwersji komunikatów:

**Czy trzeba w ogóle myśleć o nawróceniu wiadomości?**
W niektórych przypadkach, takich jak przesyłanie komunikatów JMS na JMS i wymiana komunikatów tekstowych z programami IBM MQ , program IBM MQ automatycznie wykonuje niezbędne konwersje. Można kontrolować konwersję danych ze względu na wydajność lub wymieniać złożone komunikaty, które mają predefiniowany format. W takich przypadkach należy zapoznać się z konwersją komunikatów i przeczytać poniższe tematy.

**Co to za nawrócenie?**
Istnieją cztery główne typy konwersji, które zostały wyjaśnione w następujących sekcjach:

1. “Konwersja danych klienta JMS” na stronie 173

2. “Konwersja danych aplikacji” na stronie 173
3. “Konwersja danych menedżera kolejek” na stronie 174
4. “Konwersja danych kanału komunikatów” na stronie 175

**Gdzie należy przeprowadzić konwersję?**

W sekcji “Wybór podejścia do konwersji komunikatów: odbiorca jest dobry” na stronie 175opisano zwykle sposób, w który "odbiorca jest dobry". "Dziennik ma dobre znaczenie" również w przypadku konwersji danych w programie JMS .

## Konwersja danych klienta JMS

JMS klient[1]Konwersja danych to konwersja operacji podstawowych i obiektów systemu Java na bajty w komunikacie JMS wysyłanym do miejsca docelowego oraz ponowna konwersja po odebraniu. Konwersja danych klienta JMS korzysta z metod klas `JMSMessage` . Metody są wymienione według typu klasy `JMSMessage` w pliku Tabela 31 na stronie 176.

Konwersja do i z wewnętrznej reprezentacji liczb i tekstu maszyny JVM jest wykonywana dla metod odczytu, pobierania, ustawiania i zapisu. Konwersja jest wykonywana, gdy komunikat jest wysyłany i gdy dowolna z metod read lub get jest wywoływana dla odebranego komunikatu.

Strona kodowa i kodowanie liczbowe używane do zapisywania lub ustawiania treści komunikatu są definiowane jako atrybuty miejsca docelowego. Docelową stronę kodową i kodowanie liczbowe można zmieniać administracyjnie. Aplikacja może również przesłonić stronę kodową miejsca docelowego i kodowanie, ustawiając właściwości komunikatu, które sterują zapisywaniem lub ustawianiem treści komunikatu.

Aby przekształcić kodowanie liczb, gdy komunikat `JMSBytesMessage` jest wysyłany do miejsca docelowego, które nie jest zdefiniowane jako kodowanie `Native` , należy przed wysłaniem komunikatu ustawić właściwość komunikatu `JMS_IBM_ENCODING` . Jeśli używany jest wzorzec "receiver sprawia, że jest dobry" lub jeśli komunikaty są wymieniane między aplikacjami JMS , aplikacja nie musi ustawiać wartości `JMS_IBM_ENCODING`. W większości przypadków właściwość `Encoding` może mieć wartość `Native`.

W przypadku komunikatów `JMSStreamMessage`, `JMSMapMessage`i `JMSTextMessage` używane są właściwości identyfikatora zestawu znaków miejsca docelowego. Kodowanie jest ignorowane podczas wysyłania, ponieważ liczby są zapisywane w formacie tekstowym. Aplikacja kliencka JMS nie musi ustawiać `JMS_IBM_CHARACTER_SET` przed wysłaniem komunikatu, jeśli ma zostać zastosowana właściwość docelowego zestawu znaków.

Aby pobrać dane z komunikatu, aplikacja wywołuje metody odczytu lub pobierania komunikatu JMS . Metody odwołują się do strony kodowej i kodowania zdefiniowanych w poprzednim nagłówku komunikatu, aby poprawnie utworzyć operacje podstawowe i obiekty Java .

Konwersja danych klienta JMS spełnia wymagania większości aplikacji JMS , które wymieniają komunikaty między klientami JMS . Nie jest wykonywana jawna konwersja danych. Nie jest używana klasa `java.nio.charset.Charset` , która jest zwykle używana podczas zapisywania tekstu do pliku. Metody `writeString` i `setString` przeprowadzają konwersję za użytkownika.

Więcej informacji na temat konwersji danych klienta JMS zawiera sekcja “Konwersja i kodowanie komunikatów klienta JMS” na stronie 185.

## Konwersja danych aplikacji

Aplikacja kliencka JMS może przeprowadzić jawną konwersję danych znakowych przy użyciu klasy `java.nio.charset.Charset` . Przykłady znajdują się w sekcji Rysunek 14 na stronie 178 i Rysunek 15 na stronie 178. Dane łańcuchowe są przekształcane w bajty przy użyciu metody `getBytes` i wysyłane jako bajty. Bajty są przekształcane z powrotem w tekst przy użyciu konstruktora `String` , który pobiera tablicę bajtów i `Charset`. Dane znakowe są konwertowane za pomocą metod `encode` i `decode`

[1] "JMS Klient" oznacza IBM MQ classes for JMS implementujące interfejs JMS , który działa w trybie klienta lub w trybie powiązań.

Charset . Zwykle komunikat jest wysyłany lub odbierany jako JMSBytesMessage, ponieważ część komunikatu JMSBytesMessage nie zawiera nic innego niż dane zapisane przez aplikację.[2]Można również wysyłać i odbierać bajty za pomocą JMSStreamMessage, JMSMapMessagelub JMSObjectMessage.

Nie istnieją metody języka Java służące do kodowania i dekodowania bajtów, które zawierają dane liczbowe reprezentowane w różnych formatach kodowania. Dane liczbowe są kodowane i dekodowane automatycznie przy użyciu metod odczytu i zapisu JMSMessage . Metody odczytu i zapisu używają wartości atrybutu JMS_IBM_ENCODING danych komunikatu.

Typowym zastosowaniem konwersji danych aplikacji jest wysyłanie lub odbieranie przez klienta JMS sformatowanego komunikatu z aplikacji innej niżJMS . Sformatowany komunikat zawiera dane tekstowe, liczbowe i bajtowe uporządkowane według długości pól danych. Jeśli aplikacja inna niżJMS nie określiła formatu komunikatu jako "MQSTR ", komunikat jest konstruowany jako JMSBytesMessage. Aby otrzymać sformatowane dane komunikatu w JMSBytesMessage , należy wywołać sekwencję metod. Metody muszą być wywoływane w tej samej kolejności, w jakiej pola zostały zapisane w komunikacie. Jeśli pola są numeryczne, należy znać kodowanie i długość danych liczbowych. Jeśli dowolne pole zawiera dane bajtowe lub tekstowe, należy znać długość wszystkich danych bajtowych w komunikacie. Istnieją dwa sposoby przekształcania sformatowanego komunikatu w obiekt Java , który jest łatwy w użyciu.

1. Utwórz klasę Java odpowiadającą rekordowi, aby hermetyzować odczyt i zapis komunikatu. Dostęp do danych w rekordzie odbywa się za pomocą metod get i set klasy.
2. Utwórz klasę Java odpowiadającą rekordowi, rozszerzając klasę com.ibm.mq.headers . Dostęp do danych w klasie jest uzyskiwany za pomocą obiektów korzystających z formularza, getStringValue(*fieldname*);

Patrz “Wymiana sformatowanego rekordu z aplikacją inną niżJMS” na stronie 193.

## Konwersja danych menedżera kolejek

Konwersja stron kodowych może być wykonywana przez menedżer kolejek, gdy program kliencki JMS otrzyma komunikat. Konwersja jest taka sama, jak konwersja wykonywana dla programu w języku C. Program w języku C ustawia MQGMO_CONVERT jako opcję parametru MQGET GetMsgOpts (patrz Rysunek 13 na stronie 177). Jeśli właściwość miejsca docelowego WMQ_RECEIVE_CONVERSION jest ustawiona na wartość WMQ_RECEIVE_CONVERSION_QMGR, menedżer kolejek wykonuje konwersję dla programu klienckiego JMS , który odbiera komunikat. Program kliencki JMS może również ustawić właściwość miejsca docelowego. Patrz sekcja Rysunek 12 na stronie 174.

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Lub,

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

*Rysunek 12. Włącz konwersję danych menedżera kolejek*

Główną zaletą konwersji menedżera kolejek jest możliwość wymiany komunikatów z aplikacjami innymi niżJMS . Jeśli pole Format w komunikacie jest zdefiniowane, a docelowy zestaw znaków lub kodowanie są inne niż w komunikacie, menedżer kolejek wykonuje konwersję danych dla aplikacji docelowej, jeśli aplikacja tego zażąda. Menedżer kolejek przekształca dane komunikatu sformatowane zgodnie z jednym z predefiniowanych typów komunikatów produktu IBM MQ , na przykład z nagłówkiem CICS bridge

[2] Jeden wyjątek: dane zapisane przy użyciu parametru writeUTF rozpoczynają się od pola o długości 2 bajtów

( MQCIH). Jeśli pole Format jest zdefiniowane przez użytkownika, menedżer kolejek szuka wyjścia konwersji danych o nazwie podanej w polu Format .

Konwersja danych menedżera kolejek jest używana w celu najlepszego efektu, gdy odbiorca "tworzy dobry" wzorzec projektu. Klient wysyłający JMS nie musi wykonywać konwersji. Program odbierający inny niżJMS korzysta z wyjścia konwersji w celu zapewnienia, że komunikat jest dostarczany z wymaganą stroną kodową i kodowaniem. W przypadku wysyłającego klienta JMS i odbiornika innego niżJMS przykład dotyczy klienta IBM MQ.

Za pomocą programu narzędziowego do obsługi wyjścia konwersji danych ( **crtmqcvx**) można utworzyć wyjście konwersji danych, aby umożliwić menedżerowi kolejek przekształcanie własnych sformatowanych danych rekordu. Można zbudować własny format rekordu, użyć klasy com.ibm.mq.headers , aby uzyskać do niego dostęp jako klasa Java , i użyć własnego wyjścia konwersji, aby go przekształcić. W systemie z/OS program narzędziowy ma nazwę **CSQUCVX**, a w systemie IBM i- **CVTMQMDTA**. Patrz sekcja “Wymiana sformatowanego rekordu z aplikacją inną niżJMS” na stronie 193.

## Konwersja danych kanału komunikatów

IBM MQ Kanały nadawcze, serwerowe, odbierające klastry i wysyłające klastry mają opcję konwersji komunikatów, CONVERT. Treść komunikatu może być opcjonalnie przekształcona podczas wysyłania komunikatu. Konwersja odbywa się na wysyłającym końcu kanału. Definicja odbiorcy klastra jest używana do automatycznego definiowania odpowiedniego kanału nadawczego klastra.

Konwersja danych przez kanały komunikatów jest zwykle używana, jeśli nie jest możliwe użycie innych form konwersji.

## Wybór podejścia do konwersji komunikatów: "odbiorca jest dobry"

Typowym podejściem w projekcie aplikacji IBM MQ do konwersji kodu jest "odbiorca sprawia, że jest dobry". Opcja "Odbiorca jest dobry" zmniejsza liczbę konwersji komunikatów. Pozwala to również uniknąć problemu z nieoczekiwanymi błędami kanału, jeśli konwersja komunikatów nie powiedzie się w jakimś pośrednim menedżerze kolejek podczas przesyłania komunikatów. Reguła "receiver make good" jest zerwana tylko wtedy, gdy istnieje powód, dla którego odbiorca nie może być dobry. Platforma odbierająca może nie mieć na przykład odpowiedniego zestawu znaków.

"Receiver sprawia, że dobry" jest również dobrym ogólnym poradnikiem dla aplikacji klienckich JMS . Jednak w szczególnych przypadkach konwersja na poprawny zestaw znaków w źródle może być bardziej wydajna. Konwersja z reprezentacji wewnętrznej maszyny JVM musi zostać przeprowadzona, gdy wysyłany jest komunikat zawierający typy tekstowe lub liczbowe. Konwersja na zestaw znaków wymagany przez odbiornik, jeśli odbiornik nie jest klientem JMS , może usunąć konieczność przeprowadzenia konwersji przez odbiorcę innego niżJMS . Jeśli odbiorca jest klientem JMS , mimo to zostanie ponownie przekształcony w celu zdekodowania danych komunikatu oraz utworzenia operacji podstawowych i obiektów Java .

Różnica między aplikacjami klienckimi JMS i aplikacjami napisanymi w języku takim jak C polega na tym, że program Java musi przeprowadzić konwersję danych. Aplikacja Java musi przekształcić liczby i tekst z ich wewnętrznej reprezentacji w zakodowany format używany w komunikatach.

Ustawiając miejsce docelowe lub właściwości komunikatu, można ustawić zestaw znaków i kodowanie używane przez produkt IBM MQ do kodowania liczb i tekstu w komunikatach. Zwykle pozostawiany jest zestaw znaków 1208 , a kodowanie Native.

Produkt IBM MQ nie przekształca tablic bajtów. Aby zakodować łańcuchy i tablice znaków w tablicach bajtów, należy użyć pakietu java.nio.charset . Charset określa zestaw znaków używany do przekształcania łańcucha lub tablicy znaków w tablicę bajtów. Można również zdekodować tablicę bajtów na łańcuch lub tablicę znaków za pomocą Charset. Podczas kodowania łańcuchów i tablic znaków nie zaleca się korzystania z języka java.nio.charset.Charset.defaultCodePage . Wartością domyślną parametru Charset jest zwykle windows-1252 w systemie Windowsi UTF-8 w systemie AIX and Linux. windows-1252 jest zestawem znaków jednobajtowych, a UTF-8 jest zestawem znaków wielobajtowych.

Zwykle podczas wymiany komunikatów z innymi aplikacjami JMS należy pozostawić domyślne wartości właściwości kodowania i zestawu znaków miejsca docelowego ( `UTF-8` i `Native` ). Jeśli komunikaty zawierające cyfry lub tekst są wymieniane z aplikacją JMS , należy wybrać jeden z typów komunikatów `JMSTextMessage`, `JMSStreamMessage`, `JMSMapMessage`lub `JMSObjectMessage` , który jest odpowiedni dla danego celu. Brak innych zadań konwersji do wykonania.

W przypadku wymiany komunikatów z aplikacjami innymi niżJMS , które używają formatu rekordu, jest to bardziej skomplikowane. Jeśli cały rekord nie zawiera tekstu i można go przesłać jako `JMSTextMessage`, należy zakodować i zdekodować tekst w aplikacji. Ustaw typ komunikatu docelowego na `MQ`i użyj `JMSBytesMessage` , aby uniknąć dodawania przez IBM MQ classes for JMS dodatkowego nagłówka i informacji o znacznikach do danych komunikatu. Metody `JMSBytesMessage` służą do zapisywania liczb i bajtów, a klasa `Charset` jawnie przekształca tekst w tablice bajtów. Na wybór zestawu znaków może mieć wpływ wiele czynników:

- Wydajność: Czy można zmniejszyć liczbę konwersji przez przekształcenie tekstu w zestaw znaków, który jest używany na największej liczbie serwerów?
- Jednolitość: wszystkie komunikaty są przesyłane w tym samym zestawie znaków.
- Bogactwo: jakie zestawy znaków mają wszystkie punkty kodowe, których muszą używać aplikacje?
- Prostota: zestawy znaków jednobajtowych są prostsze w użyciu niż zestawy znaków o zmiennej długości i wielobajtowych.

Patrz “Wymiana sformatowanego rekordu z aplikacją inną niżJMS” na stronie 193. Przykłady konwersji komunikatów wymienianych z aplikacjami innymi niżJMS .

**Przykłady**

**Tabela typów komunikatów i typów konwersji**

*Tabela 31. Typy komunikatów i typy konwersji*

| | **Typ konwersji** | | | |
|---|---|---|---|---|
| **Typ komunikatu** | **Tekst** | **Liczbowy** | **Inne** | **Brak** |
| `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` |

| *Tabela 31. Typy komunikatów i typy konwersji (kontynuacja)* | | | | |
|---|---|---|---|---|
| | **Typ konwersji** | | | |
| **Typ komunikatu** | **Tekst** | **Liczbowy** | **Inne** | **Brak** |
| JMSStreamMessage | readString<br>writeString | readDouble<br>readFloat<br>readInt<br>readLong<br>readShort<br>writeDouble<br>writeFloat<br>writeInt<br>writeLong<br>writeShort | readBoolean<br>readObject<br>writeBoolean<br>writeObject | readByte<br>readBytes<br>readChar<br>writeByte<br>writeBytes<br>writeChar |
| JMSMapMessage | getString<br>setString | getDouble<br>getFloat<br>getInt<br>getLong<br>getShort<br>setDouble<br>setFloat<br>setInt<br>setLong<br>setShort | getBoolean<br>getObject<br>setBoolean<br>setObject | getByte<br>getBytes<br>readChar<br>setByte<br>setBytes<br>setChar |

**Wywoływanie konwersji danych z programu w języku C**

```
gmo.Options = MQGMO_WAIT         /* wait for new messages           */
            | MQGMO_NO_SYNCPOINT /* no transaction                 */
            | MQGMO_CONVERT;     /* convert if necessary            */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle               */
           Hobj,                /* object handle                   */
           &md,                 /* message descriptor              */
           &gmo,                /* get message options             */
           buflen,              /* buffer length                   */
           buffer,              /* message buffer                  */
           &messlen,            /* message length                  */
           &CompCode,           /* completion code                 */
           &Reason);            /* reason code                     */
```

*Rysunek 13. Fragment kodu z pliku amqsget0.c*

**Wysyłanie i odbieranie tekstu w JMSBytesMessage**

Kod w pliku Rysunek 14 na stronie 178 wysyła łańcuch w komunikacie BytesMessage. Dla uproszczenia przykład wysyła pojedynczy łańcuch, dla którego bardziej odpowiedni jest łańcuch JMSTextMessage . Aby otrzymać łańcuch tekstowy w bajtach, zawierający mieszaninę typów, należy znać długość łańcucha w bajtach, o nazwie *TEXT_LENGTH* w Rysunek 15 na stronie 178. Nawet w przypadku łańcucha o stałej liczbie znaków długość reprezentacji bajtowej może być dłuższa.

```
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);
```

*Rysunek 14. Wysyłanie `String` w `JMSBytesMessage`*

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

*Rysunek 15. Odbieranie `String` z `JMSBytesMessage`*

**Pojęcia pokrewne**

Konwersja i kodowanie komunikatów klienta JMS
Metody używane do przeprowadzania konwersji i kodowania komunikatów klienta JMS zostały wymienione na liście wraz z przykładami kodu dla każdego typu konwersji.

Konwersja danych menedżera kolejek
Konwersja danych menedżera kolejek była zawsze dostępna dla aplikacji innych niż aplikacjeJMS , które odbierały komunikaty od klientów JMS . Klienty JMS odbierające komunikaty również używają opcjonalnej konwersji danych menedżera kolejek.

**Zadania pokrewne**

Wymiana sformatowanego rekordu z aplikacją inną niżJMS
Wykonaj kroki sugerowane w tym zadaniu, aby zaprojektować i zbudować wyjście konwersji danych oraz aplikację kliencką JMS , która może wymieniać komunikaty z aplikacją inną niżJMS przy użyciu języka `JMSBytesMessage`. Wymiana sformatowanego komunikatu z aplikacją inną niżJMS może odbywać się z wyjściem konwersji danych lub bez niego.

**Odsyłacze pokrewne**

Typy i konwersja komunikatów JMS
Wybór typu komunikatu ma wpływ na podejście do konwersji komunikatów. Interakcja konwersji komunikatu i typu komunikatu jest opisana dla typów komunikatów JMS , `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage`i `JMSBytesMessage`.

*Typy i konwersja komunikatów JMS*
Wybór typu komunikatu ma wpływ na podejście do konwersji komunikatów. Interakcja konwersji komunikatu i typu komunikatu jest opisana dla typów komunikatów JMS , `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage`i `JMSBytesMessage`.

## `JMSObjectMessage`

Plik `JMSObjectMessage` zawiera jeden obiekt i wszystkie obiekty, do których się odwołuje, serializowane do strumienia bajtów przez maszynę JVM. Tekst jest serializowany do postaci UTF-8i ograniczony do łańcuchów lub tablic znaków o długości nie większej niż 65534 bajty. Zaletą produktu `JMSObjectMessage` jest to, że aplikacje nie biorą udziału w żadnych problemach z konwersją danych, o ile używają tylko metod i atrybutów obiektu. Produkt `JMSObjectMessage` udostępnia konwersję danych dla obiektów złożonych bez konieczności rozważania przez programistę aplikacji sposobu kodowania obiektu w komunikacie. Wadą korzystania z produktu `JMSObjectMessage` jest to, że może on być wymieniany tylko z innymi aplikacjami JMS . Wybierając jeden z innych typów komunikatów JMS , można wymieniać komunikaty JMS z aplikacjami innymi niżJMS .

"Wysyłanie i odbieranie JMSObjectMessage" na stronie 181 przedstawia obiekt String wymieniany w komunikacie.

Aplikacja kliencka JMS może odebrać JMSObjectMessage tylko w komunikacie, który ma treść w stylu JMS. Miejsce docelowe musi określać treść stylu JMS .

## JMSTextMessage

JMSTextMessage zawiera pojedynczy łańcuch tekstowy. Po wysłaniu komunikatu tekstowego tekst Format jest ustawiany na "MQSTR ", WMQConstants.MQFMT_STRING. Wartość CodedCharacterSetId tekstu jest ustawiana na identyfikator kodowanego zestawu znaków zdefiniowany dla miejsca docelowego. Tekst jest kodowany w pliku CodedCharacterSetId przez IBM MQ. Pola CodedCharacterSetId i Format są ustawiane w deskryptorze komunikatu MQMDlub w polach JMS w pliku MQRFH2. Jeśli komunikat jest zdefiniowany jako mający styl treści komunikatu WMQ_MESSAGE_BODY_MQ lub nie określono stylu treści, ale miejscem docelowym jest element WMQ_TARGET_DEST_MQ, pola deskryptora komunikatu są ustawiane. W przeciwnym razie komunikat będzie miał JMS RFH2 , a pola zostaną ustawione w stałej części pliku MQRFH2.

Aplikacja może nadpisać identyfikator kodowanego zestawu znaków zdefiniowany dla miejsca docelowego. Musi on ustawić właściwość komunikatu JMS_IBM_CHARACTER_SET na identyfikator kodowanego zestawu znaków; patrz przykład w sekcji "Wysyłanie i odbieranie JMSTextmessage" na stronie 181.

Gdy klient JMS wywołuje konwersję menedżera kolejek metody consumer.receive , jest ona opcjonalna. Konwersję menedżera kolejek można włączyć, ustawiając właściwość miejsca docelowego WMQ_RECEIVE_CONVERSION na wartość WMQ_RECEIVE_CONVERSION_QMGR. Przed przesłaniem komunikatu do klienta JMS menedżer kolejek przekształca komunikat tekstowy z wartości JMS_IBM_CHARACTER_SET określonej dla komunikatu. Zestaw znaków przekształconego komunikatu to 1208, UTF-8, chyba że miejsce docelowe ma inną wartość WMQ_RECEIVE_CCSID. Element CodedCharacterSetId w komunikacie, który odwołuje się do elementu JMSTextMessage , zostanie zaktualizowany do docelowego identyfikatora zestawu znaków. Tekst jest dekodowany z docelowego zestawu znaków do kodu Unicode za pomocą metody getText ; patrz przykład w sekcji "Wysyłanie i odbieranie JMSTextmessage" na stronie 181.

Element JMSTextMessage może być wysyłany w treści komunikatu w stylu produktu MQbez nagłówka JMS MQRFH2 . Wartość atrybutów miejsca docelowego WMQ_MESSAGE_BODY i WMQ_TARGET_DEST określa styl treści komunikatu, chyba że zostanie przesłonięta przez aplikację. Aplikacja może przesłonić wartości ustawione w miejscu docelowym, wywołując metodę destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) lub destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ).

Jeśli produkt JMSTextMessage jest wysyłany z treścią w stylu produktu MQ do miejsca docelowego, w którym właściwość WMQ_MESSAGE_BODY jest ustawiona na wartość WMQ_MESSAGE_BODY_MQ, nie można go odebrać jako JMSTextMessage z tego samego miejsca docelowego. Wszystkie komunikaty odebrane z miejsca docelowego z wartością WMQ_MESSAGE_BODY ustawioną na wartość WMQ_MESSAGE_BODY_MQ są odbierane jako JMSBytesMessage. Próba odebrania komunikatu jako JMSTextMessage powoduje wystąpienie wyjątku ClassCastException: com.ibm.jms.JMSBytesMessage cannot be cast to jakarta (or javax).jms.TextMessage.

**Uwaga:** Tekst w JMSBytesMessage nie jest przekształcany przez klienta JMS. Klient może odebrać tekst w komunikacie tylko w postaci tablicy bajtów. Jeśli konwersja menedżera kolejek jest włączona, tekst jest przekształcany przez menedżer kolejek, ale klient JMS nadal musi go odbierać jako tablicę bajtów w JMSBytesMessage.

Zwykle lepiej jest użyć właściwości WMQ_TARGET_DEST w celu określenia, czy obiekt JMSTextMessage jest wysyłany ze stylem treści MQ lub JMS. Następnie można odebrać komunikat z miejsca docelowego, w którym właściwość WMQ_TARGET_DEST ma wartość WMQ_TARGET_DEST_MQ lub WMQ_TARGET_DEST_JMS. Wartość WMQ_TARGET_DEST nie ma wpływu na odbiornik.

## JMSMapMessage i JMSStreamMessage

Te dwa typy komunikatów JMS są podobne. Typy podstawowe można odczytywać i zapisywać w komunikatach przy użyciu metod opartych na interfejsach DataInputStream i DataOutputStream (patrz sekcja “Tabela typów komunikatów i typów konwersji” na stronie 184). Szczegółowe informacje zawiera sekcja “Konwersja i kodowanie komunikatów klienta JMS” na stronie 185. Każda operacja podstawowa jest oznaczona; patrz sekcja “Treść komunikatu JMS” na stronie 170.

Dane liczbowe są odczytywane i zapisywane w komunikacie zakodowanym jako tekst XML. Nie jest tworzone odwołanie do właściwości miejsca docelowego JMS_IBM_ENCODING. Dane tekstowe są traktowane tak samo, jak tekst w JMSTextMessage. Jeśli użytkownik spojrzy na treść komunikatu utworzonego w przykładzie w sekcji Rysunek 20 na stronie 182, wszystkie dane komunikatu będą w kodzie EBCDIC, ponieważ zostały wysłane z wartością zestawu znaków równą 37.

Można wysłać wiele elementów w JMSMapMessage lub JMSStreamMessage.

Poszczególne elementy danych można pobierać według nazwy z JMSMapMessagelub według pozycji z JMSStreamMessage. Każdy element jest dekodowany, gdy metoda get lub read jest wywoływana przy użyciu wartości CodedCharacterSetId zapisanej w komunikacie. Jeśli metoda użyta do pobrania elementu zwraca inny typ niż typ, który został wysłany, typ jest przekształcany. Jeśli nie można przekształcić typu, zgłaszany jest wyjątek. Szczegółowe informacje na ten temat zawiera sekcja Klasa JMSStreamMessage . Przykład w sekcji “Wysyłanie danych w JMSStreamMessage i JMSMapMessage” na stronie 182 ilustruje konwersję typów i pobieranie treści JMSMapMessage poza kolejnością.

Pole MQRFH2.format dla zmiennych JMSMapMessage i JMSStreamMessage jest ustawione na wartość "MQSTR ". Jeśli właściwość miejsca docelowego WMQ_RECEIVE_CONVERSION ma wartość WMQ_RECEIVE_CONVERSION_QMGR, dane komunikatu są przekształcane przez menedżer kolejek przed wysłaniem do klienta JMS . MQRFH2.CodedCharacterSetId komunikatu to WMQ_RECEIVE_CCSID miejsca docelowego. Parametr MQRFH2.Encoding ma wartość Native. Jeśli zmienna WMQ_RECEIVE_CONVERSION ma wartość WMQ_RECEIVE_CONVERSION_CLIENT_MSG , CodedCharacterSetId , a zmienna Encoding zmiennej MQRFH2 jest wartością ustawioną przez nadawcę.

Aplikacja kliencka JMS może odebrać element JMSMapMessage lub JMSStreamMessage tylko w komunikacie, który ma treść w stylu JMS, i z miejsca docelowego, które nie określa treści w stylu MQ .

## JMSBytesMessage

Element JMSBytesMessage może zawierać wiele typów podstawowych. Typy podstawowe można odczytywać i zapisywać w komunikatach przy użyciu metod opartych na interfejsach DataInputStream i DataOutputStream (patrz sekcja “Tabela typów komunikatów i typów konwersji” na stronie 184). Szczegółowe informacje zawiera sekcja “Typy i konwersja komunikatów JMS” na stronie 178.

Kodowaniem danych liczbowych w komunikacie steruje wartość parametru JMS_IBM_ENCODING , który jest ustawiany przed zapisaniem danych liczbowych w pliku JMSBytesMessage. Aplikacja może nadpisać domyślne kodowanie Native zdefiniowane dla JMSBytesMessage , ustawiając właściwość komunikatu JMS_IBM_ENCODING.

Dane tekstowe mogą być odczytywane i zapisywane w systemie UTF-8 przy użyciu metod readUTF i writeUTFlub w kodzie Unicode przy użyciu metod readChar i writeChar . Nie ma metod, które korzystają z języka CodedCharacterSetId. Alternatywnie klient JMS może kodować i dekodować tekst na bajty przy użyciu klasy Charset . Przesyła on bajty między maszyną JVM i komunikatem bez wykonywania konwersji przez IBM MQ classes for JMS ; patrz sekcja “Wysyłanie i odbieranie tekstu w JMSBytesMessage” na stronie 182.

Element JMSBytesMessage wysyłany do aplikacji MQ jest zwykle wysyłany w treści komunikatu w stylu produktu MQbez nagłówka JMS MQRFH2 . Jeśli jest on wysyłany do aplikacji JMS , styl treści komunikatu to zazwyczaj JMS. Wartość atrybutów miejsca docelowego WMQ_MESSAGE_BODY i WMQ_TARGET_DEST określa styl treści komunikatu, chyba że zostanie przesłonięta przez aplikację. Aplikacja może przesłonić wartości ustawione w miejscu docelowym, wywołując

metodę `destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ)` lub `destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ)`.

W przypadku wysłania komunikatu `JMSBytesMessage` z treścią w stylu MQ można odebrać komunikat z miejsca docelowego, które definiuje styl treści komunikatu MQ lub JMS . Jeśli zostanie wysłany komunikat `JMSBytesMessage` z treścią w stylu JMS, należy odebrać komunikat z miejsca docelowego, które definiuje styl treści komunikatu JMS . W przeciwnym razie zmienna `MQRFH2` będzie traktowana jako część danych komunikatu użytkownika, co może być niezgodne z oczekiwaniami.

Niezależnie od tego, czy komunikat ma styl treści MQ , czy JMS , ustawienie `WMQ_TARGET_DEST`nie ma wpływu na sposób jego odbierania.

Komunikat może zostać później przetransformowany przez menedżer kolejek, jeśli dla danych komunikatu zostanie podana wartość `Format` , a konwersja danych menedżera kolejek jest włączona. Nie należy używać pola formatu dla niczego innego niż określanie formatu danych komunikatu lub pozostawienie pustego pola `MQConstants.MQFMT_NONE`

W `JMSBytesMessage`można wysłać wiele elementów. Każdy element liczbowy jest przekształcany podczas wysyłania komunikatu przy użyciu kodowania zdefiniowanego dla komunikatu.

Poszczególne elementy danych można pobrać z programu `JMSBytesMessage`. Aby utworzyć komunikat, wywołaj metody odczytu w tej samej kolejności, w jakiej zostały wywołane metody zapisu. Każdy element liczbowy jest przekształcany, gdy komunikat jest wywoływany przy użyciu wartości `Encoding` zapisanej w komunikacie.

W przeciwieństwie do produktów `JMSMapMessage` i `JMSStreamMessage`plik `JMSBytesMessage` zawiera tylko dane zapisane przez aplikację. W danych komunikatu nie są zapisywane żadne dodatkowe dane, takie jak znaczniki XML używane do definiowania elementów w plikach `JMSMapMessage` i `JMSStreamMessage`. Z tego powodu do przesyłania komunikatów sformatowanych dla innych aplikacji należy użyć `JMSBytesMessage` .

Konwersja między produktami `JMSBytesMessage` i `DataInputStream` oraz `DataOutputStream` jest przydatna w niektórych aplikacjach. Kod oparty na przykładzie “Odczytywanie i zapisywanie komunikatów przy użyciu produktów DataInputStream i DataOutputStream” na stronie 183jest niezbędny do użycia pakietu `com.ibm.mq.header` z JMS.

### Przykłady

#### Wysyłanie i odbieranie `JMSObjectMessage`

```
ObjectMessage omo = session.createObjectMessage();
omo.setObject(new String("A string"));
producer.send(omo);
...
ObjectMessage omi = (ObjectMessage)consumer.receive();
System.out.println((String)omi.getObject());
...
A string
```

*Rysunek 16. Wysyłanie i odbieranie `JMSObjectMessage`*

#### Wysyłanie i odbieranie `JMSTextmessage`

Komunikat tekstowy nie może zawierać tekstu w różnych zestawach znaków. Przykład przedstawia tekst w różnych zestawach znaków, wysłany w dwóch różnych komunikatach.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

*Rysunek 17. Wyślij komunikat tekstowy w zestawie znaków zdefiniowanym przez miejsce docelowe*

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

*Rysunek 18. Wyślij wiadomość tekstową w ccsid 37*

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

*Rysunek 19. Odbierz komunikat tekstowy*

### Wysyłanie danych w JMSStreamMessage i JMSMapMessage

```
StreamMessage smo = session.createStreamMessage();
smo.writeString("256");
smo.writeInt(512);
smo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(smo);
...
MapMessage mmo = session.createMapMessage();
mmo.setString("First", "256");
mmo.setInt("Second", 512);
mmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(mmo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println("Stream: First as float "   + smi.readFloat() +
                         " Second as String " + smi.readString());
...
Stream: First as float: 256.0, Second as String: 512
...
MapMessage mmi = (MapMessage)consumer.receive();
System.out.println("Map: Second as String " + mmi.getString("Second") +
                       " First as double "  + mmi.getDouble("First"));
...
Map: Second as String: 512, First as double: 256.0
```

*Rysunek 20. Wysyłanie danych w systemach JMSStreamMessage i JMSMapMessage*

### Wysyłanie i odbieranie tekstu w JMSBytesMessage

Kod w pliku Rysunek 21 na stronie 183 wysyła łańcuch w komunikacie BytesMessage. Dla uproszczenia przykład wysyła pojedynczy łańcuch, dla którego bardziej odpowiedni jest łańcuch JMSTextMessage . Aby otrzymać łańcuch tekstowy w bajtach, zawierający mieszaninę typów, należy znać długość łańcucha w bajtach, o nazwie *TEXT_LENGTH* w Rysunek 22 na stronie 183. Nawet w przypadku łańcucha o stałej liczbie znaków długość reprezentacji bajtowej może być dłuższa.

```
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);
```

*Rysunek 21. Wysyłanie `String` w `JMSBytesMessage`*

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

*Rysunek 22. Odbieranie `String` z `JMSBytesMessage`*

### Odczytywanie i zapisywanie komunikatów przy użyciu produktów `DataInputStream` i `DataOutputStream`

Kod w pliku Rysunek 23 na stronie 183 tworzy `JMSBytesMessage` za pomocą `DataOutputStream`.

```
ByteArrayOutputStream bout = new ByteArrayOutputStream();
DataOutputStream dout = new DataOutputStream(bout);
BytesMessage messageOut = prod.session.createBytesMessage();
// messageOut.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
//                           ((MQDestination) (prod.destination)).getIntProperty
//                           (WMQConstants.WMQ_ENCODING));
int ccsidOut = (((MQDestination)prod.destination).getIntProperty(WMQConstants.WMQ_CCSID));
String codePageOut = CCSID.getCodepage(ccsidOut);
dout.writeInt(ccsidOut);
dout.write(codePageOut.getBytes(codePageOut));
messageOut.writeBytes(bout.toByteArray());
producer.send(messageOut);
```

*Rysunek 23. Wysyłanie `JMSBytesMessage` za pomocą `DataOutputStream`*

Instrukcja ustawiająca właściwość `JMS_IBM_ENCODING` jest przekształcona w komentarz. Instrukcja jest poprawna w przypadku zapisu bezpośrednio w `JMSBytesMessage`, ale nie ma wpływu na zapis w `DataOutputStream`. Liczby zapisywane w pliku `DataOutputStream` są kodowane w języku `Native` . Ustawienie `JMS_IBM_ENCODING` nie ma żadnego efektu.

Kod w pliku Rysunek 24 na stronie 184 otrzymuje `JMSBytesMessage` za pomocą `DataInputStream`.

```
static final int ccsidIn_SIZE = (Integer.SIZE)/8;
...
connection.start();
BytesMessage messageIn = (BytesMessage) consumer.receive();
int messageLength = new Long(messageIn.getBodyLength()).intValue();
byte [] bin = new byte[messageLength];
messageIn.readBytes(bin, messageLength);
DataInputStream din = new DataInputStream(new ByteArrayInputStream(bin));
int ccsidIn = din.readInt();
byte [] codePageByte = new byte[messageLength - ccsidIn_SIZE];
din.read(codePageByte, 0, codePageByte.length);
System.out.println("CCSID " + ccsidIn + " code page " + new String(codePageByte,
                    messageIn.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET)));
```

*Rysunek 24. Odbieranie JMSBytesMessage za pomocą DataInputStream*

Strona kodowa jest drukowana przy użyciu właściwości strony kodowej danych komunikatu wejściowego, JMS_IBM_CHARACTER_SET. Na wejściu JMS_IBM_CHARACTER_SET jest stroną kodową Java , a nie liczbowym identyfikatorem kodowanego zestawu znaków.

**Tabela typów komunikatów i typów konwersji**

*Tabela 32. Typy komunikatów i typy konwersji*

| | **Typ konwersji** | | | |
|---|---|---|---|---|
| **Typ komunikatu** | **Tekst** | **Liczbowy** | **Inne** | **Brak** |
| JMSObjectMessage | | | | getObject<br>setObject |
| JMSTextMessage | getText<br>setText | | | |
| JMSBytesMessage | readUTF<br>writeUTF | readDouble<br>readFloat<br>readInt<br>readLong<br>readShort<br>readUnsignedShort<br>writeDouble<br>writeFloat<br>writeInt<br>writeLong<br>writeShort | readBoolean<br>readObject<br>writeBoolean<br>writeObject | readByte<br>readUnsignedByte<br>readBytes<br>readChar<br>writeByte<br>writeBytes<br>writeChar |

| *Tabela 32. Typy komunikatów i typy konwersji (kontynuacja)* | | | | |
|---|---|---|---|---|
| | **Typ konwersji** | | | |
| **Typ komunikatu** | **Tekst** | **Liczbowy** | **Inne** | **Brak** |
| `JMSStreamMessage` | `readString`<br>`writeString` | `readDouble`<br>`readFloat`<br>`readInt`<br>`readLong`<br>`readShort`<br>`writeDouble`<br>`writeFloat`<br>`writeInt`<br>`writeLong`<br>`writeShort` | `readBoolean`<br>`readObject`<br>`writeBoolean`<br>`writeObject` | `readByte`<br>`readBytes`<br>`readChar`<br>`writeByte`<br>`writeBytes`<br>`writeChar` |
| `JMSMapMessage` | `getString`<br>`setString` | `getDouble`<br>`getFloat`<br>`getInt`<br>`getLong`<br>`getShort`<br>`setDouble`<br>`setFloat`<br>`setInt`<br>`setLong`<br>`setShort` | `getBoolean`<br>`getObject`<br>`setBoolean`<br>`setObject` | `getByte`<br>`getBytes`<br>`readChar`<br>`setByte`<br>`setBytes`<br>`setChar` |

**Pojęcia pokrewne**

Podejścia do konwersji komunikatów JMS
Wiele podejść do konwersji danych jest otwartych dla projektantów aplikacji JMS . Te podejścia nie są wyłączne; niektóre aplikacje mogą korzystać z ich kombinacji. Jeśli aplikacja wymienia tylko tekst lub komunikaty tylko z innymi aplikacjami JMS , zwykle nie jest rozważana konwersja danych. Konwersja danych jest wykonywana automatycznie przez IBM MQ.

Konwersja i kodowanie komunikatów klienta JMS
Metody używane do przeprowadzania konwersji i kodowania komunikatów klienta JMS zostały wymienione na liście wraz z przykładami kodu dla każdego typu konwersji.

Konwersja danych menedżera kolejek
Konwersja danych menedżera kolejek była zawsze dostępna dla aplikacji innych niż aplikacjeJMS , które odbierały komunikaty od klientów JMS . Klienty JMS odbierające komunikaty również używają opcjonalnej konwersji danych menedżera kolejek.

**Zadania pokrewne**

Wymiana sformatowanego rekordu z aplikacją inną niżJMS
Wykonaj kroki sugerowane w tym zadaniu, aby zaprojektować i zbudować wyjście konwersji danych oraz aplikację kliencką JMS , która może wymieniać komunikaty z aplikacją inną niżJMS przy użyciu języka `JMSBytesMessage`. Wymiana sformatowanego komunikatu z aplikacją inną niżJMS może odbywać się z wyjściem konwersji danych lub bez niego.

*Konwersja i kodowanie komunikatów klienta JMS*
Metody używane do przeprowadzania konwersji i kodowania komunikatów klienta JMS zostały wymienione na liście wraz z przykładami kodu dla każdego typu konwersji.

Konwersja i kodowanie występują, gdy operacje podstawowe lub obiekty Java są odczytywane lub zapisywane z i do komunikatów JMS . Konwersja jest nazywana konwersją danych klienta JMS , aby odróżnić ją od konwersji danych menedżera kolejek i konwersji danych aplikacji. Konwersja jest wykonywana tylko wtedy, gdy dane są odczytywane lub zapisywane w komunikacie JMS . Tekst jest

konwertowany na i z wewnętrznej 16-bitowej reprezentacji Unicode[3]do zestawu znaków używanego dla tekstu w komunikatach. Dane liczbowe są przekształcane w prymitywne typy liczbowe Java i przekształcane w kodowanie zdefiniowane dla komunikatu. To, czy wykonywana jest konwersja i jaki jest jej typ, zależy od typu komunikatu JMS i operacji odczytu lub zapisu.

Tabela 33 na stronie 186 klasyfikuje metody odczytu i zapisu dla różnych typów komunikatów JMS według typu wykonywanej konwersji. Typy konwersji są opisane w tekście znajdującym się poniżej tabeli.

*Tabela 33. Typy komunikatów i typy konwersji*

| | **Typ konwersji** | | | |
|---|---|---|---|---|
| **Typ komunikatu** | **Tekst** | **Liczbowy** | **Inne** | **Brak** |
| JMSObjectMessage | | | | getObject<br>setObject |
| JMSTextMessage | getText<br>setText | | | |
| JMSBytesMessage | readUTF<br>writeUTF | readDouble<br>readFloat<br>readInt<br>readLong<br>readShort<br>readUnsignedShort<br>writeDouble<br>writeFloat<br>writeInt<br>writeLong<br>writeShort | readBoolean<br>readObject<br>writeBoolean<br>writeObject | readByte<br>readUnsignedByte<br>readBytes<br>readChar<br>writeByte<br>writeBytes<br>writeChar |
| JMSStreamMessage | readString<br>writeString | readDouble<br>readFloat<br>readInt<br>readLong<br>readShort<br>writeDouble<br>writeFloat<br>writeInt<br>writeLong<br>writeShort | readBoolean<br>readObject<br>writeBoolean<br>writeObject | readByte<br>readBytes<br>readChar<br>writeByte<br>writeBytes<br>writeChar |
| JMSMapMessage | getString<br>setString | getDouble<br>getFloat<br>getInt<br>getLong<br>getShort<br>setDouble<br>setFloat<br>setInt<br>setLong<br>setShort | getBoolean<br>getObject<br>setBoolean<br>setObject | getByte<br>getBytes<br>readChar<br>setByte<br>setBytes<br>setChar |

[3] Niektóre reprezentacje Unicode wymagają więcej niż 16 bitów. Patrz Java SE.

**Tekst**

Domyślny CodedCharacterSetId dla miejsca docelowego to 1208, UTF-8. Domyślnie tekst jest przekształcany z formatu Unicode i wysyłany jako łańcuch tekstowy UTF-8 . Podczas odbierania tekst jest przekształcany z kodowanego zestawu znaków w komunikacie odebranym przez klienta na kod Unicode.

Metody setText i writeString dokonują konwersji tekstu z formatu Unicode na zestaw znaków zdefiniowany dla miejsca docelowego. Aplikacja może nadpisać zestaw znaków miejsca docelowego, ustawiając właściwość komunikatu JMS_IBM_CHARACTER_SET. JMS_IBM_CHARACTER_SETpodczas wysyłania komunikatu musi być liczbowym identyfikatorem kodowanego zestawu znaków[4].

Fragmenty kodu w pliku “Wysyłanie i odbieranie JMSTextmessage” na stronie 189 wysyłają dwa komunikaty. Jeden jest wysyłany w zestawie znaków zdefiniowanym dla miejsca docelowego, a drugi w zestawie znaków 37, zdefiniowanym przez aplikację.

Metody getText i readString przekształcają tekst w komunikacie z zestawu znaków zdefiniowanego w komunikacie na kod Unicode. Metody używają strony kodowej zdefiniowanej we właściwości komunikatu JMS_IBM_CHARACTER_SET. Strona kodowa jest odwzorowywana z elementu MQRFH2.CodedCharacterSetId , chyba że komunikat jest komunikatem typu MQi nie ma elementu MQRFH2. Jeśli komunikat jest komunikatem typu MQbez typu MQRFH2, strona kodowa jest odwzorowywana z pliku MQMD.CodedCharacterSetId.

Fragment kodu w pliku Rysunek 29 na stronie 190 odbiera komunikat wysłany do miejsca docelowego. Tekst w komunikacie jest przekształcany ze strony kodowej IBM037 z powrotem na stronę kodową Unicode.

**Uwaga:** Prostym sposobem sprawdzenia, czy tekst został przekształcony w kodowany zestaw znaków 37, jest użycie Eksploratora IBM MQ . Przejrzyj kolejkę i wyświetl właściwości komunikatu przed jego pobraniem.

Porównaj fragment kodu w pliku Rysunek 28 na stronie 189 z niepoprawnym fragmentem kodu w pliku Rysunek 25 na stronie 187. W niepoprawnym fragmencie kodu łańcuch tekstowy jest przekształcany dwukrotnie, raz przez aplikację i ponownie przez IBM MQ.

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText(new String("Sent in EBCDIC character set 37".getBytes(CCSID.getCodepage(37))));
producer.send(tmo);
```

*Rysunek 25. Niepoprawna konwersja strony kodowej*

Metoda writeUTF konwertuje tekst z Unicode do 1208, UTF-8. Łańcuch tekstowy jest poprzedzony 2-bajtową długością. Maksymalna długość łańcucha tekstowego wynosi 65534 bajty. Metoda readUTF odczytuje element w komunikacie zapisanym przez metodę writeUTF . Odczytuje dokładnie liczbę bajtów zapisanych przez metodę writeUTF .

**Liczbowy**

Domyślnym kodowaniem liczbowym dla miejsca docelowego jest Native. Stała kodowania Native dla Java ma wartość 273, x'00000111', która jest taka sama dla wszystkich platform. Podczas odbierania liczby w komunikacie są poprawnie transformowane w liczbowe operacje podstawowe Java . Transformacja używa kodowania zdefiniowanego w komunikacie i typu zwracanego przez metodę read.

Metoda send przekształca liczby dodane do komunikatu przez set i write w kodowanie liczbowe zdefiniowane dla miejsca docelowego. Kodowanie miejsca docelowego może zostać przesłonięte

[4] Podczas odbierania komunikatu JMS_IBM_CHARACTER_SET jest to nazwa strony kodowej produktu Java Charset .

dla komunikatu przez ustawienie przez aplikację właściwości komunikatu JMS_IBM_ENCODING . na przykład:

```
message.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
WMQConstants.WMQ_ENCODING_INTEGER_REVERSED);
```

Metody liczbowe get i read przekształcają liczby w komunikacie z kodowania liczbowego zdefiniowanego w komunikacie. Liczby są przekształcane w typ określony przez metodę read lub get . Więcej informacji na ten temat zawiera sekcja Właściwość ENCODING. Metody używają kodowania zdefiniowanego w pliku JMS_IBM_ENCODING. Kodowanie jest odwzorowywane z pliku MQRFH2.Encoding , chyba że komunikat jest komunikatem typu MQi nie ma wartości MQRFH2. Jeśli komunikat jest komunikatem typu MQbez atrybutu MQRFH2, metody używają kodowania zdefiniowanego w pliku MQMD.Encoding.

W przykładzie, który przedstawia Rysunek 30 na stronie 190 , przedstawiono aplikację kodującą liczbę w formacie docelowym i wysyłającą ją w formacie JMSStreamMessage. Porównaj przykład w sekcji Rysunek 30 na stronie 190 z przykładem w sekcji Rysunek 31 na stronie 190. Różnica polega na tym, że parametr JMS_IBM_ENCODING musi być ustawiony w JMSBytesMessage.

**Uwaga:** Prostym sposobem sprawdzenia, czy liczba jest poprawnie zakodowana, jest użycie programu IBM MQ Explorer. Przejrzyj kolejkę i wyświetl właściwości komunikatu przed jego zużyciem.

**Inne**

Metody boolean kodują metody true i false jako x'01' i x'00' w systemach JMSByteMessage, JMSStreamMessagei JMSMapMessage.

Metody UTF kodują i dekodują kod Unicode do łańcuchów tekstowych UTF-8 . Łańcuchy są ograniczone do mniej niż 65536 znaków i są poprzedzone polem o długości 2 bajtów.

Metody Object zawijają typy podstawowe jako obiekty. Typy liczbowe i tekstowe są kodowane lub przekształcane tak, jakby typy podstawowe były odczytywane lub zapisywane przy użyciu metod numerycznych i tekstowych.

**Brak**

Metody readByte, readBytes, readUnsignedByte, writeBytei writeBytes powodują pobranie lub umieszczenie pojedynczych bajtów lub tablic bajtów między aplikacją a komunikatem bez konwersji. Metody readChar i writeChar dostają i umieszczają 2-bajtowe znaki Unicode między aplikacją i komunikatem bez konwersji.

Za pomocą metod readBytes i writeBytes aplikacja może wykonać własną konwersję punktu kodowego, tak jak w przypadku metody “Wysyłanie i odbieranie tekstu w JMSBytesMessage” na stronie 190.

IBM MQ nie wykonuje żadnej konwersji strony kodowej w kliencie, ponieważ komunikat jest komunikatem JMSBytesMessage, a także dlatego, że używane są metody readBytes i writeBytes . Jeśli jednak bajty reprezentują tekst, należy upewnić się, że strona kodowa używana przez aplikację jest zgodna z kodowanym zestawem znaków w miejscu docelowym. Komunikat może zostać ponownie przekształcony przez wyjście konwersji menedżera kolejek. Inną możliwością jest użycie w odbierającym programie klienckim JMS konwencji konwersji tablic bajtów reprezentujących tekst w komunikacie na łańcuchy lub znaki przy użyciu właściwości JMS_IBM_CHARACTER_SET w komunikacie.

W tym przykładzie klient używa do konwersji docelowego kodowanego zestawu znaków:

```
bytes.writeBytes("In the destination code page".getBytes(
CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

Alternatywnie klient mógł wybrać stronę kodową, a następnie ustawić odpowiedni kodowany zestaw znaków we właściwości JMS_IBM_CHARACTER_SET komunikatu. IBM MQ classes for Java

Należy użyć parametru JMS_IBM_CHARACTER_SET , aby ustawić pole CodedCharacterSetId we właściwościach JMS w pliku MQRFH2lub w deskryptorze komunikatu MQMD:

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage);
5
```

Jeśli tablica bajtów jest zapisywana do systemu JMSStringMessage lub JMSMapMessage, system IBM MQ classes for JMS nie przeprowadza konwersji danych, ponieważ bajty są wpisywane jako dane szesnastkowe, a nie jako tekst w JMSStringMessage i JMSMapMessage.

Jeśli bajty reprezentują znaki w aplikacji, należy wziąć pod uwagę punkty kodowe, które mają być odczytane i zapisane w komunikacie. Kod w pliku Rysunek 26 na stronie 189 jest zgodny z konwencją używania kodowanego zestawu znaków miejsca docelowego. Jeśli łańcuch zostanie utworzony przy użyciu domyślnego zestawu znaków dla maszyny JVM, zawartość bajtu zależy od platformy. Maszyna JVM w systemie Windows ma zwykle domyślny Charset windows-1252, a AIX and Linux ma UTF-8. W przypadku wymiany między systemami Windowsi AIX and Linuxnależy wybrać jawną stronę kodową dla wymiany tekstu jako bajtów.

```
StreamMessage smo = producer.session.createStreamMessage();
smo.writeBytes("123".getBytes(CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

*Rysunek 26. Zapisywanie bajtów reprezentujących łańcuch w JMSStreamMessage przy użyciu docelowego zestawu znaków*

**Przykłady**

### Wysyłanie i odbieranie JMSTextmessage

Komunikat tekstowy nie może zawierać tekstu w różnych zestawach znaków. Przykład przedstawia tekst w różnych zestawach znaków, wysłany w dwóch różnych komunikatach.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

*Rysunek 27. Wyślij komunikat tekstowy w zestawie znaków zdefiniowanym przez miejsce docelowe*

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

*Rysunek 28. Wyślij wiadomość tekstową w ccsid 37*

[5] SetStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage) currently accepts only numeric character set identifiers.

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

*Rysunek 29. Odbierz komunikat tekstowy*

### Przykłady kodowania

Przykłady przedstawiające liczbę wysyłaną w kodowaniu definiują miejsce docelowe. Należy zauważyć, że dla właściwości JMS_IBM_ENCODING JMSBytesMessage należy ustawić wartość określoną dla miejsca docelowego.

```
StreamMessage smo = session.createStreamMessage();
smo.writeInt(256);
producer.send(smo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println(smi.readInt());
...
256
```

*Rysunek 30. Wysyłanie liczby przy użyciu kodowania docelowego w JMSStreamMessage*

```
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
```

*Rysunek 31. Wysyłanie liczby przy użyciu kodowania docelowego w JMSBytesMessage*

### Wysyłanie i odbieranie tekstu w JMSBytesMessage

Kod w pliku Rysunek 32 na stronie 190 wysyła łańcuch w komunikacie BytesMessage. Dla uproszczenia przykład wysyła pojedynczy łańcuch, dla którego bardziej odpowiedni jest łańcuch JMSTextMessage . Aby otrzymać łańcuch tekstowy w bajtach, zawierający mieszaninę typów, należy znać długość łańcucha w bajtach, o nazwie *TEXT_LENGTH* w Rysunek 33 na stronie 191. Nawet w przypadku łańcucha o stałej liczbie znaków długość reprezentacji bajtowej może być dłuższa.

```
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);
```

*Rysunek 32. Wysyłanie String w JMSBytesMessage*

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

*Rysunek 33. Odbieranie `String` z `JMSBytesMessage`*

**Pojęcia pokrewne**

Podejścia do konwersji komunikatów JMS
Wiele podejść do konwersji danych jest otwartych dla projektantów aplikacji JMS . Te podejścia nie są wyłączne; niektóre aplikacje mogą korzystać z ich kombinacji. Jeśli aplikacja wymienia tylko tekst lub komunikaty tylko z innymi aplikacjami JMS , zwykle nie jest rozważana konwersja danych. Konwersja danych jest wykonywana automatycznie przez IBM MQ.

Konwersja danych menedżera kolejek
Konwersja danych menedżera kolejek była zawsze dostępna dla aplikacji innych niż aplikacjeJMS , które odbierały komunikaty od klientów JMS . Klienty JMS odbierające komunikaty również używają opcjonalnej konwersji danych menedżera kolejek.

**Zadania pokrewne**

Wymiana sformatowanego rekordu z aplikacją inną niżJMS
Wykonaj kroki sugerowane w tym zadaniu, aby zaprojektować i zbudować wyjście konwersji danych oraz aplikację kliencką JMS , która może wymieniać komunikaty z aplikacją inną niżJMS przy użyciu języka `JMSBytesMessage`. Wymiana sformatowanego komunikatu z aplikacją inną niżJMS może odbywać się z wyjściem konwersji danych lub bez niego.

**Odsyłacze pokrewne**

Typy i konwersja komunikatów JMS
Wybór typu komunikatu ma wpływ na podejście do konwersji komunikatów. Interakcja konwersji komunikatu i typu komunikatu jest opisana dla typów komunikatów JMS , `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage`i `JMSBytesMessage`.

*Konwersja danych menedżera kolejek*
Konwersja danych menedżera kolejek była zawsze dostępna dla aplikacji innych niż aplikacjeJMS , które odbierały komunikaty od klientów JMS . Klienty JMS odbierające komunikaty również używają opcjonalnej konwersji danych menedżera kolejek.

Menedżer kolejek może dokonywać konwersji danych znakowych i liczbowych w danych komunikatu przy użyciu wartości `CodedCharacterSetId`, `Encoding`i `Format` ustawionych dla danych komunikatu. W przypadku aplikacji innych niż aplikacjeJMS możliwość konwersji była zawsze dostępna po ustawieniu opcji `GetMessage`( `GMO_CONVERT`).

Menedżer kolejek może przekształcać komunikaty wysyłane do klientów JMS . Konwersją menedżera kolejek steruje się, ustawiając właściwość miejsca docelowego `WMQ_RECEIVE_CONVERSION`na wartość `WMQ_RECEIVE_CONVERSION_QMGR`lub `WMQ_RECEIVE_CONVERSION_CLIENT_MSG`. Aplikacja może zmienić ustawienie miejsca docelowego:

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Lub,

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

*Rysunek 34. Włącz konwersję danych menedżera kolejek*

Konwersja danych menedżera kolejek dla klienta JMS jest wykonywana, gdy klient wywołuje metodę `consumer.receive` . Domyślnie dane tekstowe są transformowane do formatu UTF-8 (1208). Kolejne metody read i get dekodują tekst w odebranych danych z UTF-8, tworząc operacje podstawowe tekstu Java w wewnętrznym kodowaniu Unicode. UTF-8 nie jest jedynym docelowym zestawem znaków z konwersji danych menedżera kolejek. Aby wybrać inny identyfikator CCSID, należy ustawić właściwość miejsca docelowego `WMQ_RECEIVE_CCSID` .

Aplikacja może również zmienić ustawienie miejsca docelowego, na przykład ustawić go na 437, DOS-US:

```
((MQDestination)destination).setIntProperty
(WMQConstants.WMQ_RECEIVE_CCSID, 437);
```

Lub,

```
((MQDestination)destination).setReceiveCCSID(437);
```

*Rysunek 35. Ustaw docelowy kodowany zestaw znaków dla konwersji menedżera kolejek*

Przyczyna zmiany wartości zmiennej `WMQ_RECEIVE_CCSID` jest wyspecjalizowana. Wybrany identyfikator CCSID nie ma znaczenia dla obiektów tekstowych tworzonych w maszynie JVM. Jednak niektóre maszyny JVM na niektórych platformach mogą nie być w stanie obsłużyć konwersji z identyfikatora CCSID tekstu w komunikacie do kodu Unicode. Ta opcja umożliwia wybór identyfikatora CCSID dla każdego tekstu dostarczonego do klienta w komunikacie. Na niektórych platformach klienckich systemu JMS wystąpiły problemy z dostarczaniem tekstu komunikatu w kodowaniu UTF-8.

Kod JMS jest odpowiednikiem tekstu pogrubionego w kodzie C w języku Rysunek 36 na stronie 193,

```
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                     */
```

*Rysunek 36. Fragment kodu z pliku amqsget0.c*

**Uwaga:**

Konwersja menedżera kolejek jest wykonywana tylko na danych komunikatu, które mają znany format IBM MQ . MQSTR lub MQCIH są przykładami znanych formatów, które są predefiniowane. Znanym formatem może być również format zdefiniowany przez użytkownika, o ile podano wyjście konwersji danych.

Komunikaty utworzone jako JMSTextMessage, JMSMapMessage i JMSStreamMessagemają format MQSTR i mogą być przekształcane przez menedżer kolejek.

**Pojęcia pokrewne**

Podejścia do konwersji komunikatów JMS
Wiele podejść do konwersji danych jest otwartych dla projektantów aplikacji JMS . Te podejścia nie są wyłączne; niektóre aplikacje mogą korzystać z ich kombinacji. Jeśli aplikacja wymienia tylko tekst lub komunikaty tylko z innymi aplikacjami JMS , zwykle nie jest rozważana konwersja danych. Konwersja danych jest wykonywana automatycznie przez IBM MQ.

Konwersja i kodowanie komunikatów klienta JMS
Metody używane do przeprowadzania konwersji i kodowania komunikatów klienta JMS zostały wymienione na liście wraz z przykładami kodu dla każdego typu konwersji.

“Wywoływanie wyjścia konwersji danych” na stronie 1014
Wyjście konwersji danych to zapisane przez użytkownika wyjście, które odbiera sterowanie podczas przetwarzania wywołania MQGET.

**Zadania pokrewne**

Wymiana sformatowanego rekordu z aplikacją inną niżJMS
Wykonaj kroki sugerowane w tym zadaniu, aby zaprojektować i zbudować wyjście konwersji danych oraz aplikację kliencką JMS , która może wymieniać komunikaty z aplikacją inną niżJMS przy użyciu języka JMSBytesMessage. Wymiana sformatowanego komunikatu z aplikacją inną niżJMS może odbywać się z wyjściem konwersji danych lub bez niego.

**Odsyłacze pokrewne**

Typy i konwersja komunikatów JMS
Wybór typu komunikatu ma wpływ na podejście do konwersji komunikatów. Interakcja konwersji komunikatu i typu komunikatu jest opisana dla typów komunikatów JMS , JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessagei JMSBytesMessage.

*Wymiana sformatowanego rekordu z aplikacją inną niżJMS*
Wykonaj kroki sugerowane w tym zadaniu, aby zaprojektować i zbudować wyjście konwersji danych oraz aplikację kliencką JMS , która może wymieniać komunikaty z aplikacją inną niżJMS przy użyciu języka

JMSBytesMessage. Wymiana sformatowanego komunikatu z aplikacją inną niżJMS może odbywać się z wyjściem konwersji danych lub bez niego.

## Zanim rozpoczniesz

Może być możliwe zaprojektowanie prostszego rozwiązania do wymiany komunikatów z aplikacją inną niżJMS za pomocą JMSTextMessage. Wyeliminuj tę możliwość przed wykonaniem kroków tego zadania.

## O tym zadaniu

Klient JMS jest łatwiejszy w pisaniu, jeśli nie jest zaangażowany w szczegóły formatowania komunikatów JMS wymienianych z innymi klientami JMS . Tak długo, jak typem komunikatu jest JMSTextMessage, JMSMapMessage, JMSStreamMessagelub JMSObjectMessage, IBM MQ będzie szukać szczegółów formatowania komunikatu. Produkt IBM MQ obsługuje różnice w kodowaniu stron kodowych i kodowania liczbowego na różnych platformach.

Tych typów komunikatów można używać do wymiany komunikatów z aplikacjami innymi niżJMS . W tym celu należy zrozumieć, w jaki sposób te komunikaty są konstruowane przez produkt IBM MQ classes for JMS. Użytkownik może mieć możliwość zmodyfikowania aplikacji innej niżJMS w celu interpretowania komunikatów; patrz sekcja “Odwzorowywanie komunikatów JMS na komunikaty IBM MQ” na stronie 153.

Zaletą używania jednego z tych typów komunikatów jest to, że programowanie klienta JMS nie zależy od typu aplikacji, z którą jest wymieniany komunikat. Wadą jest to, że może wymagać modyfikacji innego programu, a użytkownik może nie być w stanie zmienić innego programu.

Alternatywnym podejściem jest napisanie aplikacji klienckiej JMS , która może zajmować się istniejącymi formatami komunikatów. Często istniejące komunikaty mają stały format i zawierają kombinację niesformatowanych danych, tekstu i liczb. Wykonaj czynności opisane w tym zadaniu oraz w przykładowym kliencie JMS w sekcji “Pisanie klas w celu hermetyzowania układu rekordów w JMSBytesMessage” na stronie 197, aby rozpocząć budowanie klienta JMS , który może wymieniać sformatowane rekordy z aplikacjami innymi niżJMS .

## Procedura

1. Zdefiniuj układ rekordu lub użyj jednej z predefiniowanych klas nagłówka IBM MQ .

   Informacje na temat obsługi predefiniowanych nagłówków IBM MQ zawiera sekcja Obsługa nagłówków komunikatów produktu IBM MQ.

   Rysunek 37 na stronie 195 jest przykładem zdefiniowanego przez użytkownika układu rekordów o stałej długości, który może być przetwarzany przez program narzędziowy do konwersji danych.
2. Utwórz wyjście konwersji danych.

   Postępuj zgodnie z instrukcjami w sekcji Pisanie programu obsługi wyjścia konwersji danych , aby zapisać program obsługi wyjścia konwersji danych.

   Aby wypróbować przykład w sekcji “Pisanie klas w celu hermetyzowania układu rekordów w JMSBytesMessage” na stronie 197, nadaj nazwę wyjściu konwersji danych MYRECORD.
3. Napisz klasy Java , aby hermetyzować układ rekordów oraz wysyłać i odbierać rekordy. Dwa podejścia, które można zastosować, to:

   - Napisz klasę, która odczytuje i zapisuje plik JMSBytesMessage , który zawiera rekord; patrz “Pisanie klas w celu hermetyzowania układu rekordów w JMSBytesMessage” na stronie 197.
   - Napisz klasę rozszerzającą klasę com.ibm.mq.header.Header , aby zdefiniować strukturę danych rekordu; patrz sekcja Tworzenie klas dla nowych typów nagłówków.
4. Zdecyduj, w którym kodowanym zestawie znaków mają być wymieniane komunikaty.

   Więcej informacji na ten temat zawiera sekcja Wybór podejścia do konwersji komunikatów: odbiorca jest dobry.
5. Skonfiguruj miejsce docelowe na potrzeby wymiany komunikatów typu MQbez nagłówka JMS MQRFH2 .

Zarówno miejsce docelowe wysyłania, jak i odbierania musi być skonfigurowane do wymiany komunikatów typu MQ. Można użyć tego samego miejsca docelowego zarówno dla wysyłania, jak i odbierania.

Aplikacja może przesłonić właściwość treści komunikatu docelowego:

```
((MQDestination)destination).setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

Przykład, który przedstawia “Pisanie klas w celu hermetyzowania układu rekordów w JMSBytesMessage” na stronie 197 , nadpisuje właściwość treści komunikatu miejsca docelowego, zapewniając wysyłanie komunikatu w stylu produktu MQ.

6. Testowanie rozwiązania z aplikacjami JMS i aplikacjami innymi niżJMS

   Przydatne narzędzia do testowania wyjścia konwersji danych to:

   - Przykładowy program `amqsgetc0.c` jest przydatny do testowania odbierania komunikatu wysłanego przez klienta JMS . Zapoznaj się z sugerowanymi modyfikacjami, aby użyć przykładowego nagłówka `RECORD.hw` pliku Rysunek 38 na stronie 196. Po wprowadzeniu modyfikacji program `amqsgetc0.c` odbiera komunikat wysłany przez przykładowego klienta JMS , `TryMyRecord.java` ; zawiera sekcja “Pisanie klas w celu hermetyzowania układu rekordów w JMSBytesMessage” na stronie 197.
   - Przykładowy IBM MQ program przeglądania `amqsbcg0.c`jest przydatny do sprawdzania treści nagłówka komunikatu, nagłówka JMS , `MQRFH2`i treści komunikatu.
   - Program **`rfhutil`** , poprzednio dostępny w pakiecie SupportPac IH03, umożliwia przechwytywanie i zapisywanie komunikatów testowych w plikach, a następnie używanie ich do sterowania przepływami komunikatów. Komunikaty wyjściowe mogą być również odczytywane i wyświetlane w różnych formatach. Formaty obejmują dwa typy XML, a także dopasowanie do struktury copybook języka COBOL. Dane mogą być w kodzie EBCDIC lub ASCII. Nagłówek RFH2 można dodać do komunikatu przed wysłaniem komunikatu.

   W przypadku próby odebrania komunikatów za pomocą zmodyfikowanego przykładowego programu `amqsgetc0.c` i wystąpienia błędu o kodzie przyczyny `2080`, należy sprawdzić, czy komunikat zawiera kod `MQRFH2`. W modyfikacjach założono, że komunikat został wysłany do miejsca docelowego, które nie określa `MQRFH2`.

**Przykłady**

```
struct RECORD { MQCHAR StrucID[4];
                MQLONG Version;
                               MQLONG StructLength;
                               MQLONG Encoding;
                               MQLONG CodeCharSetId;
                               MQCHAR Format[8];
                               MQLONG Flags;
                               MQCHAR RecordData[32];
        };
```

*Rysunek 37. RECORD.h*

- Zadeklaruj strukturę danych RECORD.h

```
struct tagRECORD {
   MQCHAR4    StrucId;
   MQLONG     Version;
   MQLONG     StrucLength;
   MQLONG     Encoding;
   MQLONG     CCSID;
   MQCHAR8    Format;
   MQLONG     Flags;
   MQCHAR32   RecordData;
 };
typedef struct tagRECORD RECORD;
   typedef RECORD MQPOINTER PRECORD;
   RECORD record;
   PRECORD pRecord = &(record);
```

- Zmodyfikuj wywołanie MQGET w taki sposób, aby używała RECORD, .
  1. Przed modyfikacją:

```
    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. Po modyfikacji:

```
    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           */
```

- Zmień instrukcję drukowania,
  1. Od:

```
buffer[messlen] = '\0';            /* add terminator          */
printf("message <%s>\n", buffer);
```

  2. Do:

```
/* buffer[messlen] = '\0';            add terminator          */
printf("ccsid <%d>, flags <%d>, message <%32.32s>\n \0",
        md.CodedCharSetId, record.Flags, record.RecordData);
```

*Rysunek 38. Zmodyfikuj plik amqsget0.c .*

**Pojęcia pokrewne**

Podejścia do konwersji komunikatów JMS

Wiele podejść do konwersji danych jest otwartych dla projektantów aplikacji JMS . Te podejścia nie są wyłączne; niektóre aplikacje mogą korzystać z ich kombinacji. Jeśli aplikacja wymienia tylko tekst lub komunikaty tylko z innymi aplikacjami JMS , zwykle nie jest rozważana konwersja danych. Konwersja danych jest wykonywana automatycznie przez IBM MQ.

Konwersja i kodowanie komunikatów klienta JMS

Metody używane do przeprowadzania konwersji i kodowania komunikatów klienta JMS zostały wymienione na liście wraz z przykładami kodu dla każdego typu konwersji.

Konwersja danych menedżera kolejek
Konwersja danych menedżera kolejek była zawsze dostępna dla aplikacji innych niż aplikacjeJMS , które odbierały komunikaty od klientów JMS . Klienty JMS odbierające komunikaty również używają opcjonalnej konwersji danych menedżera kolejek.

Program narzędziowy do tworzenia kodu wyjścia konwersji

**Odsyłacze pokrewne**

Typy i konwersja komunikatów JMS
Wybór typu komunikatu ma wpływ na podejście do konwersji komunikatów. Interakcja konwersji komunikatu i typu komunikatu jest opisana dla typów komunikatów JMS , `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage`i `JMSBytesMessage`.

*Pisanie klas w celu hermetyzowania układu rekordów w JMSBytesMessage*
Celem tego zadania jest zbadanie, na przykład, w jaki sposób połączyć konwersję danych i stały układ rekordu w `JMSBytesMessage`. W zadaniu tworzone są klasy Java w celu wymiany przykładowej struktury rekordu w `JMSBytesMessage`. Przykład można zmodyfikować, aby zapisać klasy w celu wymiany innych struktur rekordów.

`JMSBytesMessage` to najlepszy wybór typu komunikatu JMS do wymiany rekordów mieszanego typu danych z programami innymi niżJMS . Nie zawiera on dodatkowych danych wstawianych do treści komunikatu przez dostawcę JMS . Dlatego jest to najlepszy wybór typu komunikatu, jeśli program kliencki JMS współdziała z istniejącym programem IBM MQ . Główne wyzwanie związane z używaniem `JMSBytesMessage` jest związane z dopasowaniem kodowania i zestawu znaków oczekiwanego przez inny program. Rozwiązaniem jest utworzenie klasy, która obudowuje rekord. Klasa, która hermetyzuje odczyt i zapis `JMSBytesMessage`dla konkretnego typu rekordu, ułatwia wysyłanie i odbieranie rekordów o stałym formacie w programie JMS . Przechwytując ogólne aspekty interfejsu w klasie abstrakcyjnej, można wykorzystać większość rozwiązania dla różnych formatów rekordów. W klasach, które rozszerzają abstrakcyjną klasę ogólną, można zaimplementować różne formaty rekordów.

Alternatywnym podejściem jest rozszerzenie klasy `com.ibm.mq.headers.Header` . Klasa `Header` zawiera metody, takie jak `addMQLONG`, służące do budowania formatu rekordu w bardziej deklaratywny sposób. Wadą korzystania z klasy `Header` jest uzyskiwanie i ustawianie atrybutów przy użyciu bardziej skomplikowanego interfejsu interpretacyjnego. Oba podejścia skutkują znacznie taką samą ilością kodu aplikacji.

`JMSBytesMessage` może hermetyzować tylko jeden format, oprócz `MQRFH2`, w jednym komunikacie, chyba że każdy rekord używa tego samego formatu, kodowanego zestawu znaków i kodowania. Format, kodowanie i zestaw znaków `JMSBytesMessage` są właściwościami wszystkich komunikatów następujących po komunikacie `MQRFH2`. Przykład został napisany przy założeniu, że `JMSBytesMessage` zawiera tylko jeden rekord użytkownika.

## Zanim rozpoczniesz

1. Poziom umiejętności: użytkownik musi być zaznajomiony z programowaniem Java i JMS.
   Nie udostępniono żadnych instrukcji dotyczących konfigurowania środowiska programistycznego Java . Korzystne jest napisanie programu wymiany `JMSTextMessage`, `JMSStreamMessage`lub `JMSMapMessage`. Następnie można zobaczyć różnice w wymianie komunikatu przy użyciu `JMSBytesMessage`.
2. W tym przykładzie wymagany jest parametr IBM WebSphere MQ 7.0.
3. Przykład został utworzony przy użyciu perspektywy Java środowiska roboczego Eclipse . Wymaga środowiska JRE 6.0 lub nowszego. Za pomocą perspektywy Java w Eksploratorze IBM MQ można tworzyć i uruchamiać klasy języka Java . Alternatywnie można użyć własnego środowiska programistycznego Java .
4. Korzystanie z Eksploratora IBM MQ sprawia, że konfigurowanie środowiska testowego i debugowanie jest prostsze niż korzystanie z programów narzędziowych wiersza komend.

## O tym zadaniu

Użytkownik prowadzi użytkownika przez proces tworzenia dwóch klas: RECORD i MyRecord. Te dwie klasy łącznie obudowują rekord o ustalonym formacie. Mają metody do pobierania i ustawiania atrybutów. Metoda get odczytuje rekord z JMSBytesMessage , a metoda put zapisuje rekord w JMSBytesMessage.

Celem tego zadania nie jest utworzenie produkcyjnej klasy jakości, którą można ponownie wykorzystać. Można użyć przykładów w zadaniu, aby rozpocząć pracę z własnymi klasami. Celem tego zadania jest udostępnienie wskazówek dotyczących przede wszystkim używania zestawów znaków, formatów i kodowania podczas korzystania z JMSBytesMessage. Każdy krok tworzenia klas jest wyjaśniony i opisano aspekty korzystania z produktu JMSBytesMessage, które czasami są pomijane.

Klasa RECORD jest abstrakcyjna i definiuje niektóre wspólne pola dla rekordu użytkownika. Wspólne pola są modelowane na podstawie standardowego układu nagłówka IBM MQ , w którym znajduje się element przechwytujący oko, wersja i pole długości. Pola kodowania, zestawu znaków i formatu znajdujące się w wielu nagłówkach IBM MQ są pomijane. Inny nagłówek nie może być zgodny z formatem zdefiniowanym przez użytkownika. Klasa MyRecord , która rozszerza klasę RECORD , robi to poprzez dosłowne rozszerzenie rekordu o dodatkowe pola użytkownika. Program JMSBytesMessageutworzony przez klasy może być przetwarzany przez wyjście konwersji danych menedżera kolejek.

“Klasy używane do uruchamiania przykładu” na stronie 204 zawiera pełną listę produktów RECORD i MyRecord. Zawiera również listę dodatkowych klas "rusztowania" w celu przetestowania klas RECORD i MyRecord. Dodatkowe klasy to:

**TryMyRecord**
Główny program do testowania produktów RECORD i MyRecord.

**EndPoint**
Klasa abstrakcyjna, która hermetyzuje połączenie JMS , miejsce docelowe i sesję w pojedynczej klasie. Jego interfejs spełnia tylko wymagania dotyczące testowania klas RECORD i MyRecord . Nie jest to ustanowiony wzorzec projektu na potrzeby pisania aplikacji JMS .

**Uwaga:** Klasa Endpoint zawiera następujący wiersz kodu po utworzeniu miejsca docelowego:

```
((MQDestination)destination).setReceiveConversion
                            (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

W produkcie V7.0z wersji V7.0.1.5konieczne jest włączenie konwersji menedżera kolejek. Jest ona domyślnie wyłączona. W produkcie V7.0konwersja menedżera kolejek do wersji V7.0.1.4 jest domyślnie włączona, a ten wiersz kodu powoduje błąd.

**MyProducer i MyConsumer**
Klasy, które rozszerzają klasę EndPointi tworzą klasy MessageConsumer i MessageProducer, połączone i gotowe do akceptowania żądań.

Razem wszystkie klasy tworzą kompletną aplikację, którą można zbudować i z którą można eksperymentować, aby zrozumieć, w jaki sposób używać konwersji danych w JMSBytesMessage.

## Procedura

1. Utwórz klasę abstrakcyjną, aby hermetyzować pola standardowe w nagłówku IBM MQ przy użyciu konstruktora domyślnego. Następnie należy rozszerzyć klasę, aby dostosować nagłówek do swoich wymagań.

```
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();
    }
```

**Uwaga:**

a. Atrybuty `structID` do `nextFormat`są wyświetlane w kolejności, w jakiej są umieszczone w standardowym nagłówku komunikatu IBM MQ .

b. Atrybuty `format`, `messageEncoding`i `messageCharset`opisują sam nagłówek i nie są częścią nagłówka.

c. Należy zdecydować, czy zapisać identyfikator kodowanego zestawu znaków, czy zestaw znaków rekordu. System Java używa zestawów znaków, a komunikaty systemu IBM MQ używają identyfikatorów kodowanego zestawu znaków. W przykładowym kodzie używane są zestawy znaków.

d. Łańcuch `int` jest przekształcany do postaci szeregowej `MQLONG` przez program IBM MQ. `MQLONG` to 4 bajty.

2. Utwórz procedury pobierające i ustawiające dla atrybutów prywatnych.

a) Utwórz lub wygeneruj procedury pobierające:

```
     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) Utwórz lub wygeneruj procedury ustawiające:

```
     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. Utwórz konstruktor, aby utworzyć instancję `RECORD` na podstawie klasy `JMSBytesMessage`.

```
  public RECORD(BytesMessage message) throws JMSException, IOException,
         MQDataException {
     super();
     setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
     setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
     byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
     message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
     setStructID(new String(structID, getMessageCharset()));
     setVersion(message.readInt());
     setStructLength(message.readInt());
 }
```

**Uwaga:**

a. Wartości `messageCharset` i `messageEncoding`są przechwytywane z właściwości komunikatu, ponieważ nadpisują wartości ustawione dla miejsca docelowego. Plik `format` nie jest aktualizowany. Przykład nie sprawdza błędów. Jeśli zostanie wywołany konstruktor `Record(BytesMessage)` , przyjmuje się, że `JMSBytesMessage` jest komunikatem typu `RECORD` . Linia "`setStructID(new String(structID, getMessageCharset()))`" ustawia przyciągacz oka.

b. Wiersze kodu, które wypełniają pola w komunikacie przekształcające metodę z postaci szeregowej, aktualizują wartości domyślne ustawione w instancji RECORD.

4. Utwórz metodę put, aby zapisać pola nagłówka w `JMSBytesMessage`.

```
   protected BytesMessage put(MyProducer myProducer) throws IOException,
         JMSException, UnsupportedEncodingException {
      setHeaderEncoding(myProducer.getEncoding());
      setHeaderCharset(myProducer.getCharset());
      myProducer.setMQClient(true);
      BytesMessage bytes = myProducer.session.createBytesMessage();
      bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
      bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
      bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                           myProducer.getCCSID());
      bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                       + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                       .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
      bytes.writeInt(getVersion());
      bytes.writeInt(getStructLength());
      return bytes;
   }
```

**Uwaga:**

a. `MyProducer` obudowuje JMS `Connection`, `Destination`, `Session`i `MessageProducer` w pojedynczej klasie. MyConsumer, używany później, hermetyzuje JMS `Connection`, `Destination`, `Session`i `MessageConsumer` w pojedynczej klasie.

b. W przypadku systemu `JMSBytesMessage`, jeśli kodowanie jest inne niż `Native`, kodowanie musi zostać ustawione w komunikacie. Kodowanie docelowe jest kopiowane do atrybutu kodowania komunikatów `JMS_IBM_CHARACTER_SET`i zapisywane jako atrybut klasy `RECORD` .

   i) "`setMessageEncoding(myProducer.getEncoding());`" wywołuje "`(((MQDestination) destination).getIntProperty(WMQConstants.WMQ_ENCODING));`" w celu pobrania kodowania miejsca docelowego.

   ii) "`Bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getMessageEncoding());`" ustawia kodowanie komunikatu.

c. Zestaw znaków używany do przekształcania tekstu w bajty jest uzyskiwany z miejsca docelowego i zapisywany jako atrybut klasy `RECORD` . Nie jest ona ustawiona w komunikacie, ponieważ nie jest używana przez IBM MQ classes for JMS podczas zapisywania `JMSBytesMessage`.

   Wywołania "`messageCharset = myProducer.getCharset();`"

```
   public String getCharset() throws UnsupportedEncodingException,
          JMSException {
       return CCSID.getCodepage(getCCSID());
}
```

   Pobiera ona zestaw znaków Java z identyfikatora kodowanego zestawu znaków.

   " `CCSID.getCodepage(ccsid)` " znajduje się w pakiecie `com.ibm.mq.headers`. Wartość `ccsid` jest uzyskiwana z innej metody w pliku `MyProducer`, która wysyła zapytanie do miejsca docelowego:

```
   public int getCCSID() throws JMSException {
       return (((MQDestination) destination)
```

```
                .getIntProperty(WMQConstants.WMQ_CCSID));
    }
```

d. "`myProducer.setMQClient(true);`" nadpisuje ustawienie miejsca docelowego dla typu klienta, wymuszając ustawienie IBM MQ MQI client. Można pominąć ten wiersz kodu, ponieważ przesłania on błąd konfiguracji administracyjnej.

Wywołania programu "`myProducer.setMQClient(true);`":

```
((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ); }
if (!getMQDest()) setMQBody();
```

Efektem ubocznym działania kodu jest ustawienie stylu treści IBM MQ na nieokreślony, jeśli musi on przesłonić ustawienie JMS.

**Uwaga:**

IBM MQ classes for JMS zapisuje format, kodowanie i identyfikator zestawu znaków komunikatu w deskryptorze komunikatu `MQMDlub` w nagłówku JMS `MQRFH2`. Zależy to od tego, czy komunikat ma treść w stylu IBM MQ . Nie należy ręcznie ustawiać pól `MQMD` .

Istnieje metoda ręcznego ustawiania właściwości deskryptora komunikatu. Używa właściwości `JMS_IBM_MQMD_*` . Aby ustawić właściwości `JMS_IBM_MQMD_*` , należy ustawić właściwość docelową `WMQ_MQMD_WRITE_ENABLED` :

```
((MQDestination)destination).setMQMDWriteEnabled(true);
```

Aby odczytać właściwości, należy ustawić właściwość docelową `WMQ_MQMD_READ_ENABLED`.

Parametru `JMS_IBM_MQMD_*` należy używać tylko wtedy, gdy użytkownik przejmie pełną kontrolę nad całym ładunkiem komunikatu. W przeciwieństwie do właściwości `JMS_IBM_*` , właściwości `JMS_IBM_MQMD_*` nie sterują sposobem, w jaki IBM MQ classes for JMS tworzy komunikat JMS . Możliwe jest utworzenie właściwości deskryptora komunikatu, które powodują konflikt z właściwościami komunikatu JMS .

e. Wiersze kodu, które uzupełniają metodę serializują atrybuty w klasie jako pola w komunikacie.

Atrybuty łańcuchowe są dopełniane odstępami. Łańcuchy są przekształcane w bajty przy użyciu zestawu znaków zdefiniowanego dla rekordu i obcinane do długości pól komunikatu.

5. Zakończ klasę, dodając importy.

```
 package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
```

6. Utwórz klasę w celu rozszerzenia klasy RECORD o dodatkowe pola. Dołącz konstruktor domyślny.

```
public class MyRecord extends RECORD {
   private static final long serialVersionUID = -370551723162299429L;
   private final static int FLAGS = 1;
   private final static String STRUCT_ID = "MYRD";
   private final static int DATA_LENGTH = 32;
   private final static String FORMAT = "MYRECORD";
   private int flags = FLAGS;
   private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

   public MyRecord() {
       super();
       super.setStructID(STRUCT_ID);
       super.setHeaderFormat(FORMAT);
       super.setStructLength(super.getStructLength() + MQLONG_LENGTH
```

```
                + DATA_LENGTH);
    }
```

**Uwaga:**

a. Podklasa RECORD , MyRecord, dostosowuje element przechwytujący oko, format i długość nagłówka.

7. Utwórz lub wygeneruj procedury pobierające i ustawiające.

   a) Utwórz procedury pobierające:

```
     public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .
```

   b) Utwórz procedury ustawiające:

```
     public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

8. Utwórz konstruktor, aby utworzyć instancję MyRecord na podstawie klasy JMSBytesMessage.

```
   public MyRecord(BytesMessage message) throws JMSException, IOException,
          MQDataException {
      super(message);
      setFlags(message.readInt());
      byte[] recordData = new byte[DATA_LENGTH];
      message.readBytes(recordData, DATA_LENGTH);
      setRecordData(new String(recordData, super.getMessageCharset()));
   }
```

**Uwaga:**

a. Pola, które tworzą standardowy szablon komunikatu, są odczytywane jako pierwsze przez klasę RECORD .

b. Tekst recordData jest przekształcany w tekst String przy użyciu właściwości zestawu znaków komunikatu.

9. Utwórz metodę statyczną, aby pobrać komunikat od konsumenta i utworzyć nową instancję MyRecord .

```
   public static MyRecord get(MyConsumer myConsumer) throws JMSException,
          MQDataException, IOException {
      BytesMessage message = (BytesMessage) myConsumer.receive();
      return new MyRecord(message);
   }
```

**Uwaga:**

a. W przykładzie, w celu zwięzłości, konstruktor MyRecord(BytesMessage) jest wywoływany ze statycznej metody get. Zwykle użytkownik może oddzielić komunikat od tworzenia nowej instancji MyRecord .

10. Utwórz metodę put, aby dodać pola klienta do pliku JMSBytesMessage zawierającego nagłówek komunikatu.

```
   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;
    }
```

**Uwaga:**

a. Wywołania metody w kodzie przekształcają atrybuty w klasie MyRecord do postaci szeregowej jako pola w komunikacie.

- Atrybut `recordData` `String` jest dopełniany odstępami, przekształcany w bajty przy użyciu zestawu znaków zdefiniowanego dla rekordu i obcinany do długości pól `RecordData` .

11. Zakończ klasę, dodając instrukcje include.

```
 package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
```

## Wyniki

- Wyniki działania klasy `TryMyRecord` :
  - Wysyłanie komunikatu w kodowanym zestawie znaków 37 przy użyciu wyjścia konwersji menedżera kolejek:

```
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
```

  - Wysyłanie komunikatu w kodowanym zestawie znaków 37 bez użycia wyjścia konwersji menedżera kolejek:

```
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
```

- Są to wyniki modyfikowania klasy `TryMyRecord` , które nie odbierają komunikatu, a zamiast tego są odbierane przy użyciu zmodyfikowanego przykładu `amqsget0.c` . Zmodyfikowany przykład akceptuje sformatowany rekord; patrz Rysunek 38 na stronie 196 w sekcji “Wymiana sformatowanego rekordu z aplikacją inną niżJMS” na stronie 193.
  - Wysyłanie komunikatu w kodowanym zestawie znaków 37 przy użyciu wyjścia konwersji menedżera kolejek:

```
Sample AMQSGET0 start
ccsid <850>, flags <1>, message <ABCDEFGHIJKLMNOPQRSTUVWXYZ012345>
no more messages
Sample AMQSGET0 end
```

  - Wysyłanie komunikatu w kodowanym zestawie znaków 37 bez użycia wyjścia konwersji menedżera kolejek:

```
Sample AMQSGET0 start
MQGET ended with reason code 2110
ccsid <37>, flags <1>, message <--+-+ãÃ++ÐÊËÈiÐÎÐ+ÔÒõõµþÞÚ-±=¾¶§>
no more messages
Sample AMQSGET0 end
```

Aby wypróbować przykład i eksperymentować z różnymi stronami kodowymi i wyjściem konwersji danych. Utwórz klasy Java , skonfiguruj IBM MQi uruchom program główny, `TryMyRecord` ; zawiera sekcja “#unique_196/unique_196_Connect_42_Try” na stronie 204.

1. Skonfiguruj IBM MQ i JMS , aby uruchomić przykład. Instrukcje dotyczą uruchamiania przykładu w systemie Windows.
   a. Tworzenie menedżera kolejek

   ```
   crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
   strmqm QM1
   ```

   b. Utwórz kolejkę

   ```
   echo DEFINE QL('Q1') REPLACE | runmqsc QM1
   ```

   c. Utwórz katalog JNDI

   ```
   cd c:\
   md JNDI-Directory
   ```

   d. Przejdź do katalogu JMS bin

   Z tego miejsca należy uruchomić program JMS Administration. Ścieżka to *MQ_INSTALLATION_PATH*`\java\bin`.

   e. Utwórz następujące definicje JMS w pliku o nazwie `JMSQM1Q1.txt`

   ```
   DEF CF(QM1) PROVIDERVERSION(7) QMANAGER(QM1)
   DEF Q(Q1) CCSID(37) ENCODING(RRR) MSGBODY(MQ) QMANAGER(QM1) QUEUE(Q1) TARGCLIENT(MQ)
   VERSION(7)
   END
   ```

   f. Uruchom program JMSAdmin, aby utworzyć zasoby JMS .

   ```
   JMSAdmin < JMSQM1Q1.txt
   ```

2. Utworzone definicje można tworzyć, zmieniać i przeglądać za pomocą Eksploratora IBM MQ .
3. Uruchom program `TryMyRecord`.

**Klasy używane do uruchamiania przykładu**

Klasy wymienione w poniższych blokach kodu są również dostępne w skompresowanym pliku. Pobierz plik jm25529_.zip lub 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

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;

public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }

    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }


    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
```

```
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

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.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }
    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }

    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
```

```
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

MyRecord

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;

public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }

    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }

    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }

    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .


    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }

    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
```

```
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }
    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }
    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }
    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .

    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }
    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

EndPoint

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import jakarta.jms.Connection;
import jakarta.jms.ConnectionFactory;
import jakarta.jms.Destination;
import jakarta.jms.JMSException;
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, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
```

```
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

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.JMSException;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
```

```
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

MyProducer

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

MyConsumer

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

## *Tworzenie i konfigurowanie fabryk połączeń i miejsc docelowych*

Aplikacje IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging mogą tworzyć fabryki połączeń i miejsca docelowe, pobierając je jako obiekty administrowane z przestrzeni nazw JNDI ( Java Naming and Directory Interface), używając rozszerzeń IBM JMS lub rozszerzeń IBM MQ JMS . Aplikacja może również używać rozszerzeń IBM JMS lub rozszerzeń IBM MQ JMS do ustawiania właściwości fabryk połączeń i miejsc docelowych.

Fabryki połączeń i miejsca docelowe to punkty początkowe w przepływie logiki aplikacji JMS lub Jakarta Messaging . Aplikacja używa obiektu ConnectionFactory do utworzenia połączenia z serwerem przesyłania

komunikatów i używa obiektu Queue lub Topic jako obiektu docelowego do wysyłania komunikatów do źródła lub do odbierania komunikatów. Dlatego aplikacja musi utworzyć co najmniej jedną fabrykę połączeń i co najmniej jedno miejsce docelowe. Po utworzeniu fabryki połączeń lub miejsca docelowego aplikacja może wymagać skonfigurowania obiektu przez ustawienie co najmniej jednej z jego właściwości.

Podsumowując, aplikacja może tworzyć i konfigurować fabryki połączeń i miejsca docelowe w następujący sposób:

**Używanie interfejsu JNDI do pobierania obiektów administrowanych**
Administrator może użyć narzędzia administracyjnego produktu IBM MQ JMS zgodnie z opisem w sekcji Konfigurowanie obiektów JMS i Jakarta Messaging przy użyciu narzędzi administracyjnychlub IBM MQ Explorer zgodnie z opisem w sekcji Konfigurowanie obiektów JMS 2.0 przy użyciu programu IBM MQ Explorer, aby utworzyć i skonfigurować fabryki połączeń i miejsca docelowe jako obiekty administrowane w przestrzeni nazw JNDI. Aplikacja może następnie pobrać administrowane obiekty z przestrzeni nazw JNDI. Po pobraniu obiektu administrowanego aplikacja może, jeśli jest to wymagane, ustawić lub zmienić jedną lub więcej jego właściwości za pomocą rozszerzeń IBM JMS lub rozszerzeń IBM MQ JMS .

**Uwaga:** V 9.3.0 JM 3.0 V 9.3.0 W przypadku systemu Jakarta Messaging 3.0nie można administrować interfejsem JNDI przy użyciu programu IBM MQ Explorer. Administrowanie interfejsem JNDI jest obsługiwane przez Jakarta Messaging 3.0 wariant **JMSAdmin**, który ma wartość **JMS30Admin**.

**Korzystanie z rozszerzeń IBM JMS**
Aplikacja może używać rozszerzeń produktu IBM JMS do dynamicznego tworzenia fabryk połączeń i miejsc docelowych w czasie wykonywania. Aplikacja najpierw tworzy obiekt fabryki JmsFactory, a następnie używa metod tego obiektu do tworzenia fabryk połączeń i miejsc docelowych. Po utworzeniu fabryki połączeń lub miejsca docelowego aplikacja może użyć metod dziedziczonych z interfejsu kontekstu JmsPropertyw celu ustawienia swoich właściwości. Alternatywnie aplikacja może użyć identyfikatora URI (Uniform Resource Identifier) w celu określenia co najmniej jednej właściwości miejsca docelowego podczas tworzenia miejsca docelowego.

**Korzystanie z rozszerzeń IBM MQ JMS**
Aplikacja może również używać rozszerzeń produktu IBM MQ JMS do dynamicznego tworzenia fabryk połączeń i miejsc docelowych w czasie wykonywania. Aplikacja używa dostarczonych konstruktorów do tworzenia fabryk połączeń i miejsc docelowych. Po utworzeniu fabryki połączeń lub miejsca docelowego aplikacja może użyć metod obiektu do ustawienia jego właściwości. Alternatywnie aplikacja może użyć identyfikatora URI w celu określenia jednej lub większej liczby właściwości miejsca docelowego podczas tworzenia miejsca docelowego.

**Zadania pokrewne**
Konfigurowanie zasobów JMS i Jakarta Messaging

*Używanie interfejsu JNDI do pobierania obiektów administrowanych w aplikacji JMS lub Jakarta Messaging*
Aby pobrać obiekty administrowane z przestrzeni nazw JNDI ( Java Naming and Directory Interface), aplikacja JMS lub Jakarta Messaging musi utworzyć kontekst początkowy, a następnie użyć metody lookup () w celu pobrania obiektów.

Zanim aplikacja będzie mogła pobrać obiekty administrowane z przestrzeni nazw JNDI, administrator musi najpierw utworzyć obiekty administrowane.

JMS 2.0 W przypadku systemu JMS 2.0administrator może użyć narzędzia administracyjnego IBM MQ JMS , **JMSAmin**lub IBM MQ Explorer , aby utworzyć i obsługiwać obiekty administrowane w przestrzeni nazw JNDI. Więcej informacji na ten temat zawiera sekcja Konfigurowanie fabryk połączeń i miejsc docelowych w przestrzeni nazw JNDI.

V 9.3.0 JM 3.0 V 9.3.0 W przypadku systemu Jakarta Messaging 3.0nie można administrować interfejsem JNDI przy użyciu programu IBM MQ Explorer. Administrowanie interfejsem JNDI jest obsługiwane przez Jakarta Messaging 3.0 wariant **JMSAdmin**, który ma wartość **JMS30Admin**.

Serwer aplikacji zwykle udostępnia własne repozytorium dla obiektów administrowanych oraz własne narzędzia do tworzenia i obsługi obiektów.

Aby pobrać obiekty administrowane z przestrzeni nazw JNDI, aplikacja musi najpierw utworzyć kontekst początkowy, jak pokazano w poniższym przykładzie:

V 9.3.0 JM 3.0 V 9.3.0

```
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);
```

W tym kodzie zmienne łańcuchowe `url` i `icf` mają następujące znaczenie:

**url**
Adres URL (URL) usługi katalogowej. URL może mieć jeden z następujących formatów:

- `ldap://`*`hostname`*`/`*`contextName`* dla usługi katalogowej opartej na serwerze LDAP
- `file:/`*`directoryPath`* dla usługi katalogowej opartej na lokalnym systemie plików

**icf,**
Nazwa klasy fabryki kontekstu początkowego, która może mieć jedną z następujących wartości:

- `com.sun.jndi.ldap.LdapCtxFactory`dla usługi katalogowej opartej na serwerze LDAP
- `com.sun.jndi.fscontext.RefFSContextFactory`dla usługi katalogowej opartej na lokalnym systemie plików

Należy zauważyć, że niektóre kombinacje pakietu JNDI i dostawcy usług LDAP (Lightweight Directory Access Protocol) mogą spowodować wystąpienie błędu LDAP 84. Aby rozwiązać ten problem, należy wstawić następujący wiersz kodu przed wywołaniem metody InitialDirContext ():

```
environment.put(Context.REFERRAL, "throw");
```

Po uzyskaniu kontekstu początkowego aplikacja może pobrać obiekty administrowane z przestrzeni nazw JNDI przy użyciu metody lookup (), jak pokazano w poniższym przykładzie:

```
ConnectionFactory factory;
Queue queue;
Topic topic;
.
.
.
factory = (ConnectionFactory)ctx.lookup("cn=myCF");
```

```
queue = (Queue)ctx.lookup("cn=myQ");
topic = (Topic)ctx.lookup("cn=myT");
```

Ten kod pobiera następujące obiekty z przestrzeni nazw opartej na protokole LDAP:

- Obiekt ConnectionFactory powiązany z nazwą myCF .
- Obiekt kolejki powiązany z nazwą myQ
- Obiekt tematu powiązany z nazwą myT

Więcej informacji na temat używania interfejsu JNDI zawiera dokumentacja interfejsu JNDI dostarczana przez firmę Oracle Corporation.

**Zadania pokrewne**

Konfigurowanie obiektów JMS 2.0 przy użyciu programu IBM MQ Explorer

Konfigurowanie obiektów JMS i Jakarta Messaging przy użyciu narzędzi administracyjnych

Konfigurowanie zasobów JMS 2.0 na serwerze WebSphere Application Server

*Korzystanie z rozszerzeń IBM JMS*

IBM MQ classes for JMS (JMS 2.0) i IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) każdy z nich zawiera identyczny funkcjonalnie zestaw rozszerzeń do interfejsu API JMS o nazwie IBM JMS . Aplikacja może używać tych rozszerzeń do dynamicznego tworzenia fabryk połączeń i miejsc docelowych w czasie wykonywania oraz do ustawiania właściwości obiektów IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging . Rozszerzenia mogą być używane z dowolnym dostawcą przesyłania komunikatów.

Rozszerzenia produktu IBM JMS są zestawem interfejsów i klas w następujących pakietach:

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

W przypadku systemu Jakarta Messaging 3.0pakiety te znajdują się w katalogu `com.ibm.jakarta.client.jar`.

JMS 2.0 W przypadku systemu JMS 2.0pakiety te znajdują się w katalogu `com.ibm.mqjms.jar` lub `com.ibm.mq.allclient.jar`.

Rozszerzenia te udostępniają następujące funkcje:

- Oparty na fabryce mechanizm służący do dynamicznego tworzenia fabryk połączeń i miejsc docelowych w czasie wykonywania zamiast pobierania ich jako obiektów administrowanych z przestrzeni nazw JNDI ( Java Naming and Directory Interface).
- Zestaw metod służących do ustawiania właściwości obiektów IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging .
- Zestaw klas wyjątków z metodami uzyskiwania szczegółowych informacji o problemie
- Zestaw metod do sterowania śledzeniem
- Zestaw metod uzyskiwania informacji o wersji systemu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging

W celu dynamicznego tworzenia fabryk połączeń i miejsc docelowych w czasie wykonywania oraz ustawiania i pobierania ich właściwości, rozszerzenia produktu IBM JMS udostępniają alternatywny zestaw interfejsów do rozszerzeń produktu IBM MQ JMS . Chociaż rozszerzenia produktu IBM MQ JMS są specyficzne dla dostawcy przesyłania komunikatów produktu IBM MQ , rozszerzenia produktu IBM JMS nie są specyficzne dla produktu IBM MQ i mogą być używane z dowolnym dostawcą przesyłania komunikatów w architekturze warstwowej opisanej w sekcji IBM MQ classes for JMS architecture.

Interfejs com.ibm.msg.client.wmq.WMQConstants (JMS 2.0) lub com.ibm.msg.jakarta.client.wmq.WMQConstants (Jakarta Messaging 3.0) zawiera definicje stałych, które mogą być używane przez aplikację podczas ustawiania właściwości obiektów IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging przy użyciu rozszerzeń produktu IBM JMS . Interfejs zawiera stałe dla dostawcy przesyłania komunikatów IBM MQ i stałe JMS , które są niezależne od dowolnego dostawcy przesyłania komunikatów.

W poniższych przykładach kodu założono, że w klasie Java znajdują się następujące instrukcje importu:

V 9.3.0 JM 3.0 V 9.3.0

```
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;
```

## Tworzenie fabryk połączeń i miejsc docelowych

Zanim aplikacja będzie mogła tworzyć fabryki połączeń i miejsca docelowe przy użyciu rozszerzeń produktu IBM JMS , musi najpierw utworzyć obiekt fabryki JmsFactory. Aby utworzyć obiekt fabryki JmsFactory, aplikacja wywołuje metodę getInstance() klasy fabryki JmsFactory, jak pokazano w poniższym przykładzie:

V 9.3.0 JM 3.0 V 9.3.0

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.JAKARTA_WMQ_PROVIDER);
```

JMS 2.0

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
```

Parametr w wywołaniu metody getInstance() jest stałą identyfikującą dostawcę przesyłania komunikatów produktu IBM MQ jako wybranego dostawcę przesyłania komunikatów. Aplikacja może następnie użyć obiektu fabryki JmsFactorydo utworzenia fabryk połączeń i miejsc docelowych.

Aby utworzyć fabrykę połączeń, aplikacja wywołuje metodę createConnectionFactory () obiektu fabryki JmsFactory, jak pokazano w poniższym przykładzie:

```
JmsConnectionFactory factory = ff.createConnectionFactory();
```

Ta instrukcja tworzy obiekt fabryki JmsConnectionz wartościami domyślnymi dla wszystkich jego właściwości, co oznacza, że aplikacja nawiązuje połączenie z domyślnym menedżerem kolejek w trybie powiązań. Aby aplikacja mogła nawiązać połączenie w trybie klienta lub nawiązać połączenie z menedżerem kolejek innym niż domyślny menedżer kolejek, przed utworzeniem połączenia aplikacja musi ustawić odpowiednie właściwości obiektu fabryki JmsConnection. Więcej informacji na ten temat zawiera sekcja “Ustawianie właściwości obiektów IBM MQ classes for JMS” na stronie 216.

Klasa fabryki JmsFactoryzawiera również metody służące do tworzenia fabryk połączeń następujących typów:

- JmsQueueConnectionFactory
- JmsTopicConnectionFactory
- Fabryka JmsXAConnection
- JmsXAQueueConnectionFactory
- JmsXATopicConnectionFactory

Aby utworzyć obiekt Queue, aplikacja wywołuje metodę createQueue() obiektu fabryki JmsFactory, jak pokazano w poniższym przykładzie:

```
JmsQueue q1 = ff.createQueue("Q1");
```

Ta instrukcja tworzy obiekt JmsQueue z wartościami domyślnymi dla wszystkich jego właściwości. Obiekt reprezentuje kolejkę IBM MQ o nazwie Q1 , która należy do lokalnego menedżera kolejek. Ta kolejka może być kolejką lokalną, kolejką aliasową lub definicją kolejki zdalnej.

Metoda createQueue() może również zaakceptować identyfikator URI kolejki jako parametr. Identyfikator URI kolejki to łańcuch określający nazwę kolejki produktu IBM MQ i opcjonalnie nazwę menedżera kolejek będącego właścicielem kolejki oraz jedną lub więcej właściwości obiektu JmsQueue . Poniższa instrukcja zawiera przykład identyfikatora URI kolejki:

```
JmsQueue q2 = ff.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

Obiekt JmsQueue utworzony przez tę instrukcję reprezentuje kolejkę IBM MQ o nazwie Q2 , której właścicielem jest menedżer kolejek QM2, a wszystkie komunikaty wysyłane do tego miejsca docelowego są trwałe i mają priorytet 5. Więcej informacji na temat identyfikatorów URI kolejek zawiera sekcja “Identyfikatory URI (Uniform Resource Identifier)” na stronie 229. Alternatywny sposób ustawiania właściwości obiektu JmsQueue zawiera sekcja “Ustawianie właściwości obiektów IBM MQ classes for JMS” na stronie 216.

Aby utworzyć obiekt Topic, aplikacja może użyć metody createTopic() obiektu fabryki JmsFactory, jak pokazano w poniższym przykładzie:

```
JmsTopic t1 = ff.createTopic("Sport/Football/Results");
```

Ta instrukcja tworzy obiekt JmsTopic z wartościami domyślnymi dla wszystkich jego właściwości. Obiekt reprezentuje temat o nazwie Sport/Stopka/Wyniki.

Metoda createTopic() może również zaakceptować identyfikator URI tematu jako parametr. Identyfikator URI tematu to łańcuch określający nazwę tematu i opcjonalnie jedną lub więcej właściwości obiektu JmsTopic . Poniższe instrukcje zawierają przykład identyfikatora URI tematu:

```
String s1 = "topic://Sport/Tennis/Results?persistence=1&priority=0";
JmsTopic t2 = ff.createTopic(s1);
```

Obiekt JmsTopic utworzony przez te instrukcje reprezentuje temat o nazwie Sport/Tennis/Results, a wszystkie komunikaty wysyłane do tego miejsca docelowego są nietrwałe i mają priorytet 0. Więcej informacji na temat identyfikatorów URI tematów zawiera sekcja “Identyfikatory URI (Uniform Resource Identifier)” na stronie 229. Alternatywny sposób ustawiania właściwości obiektu JmsTopic zawiera sekcja “Ustawianie właściwości obiektów IBM MQ classes for JMS” na stronie 216.

Po utworzeniu fabryki połączeń lub miejsca docelowego przez aplikację ten obiekt może być używany tylko z wybranym dostawcą przesyłania komunikatów.

## Ustawianie właściwości obiektów IBM MQ classes for JMS

Aby ustawić właściwości obiektów IBM MQ classes for JMS przy użyciu rozszerzeń IBM JMS , aplikacja używa metod interfejsu com.ibm.msg.client.JmsPropertyContext . Podobnie, aby ustawić właściwości obiektów IBM MQ classes for Jakarta Messaging za pomocą rozszerzeń IBM JMS , aplikacja używa metod interfejsu com.ibm.msg.jakarta.client.JmsPropertyContext .

Dla każdego typu danych Java interfejs kontekstu JmsPropertyzawiera metodę służącą do ustawiania wartości właściwości o tym typie danych oraz metodę służącą do pobierania wartości właściwości o tym typie danych. Na przykład aplikacja wywołuje metodę setIntProperty () w celu ustawienia właściwości o wartości całkowitej i wywołuje metodę getIntProperty () w celu pobrania właściwości o wartości całkowitej.

Instancje klas w pakietach com.ibm.mq.jms i com.ibm.mq.jakarta.jms dziedziczą metody odpowiednich interfejsów kontekstu JmsProperty. W związku z tym aplikacja może używać tych metod do ustawiania właściwości obiektów MQConnectionFactory, MQQueue i MQTopic.

Gdy aplikacja tworzy obiekt IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging , wszystkie właściwości z wartościami domyślnymi są ustawiane automatycznie. Gdy aplikacja ustawia właściwość, nowa wartość zastępuje poprzednią wartość właściwości. Po ustawieniu właściwości nie można jej usunąć, ale można zmienić jej wartość.

Jeśli aplikacja spróbuje ustawić właściwość na wartość, która nie jest poprawną wartością dla tej właściwości, produkt IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging zgłosi wyjątek JMSException. Jeśli aplikacja próbuje uzyskać właściwość, która nie została ustawiona, zachowanie jest zgodne ze specyfikacją JMS . IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging zgłaszają wyjątek NumberFormatdla podstawowych typów danych i zwracają wartość NULL dla przywoływanych typów danych.

Oprócz predefiniowanych właściwości obiektu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging aplikacja może ustawiać własne właściwości. Te właściwości zdefiniowane przez aplikację są ignorowane przez produkt IBM MQ classes for JMS i produkt IBM MQ classes for Jakarta Messaging.

Więcej informacji na temat właściwości obiektów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging zawiera sekcja Właściwości obiektów IBM MQ classes for JMS.

Poniższy kod jest przykładem ustawiania właściwości przy użyciu rozszerzeń IBM JMS . Kod ustawia pięć właściwości fabryki połączeń.

```
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");
```

Efektem ustawienia tych właściwości jest połączenie aplikacji z menedżerem kolejek QM1 w trybie klienta przy użyciu kanału MQI o nazwie QM1.SVR. Menedżer kolejek działa w systemie o nazwie hosta HOST1, a program nasłuchujący menedżera kolejek nasłuchuje na porcie o numerze 1415. To połączenie i inne połączenia menedżera kolejek powiązane z sesjami w ramach tego połączenia mają powiązaną nazwę aplikacji "Moja aplikacja".

**Uwaga:** Menedżery kolejek działające na platformach z/OS nie obsługują ustawiania nazw aplikacji, dlatego to ustawienie jest ignorowane.

Interfejs kontekstu JmsPropertyzawiera również metodę setObjectProperty (), której aplikacja może używać do ustawiania właściwości. Drugim parametrem metody jest obiekt, który hermetyzuje wartość właściwości. Na przykład poniższy kod tworzy obiekt typu Integer, który hermetyzuje liczbę całkowitą 1415, a następnie wywołuje metodę setObjectProperty () w celu ustawienia właściwości PORT fabryki połączeń na wartość 1415:

```
Integer port = new Integer(1415);
factory.setObjectProperty(WMQConstants.WMQ_PORT, port);
```

Kod ten jest zatem równoznaczny z następującym stwierdzeniem:

```
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
```

I odwrotnie, metoda getObjectProperty () zwraca obiekt, który hermetyzuje wartość właściwości.

## Niejawna konwersja wartości właściwości z jednego typu danych do innego

Jeśli aplikacja używa metody interfejsu kontekstu JmsPropertyw celu ustawienia lub pobrania właściwości obiektu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging , wartość właściwości może zostać niejawnie przekształcona z jednego typu danych na inny.

Na przykład następująca instrukcja ustawia właściwość PRIORITY obiektu JmsQueue q1:

```
q1.setStringProperty(WMQConstants.WMQ_PRIORITY, "5");
```

Właściwość PRIORITY ma wartość całkowitą, więc wywołanie metody setStringProperty () niejawnie przekształca łańcuch "5" (wartość źródłowa) w liczbę całkowitą 5 (wartość docelowa), która staje się wartością właściwości PRIORITY.

I odwrotnie, następująca instrukcja pobiera właściwość PRIORITY obiektu JmsQueue q1:

```
String s1 = q1.getStringProperty(WMQConstants.WMQ_PRIORITY);
```

Liczba całkowita 5 (wartość źródłowa), która jest wartością właściwości PRIORITY, jest niejawnie przekształcana w łańcuch "5" (wartość docelowa) przez wywołanie metody getString().

Konwersje obsługiwane przez produkty IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging przedstawia Tabela 34 na stronie 218.

*Tabela 34. Obsługiwane konwersje z jednego typu danych na inny*

| **Źródłowy typ danych** | **Obsługiwane docelowe typy danych** |
|---|---|
| boolean (boolowskie) | Łańcuch |
| B | int, long, short, String |
| char | Łańcuch |
| double (podwójna) | Łańcuch |
| liczba zmiennopozycyjna | double, łańcuch |
| int | long, łańcuch |
| long | Łańcuch |
| short | int, long, String |
| Łańcuch | boolean, byte, double, float, int, long, short |

Ogólne zasady dotyczące obsługiwanych konwersji są następujące:

- Wartości liczbowe mogą być przekształcane z jednego typu danych na inny, pod warunkiem, że podczas konwersji nie zostaną utracone żadne dane. Na przykład wartość o typie danych `int` może zostać przekształcona w wartość o typie danych `long`, ale nie może zostać przekształcona w wartość o typie danych `short`.
- Wartość dowolnego typu danych może zostać przekształcona w łańcuch.
- Łańcuch może zostać przekształcony w wartość dowolnego innego typu danych (z wyjątkiem `char` ) pod warunkiem, że łańcuch ma poprawny format dla konwersji. Jeśli aplikacja podejmie próbę przekształcenia łańcucha w niepoprawnym formacie, produkt IBM MQ classes for JMS i produkt IBM MQ classes for Jakarta Messaging zgłoszą wyjątek NumberFormat.
- Jeśli aplikacja podejmie próbę konwersji, która nie jest obsługiwana, produkty IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging zgłoszą wyjątek MessageFormat.

Poniżej przedstawiono szczegółowe reguły przekształcania wartości z jednego typu danych na inny:

- Podczas przekształcania wartości boolowskiej w łańcuch, wartość `true` jest przekształcana w łańcuch "true", a wartość `false` jest przekształcana w łańcuch "false".
- Podczas przekształcania łańcucha w wartość boolowską łańcuch "true" (bez rozróżniania wielkości liter) jest przekształcany w łańcuch `true`, a łańcuch "false" (bez rozróżniania wielkości liter) jest przekształcany w łańcuch `false`. Każdy inny łańcuch jest przekształcany w łańcuch `false`.
- Podczas przekształcania łańcucha w wartość o typie danych `byte`, `int`, `long`lub `short`łańcuch musi mieć następujący format:

  [ *odstępy* ] [ *znak* ] *cyfry*

  Znaczenia komponentów łańcucha są następujące:

  ***wartości puste***
  Opcjonalne początkowe znaki odstępu.

***znak***
Opcjonalny znak plus (+) lub minus (-).

***cyfry***
Ciągła sekwencja cyfr (0-9). Musi być obecna co najmniej jedna cyfra.

Po sekwencji cyfr łańcuch może zawierać inne znaki, które nie są cyframi, ale konwersja jest zatrzymywana natychmiast po osiągnięciu pierwszego z tych znaków. Przyjmuje się, że łańcuch reprezentuje dziesiętną liczbę całkowitą.

Jeśli łańcuch nie ma poprawnego formatu, IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging zgłaszają wyjątek NumberFormat.

- Podczas przekształcania łańcucha w wartość o typie danych `double` lub `float`łańcuch musi mieć następujący format:

  [ *odstępy* ] [ *znak* ] *cyfry* [ *znak* [ *znak* ] *cyfry* ]

  Znaczenia komponentów łańcucha są następujące:

  ***wartości puste***
  Opcjonalne początkowe znaki odstępu.

  ***znak***
  Opcjonalny znak plus (+) lub minus (-).

  ***cyfry***
  Ciągła sekwencja cyfr (0-9). Musi być obecna co najmniej jedna cyfra.

  ***_znak***
  Znak wykładnika, który jest *E* lub *e*.

  ***e_sign (znak)***
  Opcjonalny znak plus (+) lub minus (-) dla wykładnika.

  ***e_cyfry***
  Ciągła sekwencja cyfr (0-9) dla wykładnika. Jeśli łańcuch zawiera znak wykładnika, musi być obecna co najmniej jedna cyfra.

  Po sekwencji cyfr lub opcjonalnych znakach reprezentujących wykładnik łańcuch może zawierać inne znaki, które nie są cyframi, ale konwersja kończy się natychmiast po osiągnięciu pierwszego z tych znaków. Przyjmuje się, że łańcuch reprezentuje dziesiętną liczbę zmiennopozycyjną z wykładnikiem, który jest potęgą liczby 10.

  Jeśli łańcuch nie ma poprawnego formatu, IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging zgłaszają wyjątek NumberFormat.

- Podczas przekształcania wartości liczbowej (w tym wartości o typie danych `byte` ) Wartość jest przekształcana w łańcuch reprezentujący wartość jako liczbę dziesiętną, a nie w łańcuch zawierający znak ASCII dla tej wartości. Na przykład liczba całkowita 65 jest przekształcana w łańcuch "65", a nie w łańcuch "A".

## Ustawianie więcej niż jednej właściwości w pojedynczym wywołaniu

Interfejs kontekstu JmsPropertyzawiera również metodę setBatchProperties (), której aplikacja może używać do ustawiania więcej niż jednej właściwości w pojedynczym wywołaniu. Parametrem metody jest obiekt Map, który hermetyzuje zestaw par nazwa-wartość właściwości.

Na przykład w poniższym kodzie użyto metody setBatchProperties () do ustawienia tych samych pięciu właściwości fabryki połączeń, które przedstawia “Ustawianie właściwości obiektów IBM MQ classes for JMS” na stronie 216. Kod tworzy instancję klasy HashMap , która implementuje interfejs Map.

```
HashMap batchProperties = new HashMap();
batchProperties.put(WMQConstants.WMQ_CONNECTION_MODE,
                    new Integer(WMQConstants.WMQ_CM_CLIENT));
batchProperties.put(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
batchProperties.put(WMQConstants.WMQ_WMQ_HOST_NAME, "HOST1");
batchProperties.put(WMQConstants.WMQ_PORT, "1414");
```

```
batchProperties.put(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setBatchProperties(batchProperties);
```

Należy zauważyć, że drugi parametr metody Map.put() musi być obiektem. Dlatego wartość właściwości z podstawowym typem danych musi być hermetyzowana w obiekcie lub reprezentowana przez łańcuch, jak pokazano w przykładzie.

Metoda setBatchProperties () sprawdza poprawność każdej właściwości. Jeśli metoda setBatchProperties () nie może ustawić właściwości, ponieważ na przykład jej wartość jest niepoprawna, żadna z podanych właściwości nie zostanie ustawiona.

## Nazwy i wartości właściwości

Jeśli aplikacja używa metod odpowiedniego interfejsu kontekstu JmsPropertyw celu ustawienia i pobrania właściwości obiektów IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging , aplikacja może określić nazwy i wartości właściwości w dowolny z następujących sposobów. Każdy z dołączonych przykładów przedstawia sposób ustawiania właściwości PRIORITY obiektu JmsQueue q1 , tak aby komunikat wysyłany do kolejki miał priorytet określony w wywołaniu send ().

**Korzystanie z nazw i wartości właściwości zdefiniowanych jako stałe w interfejsie com.ibm.msg.client.wmq.WMQConstants**

Poniższa instrukcja jest przykładem sposobu określania nazw i wartości właściwości w ten sposób:

```
q1.setIntProperty(WMQConstants.WMQ_PRIORITY, WMQConstants.WMQ_PRI_APP);
```

**Przy użyciu nazw i wartości właściwości, które mogą być używane w kolejkach i identyfikatorach URI (uniform resource identifier) tematów**

Poniższa instrukcja jest przykładem sposobu określania nazw i wartości właściwości w ten sposób:

```
q1.setIntProperty("priority", -2);
```

W ten sposób można określić tylko nazwy i wartości właściwości miejsc docelowych.

**Korzystanie z nazw i wartości właściwości rozpoznawanych przez narzędzie administracyjne IBM MQ JMS**

Poniższa instrukcja jest przykładem sposobu określania nazw i wartości właściwości w ten sposób:

```
q1.setStringProperty("PRIORITY", "APP");
```

Krótka postać nazwy właściwości jest również akceptowalna, jak pokazano w następującej instrukcji:

```
q1.setStringProperty("PRI", "APP");
```

Gdy aplikacja pobiera właściwość, zwracana wartość zależy od sposobu, w jaki aplikacja określa nazwę właściwości. Jeśli na przykład aplikacja określa stałą WMQConstants.WMQ_PRIORITY jako nazwę właściwości, zwracana wartość jest liczbą całkowitą -2:

```
int n1 = getIntProperty(WMQConstants.WMQ_PRIORITY);
```

Ta sama wartość jest zwracana, jeśli aplikacja określa łańcuch "priority" jako nazwę właściwości:

```
int n2 = getIntProperty("priority");
```

Jeśli jednak aplikacja określa łańcuch "PRIORITY" lub "PRI" jako nazwę właściwości, zwracana wartość to łańcuch "APP":

```
String s1 = getStringProperty("PRI");
```

Wewnętrznie IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging przechowują nazwy i wartości właściwości jako wartości literałów zdefiniowane w zgodnym interfejsie WMQConstants. Jest to zdefiniowany format kanoniczny dla nazw i wartości właściwości. Ogólnie rzecz biorąc, jeśli aplikacja ustawia właściwości przy użyciu jednego z dwóch innych sposobów określania nazw i wartości właściwości, IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging muszą przekształcić nazwy i wartości z określonego formatu wejściowego na format kanoniczny. Podobnie, jeśli aplikacja pobiera właściwości przy użyciu jednego z dwóch innych sposobów określania nazw i wartości właściwości, IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging muszą przekształcić nazwy z określonego formatu wejściowego na format kanoniczny i przekształcić wartości z formatu kanonicznego na wymagany format wyjściowy. Konieczność przeprowadzenia tych konwersji może mieć wpływ na wydajność.

Nazwy i wartości właściwości zwracane przez wyjątki, w plikach śledzenia lub w dzienniku produktu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging są zawsze w formacie kanonicznym.

## Korzystanie z interfejsu Map

Interfejs kontekstu JmsPropertyrozszerza interfejs java.util.Map . Dlatego aplikacja może korzystać z metod interfejsu Map w celu uzyskania dostępu do właściwości obiektu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging .

Na przykład poniższy kod wyświetla nazwy i wartości wszystkich właściwości fabryki połączeń. Kod używa tylko metod interfejsu Map do pobrania nazw i wartości właściwości.

```
// Get the names of all the properties
Set propNames = factory.keySet();

// Loop round all the property names and get the property values
Iterator iterator = propNames.iterator();
while (iterator.hasNext()){
    String pName = (String)iterator.next();
    System.out.println(pName+"="+factory.get(pName));
}
```

Użycie metod interfejsu Map nie pomija sprawdzania poprawności ani konwersji właściwości.

*Korzystanie z rozszerzeń IBM MQ JMS*
Plik IBM MQ classes for JMS zawiera zestaw rozszerzeń interfejsu API języka JMS nazywanych rozszerzeniami produktu IBM MQ JMS . Aplikacja może używać tych rozszerzeń do dynamicznego tworzenia fabryk połączeń i miejsc docelowych w czasie wykonywania oraz do ustawiania właściwości fabryk połączeń i miejsc docelowych.

Plik IBM MQ classes for JMS zawiera zestaw klas w pakietach com.ibm.jms i com.ibm.mq.jms. Te klasy implementują interfejsy JMS i zawierają rozszerzenia produktu IBM MQ JMS . W poniższych przykładach kodu założono, że te pakiety zostały zaimportowane za pomocą następujących instrukcji:

```
import com.ibm.jms.*;
import com.ibm.mq.jms.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

Aplikacja może używać rozszerzeń IBM MQ JMS do wykonywania następujących funkcji:

- Dynamiczne tworzenie fabryk połączeń i miejsc docelowych w czasie wykonywania zamiast pobierania ich jako obiektów administrowanych z przestrzeni nazw JNDI ( Java Naming and Directory Interface).
- Ustawianie właściwości fabryk połączeń i miejsc docelowych

## Tworzenie fabryk połączeń

Aby utworzyć fabrykę połączeń, aplikacja może użyć konstruktora MQConnectionFactory , jak pokazano w poniższym przykładzie:

```
MQConnectionFactory factory = new MQConnectionFactory();
```

Ta instrukcja tworzy obiekt MQConnectionFactory z wartościami domyślnymi dla wszystkich jego właściwości, co oznacza, że aplikacja nawiązuje połączenie z domyślnym menedżerem kolejek w trybie powiązań. Jeśli aplikacja ma nawiązać połączenie w trybie klienta lub nawiązać połączenie z menedżerem kolejek innym niż domyślny menedżer kolejek, przed utworzeniem połączenia aplikacja musi ustawić odpowiednie właściwości obiektu MQConnectionFactory . Więcej informacji na ten temat zawiera sekcja “Ustawianie właściwości fabryk połączeń” na stronie 222.

W podobny sposób aplikacja może tworzyć fabryki połączeń następujących typów:

- Fabryka MQQueueConnection
- Fabryka MQTopicConnection
- MQXAConnectionFactory
- Fabryka MQXAQueueConnection
- Fabryka MQXATopicConnection

## Ustawianie właściwości fabryk połączeń

Aplikacja może ustawić właściwości fabryki połączeń, wywołując odpowiednie metody fabryki połączeń. Fabryka połączeń może być obiektem administrowanym lub obiektem tworzonym dynamicznie w czasie wykonywania.

Rozważmy następujący kod, na przykład:

```
MQConnectionFactory factory = new MQConnectionFactory();
.
factory.setTransportType(WMQConstants.WMQ_CM_CLIENT);
factory.setQueueManager("QM1");
factory.setHostName("HOST1");
factory.setPort(1415);
factory.setChannel("QM1.SVR");
```

Ten kod tworzy obiekt MQConnectionFactory , a następnie ustawia pięć właściwości obiektu. Efektem ustawienia tych właściwości jest połączenie aplikacji z menedżerem kolejek QM1 w trybie klienta przy użyciu kanału MQI o nazwie QM1.SVR. Menedżer kolejek działa w systemie o nazwie hosta HOST1, a program nasłuchujący menedżera kolejek nasłuchuje na porcie o numerze 1415.

Aplikacja, która używa połączenia w czasie rzeczywistym z brokerem, może używać tylko stylu przesyłania komunikatów publikowania/subskrypcji. Nie może używać stylu przesyłania komunikatów punkt z punktem.

Poprawne są tylko niektóre kombinacje właściwości fabryki połączeń. Informacje na temat poprawnych kombinacji zawiera sekcja Zależności między właściwościami obiektów IBM MQ classes for JMS.

Więcej informacji na temat właściwości fabryki połączeń oraz metod używanych do ustawiania jej właściwości zawiera sekcja Właściwości obiektów IBM MQ classes for JMS.

## Tworzenie miejsc docelowych

Aby utworzyć obiekt kolejki, aplikacja może użyć konstruktora MQQueue, jak pokazano w poniższym przykładzie:

```
MQQueue q1 = new MQQueue("Q1");
```

Ta instrukcja tworzy obiekt MQQueue z wartościami domyślnymi dla wszystkich jego właściwości. Obiekt reprezentuje kolejkę IBM MQ o nazwie Q1 , która należy do lokalnego menedżera kolejek. Ta kolejka może być kolejką lokalną, kolejką aliasową lub definicją kolejki zdalnej.

Alternatywna forma konstruktora MQQueue ma dwa parametry, jak pokazano w poniższym przykładzie:

```
MQQueue q2 = new MQQueue("QM2", "Q2");
```

Obiekt MQQueue utworzony przez tę instrukcję reprezentuje kolejkę IBM MQ o nazwie Q2 , której właścicielem jest menedżer kolejek QM2. Zidentyfikowany w ten sposób menedżer kolejek może być lokalnym lub zdalnym menedżerem kolejek. Jeśli jest to zdalny menedżer kolejek, produkt IBM MQ musi być skonfigurowany w taki sposób, aby podczas wysyłania komunikatu przez aplikację do tego miejsca docelowego produkt WebSphere MQ mógł kierować komunikat z lokalnego menedżera kolejek do zdalnego menedżera kolejek.

Konstruktor MQQueue może również zaakceptować identyfikator URI kolejki jako pojedynczy parametr. Identyfikator URI kolejki to łańcuch określający nazwę kolejki produktu IBM MQ i opcjonalnie nazwę menedżera kolejek, który jest właścicielem kolejki, oraz co najmniej jedną właściwość obiektu MQQueue. Poniższa instrukcja zawiera przykład identyfikatora URI kolejki:

```
MQQueue q3 = new MQQueue("queue://QM3/Q3?persistence=2&priority=5");
```

Obiekt MQQueue utworzony przez tę instrukcję reprezentuje kolejkę IBM MQ o nazwie Q3 , której właścicielem jest menedżer kolejek QM3, a wszystkie komunikaty wysyłane do tego miejsca docelowego są trwałe i mają priorytet 5. Więcej informacji na temat identyfikatorów URI kolejek zawiera sekcja “Identyfikatory URI (Uniform Resource Identifier)” na stronie 229. Alternatywny sposób ustawiania właściwości obiektu MQQueue można znaleźć w sekcji “Ustawianie właściwości miejsc docelowych” na stronie 223.

Aby utworzyć obiekt Topic, aplikacja może użyć konstruktora MQTopic, jak pokazano w poniższym przykładzie:

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
```

Ta instrukcja tworzy obiekt MQTopic z wartościami domyślnymi dla wszystkich jego właściwości. Obiekt reprezentuje temat o nazwie Sport/Stopka/Wyniki.

Konstruktor MQTopic może również zaakceptować identyfikator URI tematu jako parametr. Identyfikator URI tematu jest łańcuchem określającym nazwę tematu i opcjonalnie jedną lub więcej właściwości obiektu MQTopic. Poniższa instrukcja zawiera przykład identyfikatora URI tematu:

```
MQTopic t2 = new MQTopic("topic://Sport/Tennis/Results?persistence=1&priority=0");
```

Obiekt MQTopic utworzony przez tę instrukcję reprezentuje temat o nazwie Sport/Tennis/Results, a wszystkie komunikaty wysyłane do tego miejsca docelowego są nietrwałe i mają priorytet 0. Więcej informacji na temat identyfikatorów URI tematów zawiera sekcja “Identyfikatory URI (Uniform Resource Identifier)” na stronie 229. Alternatywny sposób ustawiania właściwości obiektu MQTopic zawiera sekcja “Ustawianie właściwości miejsc docelowych” na stronie 223.

## Ustawianie właściwości miejsc docelowych

Aplikacja może ustawić właściwości miejsca docelowego, wywołując odpowiednie metody miejsca docelowego. Miejscem docelowym może być obiekt administrowany lub obiekt tworzony dynamicznie w czasie wykonywania.

Rozważmy następujący kod, na przykład:

```
MQQueue q1 = new MQQueue("Q1");
.
q1.setPersistence(WMQConstants.WMQ_PER_PER);
q1.setPriority(5);
```

Ten kod tworzy obiekt MQQueue, a następnie ustawia dwie właściwości obiektu. Efektem ustawienia tych właściwości jest to, że wszystkie komunikaty wysyłane do miejsca docelowego są trwałe i mają priorytet 5.

Aplikacja może ustawić właściwości obiektu MQTopic w podobny sposób, jak pokazano w poniższym przykładzie:

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
.
t1.setPersistence(WMQConstants.WMQ_PER_NON);
t1.setPriority(0);
```

Ten kod tworzy obiekt MQTopic, a następnie ustawia dwie właściwości obiektu. Efektem ustawienia tych właściwości jest to, że wszystkie komunikaty wysyłane do miejsca docelowego są nietrwałe i mają priorytet 0.

Więcej informacji na temat właściwości miejsca docelowego oraz metod używanych do ustawiania jego właściwości zawiera sekcja Właściwości obiektów IBM MQ classes for JMS.

### Linux AIX *Nawiązywanie połączenia z produktem IBM MQ z aplikacji JMS*

Aby zbudować połączenie, aplikacja JMS używa obiektu **ConnectionFactory** do utworzenia obiektu **Connection** , a następnie uruchamia połączenie.

W przypadku usługi JMS 2.0 i nowszych aplikacje zwykle łączą się z dostawcą przesyłania komunikatów przy użyciu obiektu **ConnectionFactory** i metody `createContext()` .

We wcześniejszych wersjach usługi JMS należało najpierw użyć funkcji `createConnection` do utworzenia obiektu **Connection** , a następnie uruchomić wywołanie połączenia `getSession()` w celu utworzenia obiektu **Session** , który może wykonywać operacje przesyłania komunikatów.

Obiekt **JMSContext** w rzeczywistości obudowuje zarówno obiekty **Connection** , jak i **Session** . Aby użyć tradycyjnego podejścia i utworzyć bezpośrednio połączenie i obiekty sesji, należy zapoznać się z informacjami w sekcji “Budowanie połączenia w aplikacji JMS” na stronie 224 i “Tworzenie sesji w aplikacji JMS” na stronie 225.

Aby utworzyć obiekt **JMSContext** , aplikacja używa metody `createContext()` obiektu **ConnectionFactory** , jak pokazano w poniższym przykładzie:

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createContext();
```

Po utworzeniu połączenia JMS program IBM MQ classes for JMS tworzy uchwyt połączenia (Hconn) i rozpoczyna konwersację z menedżerem kolejek.

**Uwaga:** Należy zauważyć, że identyfikator procesu aplikacji jest używany jako domyślna tożsamość użytkownika, która ma zostać przekazana do menedżera kolejek. Jeśli aplikacja działa w trybie transportu klienta, ten identyfikator procesu musi istnieć na serwerze z odpowiednimi autoryzacjami. Aby użyć innej tożsamości, należy użyć metody `createConnection(username,password)` .

V 9.3.5 Ten mechanizm może być również używany do dostarczania znacznika uwierzytelniania. Więcej informacji na ten temat zawiera sekcja Uzyskiwanie znacznika uwierzytelniania od wybranego wystawcy znacznika.

JMS 1.0 *Budowanie połączenia w aplikacji JMS*

Aby zbudować połączenie w produkcie JMS 1.0, aplikacja JMS używa obiektu ConnectionFactory do utworzenia obiektu Connection, a następnie uruchamia połączenie.

Aby utworzyć obiekt połączenia, aplikacja używa metody createConnection() obiektu ConnectionFactory , jak pokazano w poniższym przykładzie:

```
ConnectionFactory factory;
Connection connection;
.
.
```

```
.
connection = factory.createConnection();
```

Po utworzeniu połączenia JMS program IBM MQ classes for JMS tworzy uchwyt połączenia (Hconn) i rozpoczyna konwersację z menedżerem kolejek.

Interfejs fabryki QueueConnectioni interfejs fabryki TopicConnectiondziedziczą metodę createConnection() z interfejsu ConnectionFactory . W związku z tym do utworzenia obiektu specyficznego dla domeny można użyć metody createConnection(), jak pokazano w poniższym przykładzie:

```
QueueConnectionFactory qcf;
Connection connection;
.
.
.
connection = qcf.createConnection();
```

Ten fragment kodu tworzy obiekt QueueConnection . Aplikacja może teraz wykonać niezależną od domeny operację na tym obiekcie lub operację, która ma zastosowanie tylko do domeny punkt z punktem. Jeśli jednak aplikacja podejmie próbę wykonania operacji, która ma zastosowanie tylko do domeny publikowania/subskrybowania, zostanie zgłoszony wyjątek IllegalStatez następującym komunikatem:

```
JMSMQ1112: Operation for a domain specific object was not valid.
           Operation createProducer() is not valid for type com.ibm.mq.jms.MQTopic
```

Jest to spowodowane tym, że połączenie zostało utworzone na podstawie fabryki połączeń specyficznej dla domeny.

**Uwaga:** Należy zauważyć, że identyfikator procesu aplikacji jest używany jako domyślna tożsamość użytkownika, która ma zostać przekazana do menedżera kolejek. Jeśli aplikacja działa w trybie transportu klienta, ten identyfikator procesu musi istnieć na serwerze z odpowiednimi autoryzacjami. Aby użyć innej tożsamości, należy użyć metody createConnection(nazwa użytkownika, hasło).

Specyfikacja JMS określa, że połączenie jest tworzone w stanie `stopped` . Dopóki połączenie nie zostanie uruchomione, konsument komunikatów powiązany z połączeniem nie może odbierać żadnych komunikatów. Aby uruchomić połączenie, aplikacja używa metody start () obiektu połączenia, jak pokazano w poniższym przykładzie:

```
connection.start();
```

V 9.3.5 Ten mechanizm może być również używany do dostarczania znacznika uwierzytelniania. Więcej informacji na ten temat zawiera sekcja Uzyskiwanie znacznika uwierzytelniania od wybranego wystawcy znacznika.

JMS 1.0 *Tworzenie sesji w aplikacji JMS*

Aby utworzyć sesję w programie JMS 1.0, aplikacja JMS używa metody createSession() obiektu połączenia.

Metoda createSession() ma dwa parametry:

1. Parametr określający, czy sesja jest transakcją, czy nie.
2. Parametr określający tryb potwierdzenia dla sesji

Na przykład poniższy kod tworzy sesję, która nie jest transakcją i ma tryb potwierdzenia AUTO_ACKNOWLEDGE:

```
Session session;
.
boolean transacted = false;
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
```

Po utworzeniu sesji JMS program IBM MQ classes for JMS tworzy uchwyt połączenia (Hconn) i rozpoczyna konwersację z menedżerem kolejek.

Obiekt sesji i dowolny obiekt MessageProducer lub MessageConsumer utworzony na jego podstawie nie mogą być używane współbieżnie przez różne wątki aplikacji wielowątkowej. Najprostszym sposobem zapewnienia, że te obiekty nie są używane współbieżnie, jest utworzenie osobnego obiektu sesji dla każdego wątku.

V 9.3.5 Ten mechanizm może być również używany do dostarczania znacznika uwierzytelniania. Więcej informacji na ten temat zawiera sekcja Uzyskiwanie znacznika uwierzytelniania od wybranego wystawcy znacznika.

*Sesje transakcyjne w aplikacjach JMS*
Aplikacje JMS mogą uruchamiać transakcje lokalne, tworząc najpierw sesję transakcyjne. Aplikacja może zatwierdzić lub wycofać transakcję.

Aplikacje JMS mogą uruchamiać transakcje lokalne. Transakcja lokalna jest transakcją, która obejmuje tylko zmiany zasobów menedżera kolejek, z którym połączona jest aplikacja. Aby uruchomić transakcje lokalne, aplikacja musi najpierw utworzyć sesję transakcją, wywołując metodę createSession() obiektu połączenia, określając jako parametr, że sesja jest transakcją. Następnie wszystkie komunikaty wysyłane i odbierane w ramach sesji są grupowane w sekwencję transakcji. Transakcja kończy się, gdy aplikacja zatwierdza lub wycofuje wysłane i odebrane komunikaty od momentu rozpoczęcia transakcji.

Aby zatwierdzić transakcję, aplikacja wywołuje metodę commit () obiektu Session. Po zatwierdzeniu transakcji wszystkie komunikaty wysłane w ramach transakcji stają się dostępne do dostarczenia do innych aplikacji, a wszystkie komunikaty odebrane w ramach transakcji są potwierdzane, aby serwer przesyłania komunikatów nie próbował ponownie dostarczyć ich do aplikacji. W domenie typu punkt z punktem serwer przesyłania komunikatów usuwa również odebrane komunikaty z ich kolejek.

Aby wycofać transakcję, aplikacja wywołuje metodę rollback () obiektu Session. Po wycofaniu transakcji wszystkie komunikaty wysłane w ramach transakcji są usuwane przez serwer przesyłania komunikatów, a wszystkie komunikaty odebrane w ramach transakcji stają się ponownie dostępne do dostarczenia. W domenie typu punkt z punktem odebrane komunikaty są ponownie umieszczane w kolejkach i stają się ponownie widoczne dla innych aplikacji.

Nowa transakcja jest uruchamiana automatycznie, gdy aplikacja tworzy sesję transakcyjną lub wywołuje metodę commit () lub rollback (). Oznacza to, że sesja transakcyjna zawsze ma aktywną transakcję.

Gdy aplikacja zamyka sesję transakcyjne, następuje niejawne wycofanie zmian. Gdy aplikacja zamyka połączenie, następuje niejawne wycofanie zmian dla wszystkich sesji transakcyjnych połączenia.

Jeśli aplikacja zakończy działanie bez zamknięcia połączenia, nastąpi również niejawne wycofanie zmian dla wszystkich sesji transakcyjnych połączenia.

Transakcja jest w całości zawarta w sesji transakcyjnej. Transakcja nie może obejmować sesji. Oznacza to, że nie jest możliwe, aby aplikacja wysyłała i odbierała komunikaty w dwóch lub większej liczbie sesji transakcyjnych, a następnie zatwierdzała lub wycofywała wszystkie te działania jako pojedynczą transakcję.

*Tryby potwierdzania sesji JMS*
Każda sesja, która nie jest transakcją, ma tryb potwierdzenia, który określa sposób potwierdzania komunikatów odbieranych przez aplikację. Dostępne są trzy tryby potwierdzenia, a wybór trybu potwierdzenia ma wpływ na projekt aplikacji.

Jeśli sesja nie jest transakcją, sposób potwierdzania komunikatów odbieranych przez aplikację jest określany przez tryb potwierdzenia sesji. Trzy tryby potwierdzenia są opisane w następujących akapitach:

**AUTO_ACKNOWLEDGE (potwierdzenie automatyczne)**

Sesja automatycznie potwierdza każdy komunikat odebrany przez aplikację.

Jeśli komunikaty są dostarczane synchronicznie do aplikacji, sesja potwierdza odebranie komunikatu za każdym razem, gdy wywołanie odbioru zakończy się pomyślnie. Jeśli komunikaty są dostarczane

asynchronicznie, sesja potwierdza odebranie komunikatu za każdym razem, gdy wywołanie metody onMessage() obiektu nasłuchiwania komunikatów zakończy się pomyślnie.

Jeśli aplikacja otrzyma komunikat pomyślnie, ale niepowodzenie uniemożliwi potwierdzenie, komunikat ponownie stanie się dostępny do dostarczenia. Dlatego aplikacja musi być w stanie obsłużyć komunikat, który został ponownie dostarczony.

**DUBL_OK_ACKNOWLEDGE**

Sesja potwierdza komunikaty odebrane przez aplikację w wybranych momentach.

Użycie tego trybu potwierdzenia zmniejsza ilość pracy, jaką musi wykonać sesja, ale niepowodzenie, które uniemożliwia potwierdzenie komunikatu, może spowodować ponowne udostępnienie więcej niż jednego komunikatu do dostarczenia. Dlatego aplikacja musi być w stanie obsłużyć ponownie dostarczane komunikaty.

**Ograniczenie:** W trybach AUTO_ACKNOWLEDGE i DUPS_OK_ACKNOWLEDGE program JMS nie obsługuje aplikacji zgłaszającej nieobsługiwany wyjątek w obiekcie nasłuchiwania komunikatów. Oznacza to, że komunikaty są zawsze potwierdzane po powrocie z programu nasłuchującego komunikatów, niezależnie od tego, czy zostały przetworzone pomyślnie (pod warunkiem, że jakiekolwiek niepowodzenia nie są krytyczne i nie uniemożliwiają kontynuowania działania aplikacji). Jeśli wymagane jest dokładniejsze sterowanie potwierdzaniem komunikatów, należy użyć trybu CLIENT_ACKNOWLEDGE lub trybu transakcyjnego, który daje aplikacji pełną kontrolę nad funkcjami potwierdzania.

**POTWIERDZ_KLIENTA**

Aplikacja potwierdza odebrane komunikaty, wywołując metodę Acknowledge klasy Message.

Aplikacja może potwierdzić odbiór każdego komunikatu osobno lub może odebrać partię komunikatów i wywołać metodę potwierdzania tylko dla ostatniego komunikatu, który otrzymuje. Po wywołaniu metody Acknowledge wszystkie komunikaty odebrane od czasu ostatniego wywołania metody są potwierdzane.

W połączeniu z dowolnym z tych trybów potwierdzania aplikacja może zatrzymać i zrestartować dostarczanie komunikatów w sesji, wywołując metodę odtwarzania klasy Session. Komunikaty odebrane, ale wcześniej niepotwierdzone, są ponownie dostarczane. Mogą one jednak nie być dostarczane w tej samej kolejności, w jakiej zostały wcześniej dostarczone. W międzyczasie mogły pojawić się komunikaty o wyższym priorytecie, a niektóre z oryginalnych komunikatów mogły utracić ważność. W domenie typu punkt z punktem niektóre oryginalne komunikaty mogły zostać wykorzystane przez inną aplikację.

Aplikacja może określić, czy komunikat jest ponownie dostarczany, sprawdzając treść pola nagłówka JMSRedelivered komunikatu. W tym celu aplikacja wywołuje metodę getJMSRedelivered() klasy Message.

### *Tworzenie miejsc docelowych w aplikacji JMS*

Zamiast pobierać miejsca docelowe jako obiekty administrowane z przestrzeni nazw JNDI ( Java Naming and Directory Interface), aplikacja JMS może używać sesji do dynamicznego tworzenia miejsc docelowych w czasie wykonywania. Aplikacja może używać identyfikatora URI (uniform resource identifier) do identyfikowania kolejki IBM MQ lub tematu oraz opcjonalnie do określania jednej lub większej liczby właściwości obiektu kolejki lub tematu.

## Używanie sesji do tworzenia obiektów kolejki

Aby utworzyć obiekt kolejki, aplikacja może użyć metody createQueue() obiektu sesji, jak pokazano w poniższym przykładzie:

```
Session session;
.
Queue q1 = session.createQueue("Q1");
```

Ten kod tworzy obiekt kolejki z wartościami domyślnymi dla wszystkich jego właściwości. Obiekt reprezentuje kolejkę IBM MQ o nazwie Q1 , która należy do lokalnego menedżera kolejek. Ta kolejka może być kolejką lokalną, kolejką aliasową lub definicją kolejki zdalnej.

Metoda createQueue() akceptuje również identyfikator URI kolejki jako parametr. Identyfikator URI kolejki to łańcuch określający nazwę kolejki produktu IBM MQ i opcjonalnie nazwę menedżera kolejek będącego właścicielem kolejki oraz co najmniej jedną właściwość obiektu kolejki. Poniższa instrukcja zawiera przykład identyfikatora URI kolejki:

```
Queue q2 = session.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

Obiekt kolejki utworzony przez tę instrukcję reprezentuje kolejkę IBM MQ o nazwie Q2 , której właścicielem jest menedżer kolejek o nazwie QM2, a wszystkie komunikaty wysyłane do tego miejsca docelowego są trwałe i mają priorytet 5. Zidentyfikowany w ten sposób menedżer kolejek może być lokalnym lub zdalnym menedżerem kolejek. Jeśli jest to zdalny menedżer kolejek, produkt IBM MQ musi być skonfigurowany w taki sposób, aby podczas wysyłania komunikatu przez aplikację do tego miejsca docelowego produkt WebSphere MQ mógł kierować komunikat z lokalnego menedżera kolejek do menedżera kolejek QM2. Więcej informacji na temat identyfikatorów URI zawiera sekcja “Identyfikatory URI (Uniform Resource Identifier)” na stronie 229.

Należy zauważyć, że parametr w metodzie createQueue() zawiera informacje specyficzne dla dostawcy. Dlatego użycie metody createQueue() do utworzenia obiektu kolejki zamiast pobierania obiektu kolejki jako obiektu administrowanego z przestrzeni nazw JNDI może spowodować, że aplikacja będzie mniej przenośna.

Aplikacja może utworzyć obiekt TemporaryQueue , używając metody createTemporaryQueue () obiektu Session, jak pokazano w poniższym przykładzie:

```
TemporaryQueue q3 = session.createTemporaryQueue();
```

Chociaż sesja jest używana do tworzenia kolejki tymczasowej, zasięgiem kolejki tymczasowej jest połączenie, które zostało użyte do utworzenia sesji. Każda sesja połączenia może utworzyć producentów komunikatów i konsumentów komunikatów dla kolejki tymczasowej. Kolejka tymczasowa pozostaje do momentu zakończenia połączenia lub jawnego usunięcia kolejki tymczasowej przez aplikację przy użyciu metody TemporaryQueue.delete (), w zależności od tego, co nastąpi wcześniej.

Gdy aplikacja tworzy kolejkę tymczasową, program IBM MQ classes for JMS tworzy kolejkę dynamiczną w menedżerze kolejek, z którym aplikacja jest połączona. Właściwość TEMPMODEL fabryki połączeń określa nazwę kolejki modelowej używanej do tworzenia kolejki dynamicznej, a właściwość TEMPQPREFIX fabryki połączeń określa przedrostek używany do tworzenia nazwy kolejki dynamicznej.

## Używanie sesji do tworzenia obiektów tematu

Aby utworzyć obiekt Topic, aplikacja może użyć metody createTopic() obiektu Session, jak pokazano w poniższym przykładzie:

```
Session session;
.
Topic t1 = session.createTopic("Sport/Football/Results");
```

Ten kod tworzy obiekt Topic z wartościami domyślnymi dla wszystkich jego właściwości. Obiekt reprezentuje temat o nazwie Sport/Stopka/Wyniki.

Metoda createTopic() akceptuje również identyfikator URI tematu jako parametr. Identyfikator URI tematu to łańcuch określający nazwę tematu i opcjonalnie jedną lub więcej właściwości obiektu tematu. Poniższy kod zawiera przykład identyfikatora URI tematu:

```
String uri = "topic://Sport/Tennis/Results?persistence=1&priority=0";
Topic t2 = session.createTopic(uri);
```

Obiekt Topic utworzony przez ten kod reprezentuje temat o nazwie Sport/Tennis/Results, a wszystkie komunikaty wysyłane do tego miejsca docelowego są nietrwałe i mają priorytet 0. Więcej informacji na temat identyfikatorów URI tematów zawiera sekcja “Identyfikatory URI (Uniform Resource Identifier)” na stronie 229.

Należy zauważyć, że parametr metody createTopic() zawiera informacje specyficzne dla dostawcy. Dlatego użycie metody createTopic() do utworzenia obiektu Topic zamiast pobierania obiektu Topic jako obiektu administrowanego z przestrzeni nazw JNDI może spowodować, że aplikacja będzie mniej przenośna.

Aplikacja może utworzyć obiekt TemporaryTopic , używając metody createTemporaryTopic () obiektu Session, jak pokazano w poniższym przykładzie:

```
TemporaryTopic t3 = session.createTemporaryTopic();
```

Chociaż sesja jest używana do tworzenia tematu tymczasowego, zasięgiem tematu tymczasowego jest połączenie, które zostało użyte do utworzenia sesji. Każda sesja połączenia może utworzyć producentów komunikatów i konsumentów komunikatów dla tematu tymczasowego. Temat tymczasowy pozostaje do momentu zakończenia połączenia lub jawnego usunięcia tematu tymczasowego przez aplikację przy użyciu metody TemporaryTopic.delete (), w zależności od tego, co nastąpi wcześniej.

Gdy aplikacja tworzy temat tymczasowy, program IBM MQ classes for JMS tworzy temat o nazwie rozpoczynającej się od znaków TEMP/*tempTopicprzedrostek*, gdzie *tempTopicprzedrostek* jest wartością właściwości TEMPTOPICPREFIX fabryki połączeń.

## Identyfikatory URI (Uniform Resource Identifier)

Identyfikator URI kolejki to łańcuch określający nazwę kolejki produktu IBM MQ i opcjonalnie nazwę menedżera kolejek będącego właścicielem kolejki oraz co najmniej jedną właściwość obiektu kolejki utworzonego przez aplikację. Identyfikator URI tematu to łańcuch określający nazwę tematu i opcjonalnie jedną lub więcej właściwości obiektu tematu utworzonego przez aplikację.

Identyfikator URI kolejki ma następujący format:

```
queue://[ qMgrName ]/qName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

Identyfikator URI tematu ma następujący format:

```
topic://topicName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

Zmienne w tych formatach mają następujące znaczenie:

***qMgrNazwa***
Nazwa menedżera kolejek, który jest właścicielem kolejki identyfikowanej przez identyfikator URI.

Menedżer kolejek może być lokalnym lub zdalnym menedżerem kolejek. Jeśli jest to zdalny menedżer kolejek, program IBM MQ musi być skonfigurowany w taki sposób, aby podczas wysyłania komunikatu przez aplikację do kolejki produkt WebSphere MQ mógł kierować komunikat z lokalnego menedżera kolejek do zdalnego menedżera kolejek.

Jeśli nie zostanie podana żadna nazwa, przyjmowany jest menedżer kolejek lokalnych.

***qName***
Nazwa kolejki produktu IBM MQ .

Kolejka może być kolejką lokalną, kolejką aliasową lub definicją kolejki zdalnej.

Reguły tworzenia nazw kolejek zawiera sekcja Reguły nazewnictwa obiektów IBM MQ.

***topicName***
Nazwa tematu.

Reguły tworzenia nazw tematów zawiera sekcja Reguły nazewnictwa obiektów IBM MQ. Należy unikać stosowania znaków wieloznacznych +, #, * i? w nazwach tematów. Nazwy tematów zawierające te

znaki mogą spowodować nieoczekiwane rezultaty po ich zasubskrybowaniu. Patrz sekcja Łączenie łańcuchów tematu.

***propertyName1*, *propertyName2*, ...**
Nazwy właściwości obiektu kolejki lub tematu utworzonych przez aplikację. Tabela 35 na stronie 230 zawiera listę poprawnych nazw właściwości, które mogą być używane w identyfikatorze URI.

Jeśli nie określono żadnych właściwości, obiekt Queue lub Topic ma wartości domyślne dla wszystkich swoich właściwości.

***propertyValue1*, *propertyValue2*, ...**
Wartości właściwości obiektu kolejki lub tematu utworzonego przez aplikację. Tabela 35 na stronie 230 zawiera listę poprawnych wartości właściwości, których można użyć w identyfikatorze URI.

Nawiasy kwadratowe ([]) oznaczają komponent opcjonalny, a wielokropek (...) oznacza, że lista par nazwa-wartość właściwości, jeśli istnieje, może zawierać jedną lub więcej par nazwa-wartość.

Tabela 35 na stronie 230 zawiera listę poprawnych nazw właściwości i poprawnych wartości, które mogą być używane w identyfikatorach URI kolejek i tematów. Chociaż narzędzie administracyjne IBM MQ JMS używa stałych symbolicznych dla wartości właściwości, identyfikatory URI nie mogą zawierać stałych symbolicznych.

*Tabela 35. Nazwy właściwości i poprawne wartości do użycia w identyfikatorach URI kolejek i tematów*

| **Nazwa właściwości** | **Opis** | **Poprawne wartości** |
|---|---|---|
| CCSID | Sposób reprezentowania danych znakowych w treści komunikatu, gdy produkt IBM MQ classes for JMS przekazuje komunikat do miejsca docelowego. | • Dowolny identyfikator kodowanego zestawu znaków obsługiwany przez IBM MQ. |
| kodowanie | Sposób reprezentowania danych liczbowych w treści komunikatu, gdy produkt IBM MQ classes for JMS przekazuje komunikat do miejsca docelowego. | • Dowolna poprawna wartość w polu *Kodowanie* w deskryptorze komunikatu IBM MQ . |
| Koniec ważności | Czas życia komunikatów wysłanych do miejsca docelowego | • -2-Jak określono w wywołaniu send () lub, jeśli nie określono w wywołaniu send (), domyślny czas życia producenta komunikatu.<br>• 0-komunikat wysłany do miejsca docelowego nigdy nie traci ważności.<br>• Dodatnia liczba całkowita określająca czas życia w milisekundach. |

*Tabela 35. Nazwy właściwości i poprawne wartości do użycia w identyfikatorach URI kolejek i tematów (kontynuacja)*

| **Nazwa właściwości** | **Opis** | **Poprawne wartości** |
|---|---|---|
| rozsyłanie | Ustawienie rozsyłania grupowego dla tematu w przypadku korzystania z połączenia z brokerem w czasie rzeczywistym | Poniższa lista zawiera poprawne wartości. Z każdą wartością powiązana jest odpowiednia wartość właściwości MULTICAST używana w narzędziu administracyjnym IBM MQ JMS . Opis właściwości MULTICAST i jej poprawnych wartości zawiera sekcja Właściwości obiektów IBM MQ classes for JMS.<br>• -1-ASCF (system kaskadowy)<br>• 0 - wyłączone<br>• 3-NOTR<br>• 5-NIEZAWODNY<br>• 7-WŁĄCZONE |
| trwałość | Trwałość komunikatów wysyłanych do miejsca docelowego | • -2-Jak określono w wywołaniu send () lub, jeśli nie określono w wywołaniu send (), domyślna trwałość producenta komunikatu.<br>• -1-zgodnie z określeniem w atrybucie *DefPersistence* kolejki lub tematu IBM MQ .<br>• 1-Nietrwały.<br>• 2-Trwałe.<br>• 3-odpowiednik wartości HIGH właściwości PERSISTENCE używanej w narzędziu administracyjnym IBM MQ JMS . Wyjaśnienie tej wartości zawiera sekcja “Trwałe komunikaty JMS” na stronie 260. |
| priorytet | Priorytet komunikatów wysyłanych do miejsca docelowego | • -2-Zgodnie z określeniem w wywołaniu send () lub, jeśli nie określono w wywołaniu send (), domyślnym priorytetem producenta komunikatu.<br>• -1-zgodnie z określeniem w atrybucie *DefPriority* kolejki lub tematu IBM MQ .<br>• Liczba całkowita z zakresu od 0 do 9 określająca priorytet komunikatów wysyłanych do miejsca docelowego. |
| targetClient | Określa, czy komunikaty wysyłane do miejsca docelowego zawierają nagłówek MQRFH2 . | • 0-komunikaty zawierają nagłówek MQRFH2 .<br>• 1-komunikaty nie zawierają nagłówka MQRFH2 . |

Na przykład następujący identyfikator URI identyfikuje kolejkę IBM MQ o nazwie Q1 , której właścicielem jest lokalny menedżer kolejek. Obiekt kolejki utworzony za pomocą tego identyfikatora URI ma wartości domyślne dla wszystkich jego właściwości.

```
queue:///Q1
```

Poniższy identyfikator URI identyfikuje kolejkę IBM MQ o nazwie Q2 , której właścicielem jest menedżer kolejek o nazwie QM2. Wszystkie komunikaty wysłane do tego miejsca docelowego mają priorytet 6. Pozostałe właściwości obiektu kolejki utworzonego przy użyciu tego identyfikatora URI mają wartości domyślne.

```
queue://QM2/Q2?priority=6
```

Następujący identyfikator URI identyfikuje temat o nazwie Sport/Athletics/Results. Wszystkie komunikaty wysłane do tego miejsca docelowego są nietrwałe i mają priorytet 0. Pozostałe właściwości obiektu Topic utworzonego przy użyciu tego identyfikatora URI mają wartości domyślne.

```
topic://Sport/Athletics/Results?persistence=1&priority=0
```

### *Wysyłanie komunikatów w aplikacji JMS*

Zanim aplikacja JMS będzie mogła wysyłać komunikaty do miejsca docelowego, musi najpierw utworzyć obiekt MessageProducer dla miejsca docelowego. Aby wysłać komunikat do miejsca docelowego, aplikacja tworzy obiekt Message, a następnie wywołuje metodę send () obiektu MessageProducer .

Aplikacja używa obiektu MessageProducer do wysyłania komunikatów. Aplikacja zwykle tworzy obiekt MessageProducer dla konkretnego miejsca docelowego, który może być kolejką lub tematem, aby wszystkie komunikaty wysyłane za pomocą tego producenta komunikatów były wysyłane do tego samego miejsca docelowego. Dlatego zanim aplikacja będzie mogła utworzyć obiekt MessageProducer , musi najpierw utworzyć obiekt Queue lub Topic. Informacje na temat tworzenia obiektu kolejki lub tematu znajdują się w następujących tematach:

- “Używanie interfejsu JNDI do pobierania obiektów administrowanych w aplikacji JMS lub Jakarta Messaging” na stronie 212
- “Korzystanie z rozszerzeń IBM JMS” na stronie 214
- “Korzystanie z rozszerzeń IBM MQ JMS” na stronie 221
- “Tworzenie miejsc docelowych w aplikacji JMS” na stronie 227

Aby utworzyć obiekt MessageProducer , aplikacja używa metody createProducer() obiektu Session, jak pokazano w poniższym przykładzie:

```
MessageProducer producer = session.createProducer(destination);
```

Parametr `destination` jest obiektem kolejki lub tematu, który został wcześniej utworzony przez aplikację.

Zanim aplikacja będzie mogła wysłać komunikat, musi utworzyć obiekt Message. Treść komunikatu zawiera dane aplikacji, a JMS definiuje pięć typów treści komunikatu:

- Bajty
- Odwzoruj
- Obiekt
- Strumień
- Tekst

Każdy typ treści komunikatu ma własny interfejs JMS , który jest podinterfejsem interfejsu komunikatu, oraz metodę w interfejsie sesji służącą do tworzenia komunikatu o tym typie treści. Na przykład interfejs

dla komunikatu tekstowego nosi nazwę TextMessage, a aplikacja używa metody createTextMessage () obiektu Session w celu utworzenia komunikatu tekstowego, jak pokazano w następującej instrukcji:

```
TextMessage outMessage = session.createTextMessage(outString);
```

Więcej informacji na temat komunikatów i treści komunikatów zawiera sekcja “Komunikaty produktu JMS” na stronie 149.

Aby wysłać komunikat, aplikacja używa metody send () obiektu MessageProducer , jak pokazano w poniższym przykładzie:

```
producer.send(outMessage);
```

Aplikacja może używać metody send () do wysyłania komunikatów w dowolnej domenie przesyłania komunikatów. Rodzaj miejsca docelowego określa, która domena przesyłania komunikatów jest używana. Jednak interfejs podrzędny TopicPublisherinterfejsu MessageProducer , który jest specyficzny dla domeny publikowania/subskrypcji, ma również metodę publish (), która może być używana zamiast metody send (). Te dwie metody są funkcjonalnie takie same.

Aplikacja może utworzyć obiekt MessageProducer bez określonego miejsca docelowego. W tym przypadku aplikacja musi określić miejsce docelowe podczas wywoływania metody send ().

Jeśli aplikacja wysyła komunikat w ramach transakcji, komunikat nie jest dostarczany do miejsca docelowego, dopóki transakcja nie zostanie zatwierdzona. Oznacza to, że aplikacja nie może wysłać komunikatu i odebrać odpowiedzi na komunikat w ramach tej samej transakcji.

Miejsce docelowe można skonfigurować w taki sposób, aby podczas wysyłania do niego komunikatów przez aplikację produkt IBM MQ classes for JMS przekazywał komunikat i zwracał sterowanie z powrotem do aplikacji bez określania, czy menedżer kolejek otrzymał komunikat bezpiecznie. Jest to czasami nazywane *umieszczaniem asynchronicznym*. Więcej informacji na ten temat zawiera sekcja “Asynchroniczne umieszczanie komunikatów w produkcie IBM MQ classes for JMS” na stronie 329.

### *Odbieranie komunikatów w aplikacji JMS*

Aplikacja używa konsumenta komunikatów do odbierania komunikatów. Trwały subskrybent tematu jest konsumentem komunikatów, który odbiera wszystkie komunikaty wysyłane do miejsca docelowego, w tym te, które są wysyłane, gdy konsument jest nieaktywny. Aplikacja może wybrać komunikaty, które ma odbierać przy użyciu selektora komunikatów, a także może odbierać komunikaty asynchronicznie przy użyciu obiektu nasłuchiwania komunikatów.

Aplikacja używa obiektu MessageConsumer do odbierania komunikatów. Aplikacja tworzy obiekt MessageConsumer dla konkretnego miejsca docelowego, który może być kolejką lub tematem, aby wszystkie komunikaty odebrane przy użyciu konsumenta komunikatów były odbierane z tego samego miejsca docelowego. Dlatego zanim aplikacja będzie mogła utworzyć obiekt MessageConsumer , musi najpierw utworzyć obiekt Queue lub Topic. Informacje na temat tworzenia obiektu kolejki lub tematu znajdują się w następujących tematach:

- “Używanie interfejsu JNDI do pobierania obiektów administrowanych w aplikacji JMS lub Jakarta Messaging” na stronie 212
- “Korzystanie z rozszerzeń IBM JMS” na stronie 214
- “Korzystanie z rozszerzeń IBM MQ JMS” na stronie 221
- “Tworzenie miejsc docelowych w aplikacji JMS” na stronie 227

Aby utworzyć obiekt MessageConsumer , aplikacja używa metody createConsumer() obiektu Session, jak pokazano w poniższym przykładzie:

```
MessageConsumer consumer = session.createConsumer(destination);
```

Parametr `destination` jest obiektem kolejki lub tematu, który został wcześniej utworzony przez aplikację.

Następnie aplikacja używa metody receive () obiektu MessageConsumer do odbierania komunikatu z miejsca docelowego, jak pokazano w poniższym przykładzie:

```
Message inMessage = consumer.receive(1000);
```

Parametr w wywołaniu receive () określa, jak długo (w milisekundach) metoda oczekuje na nadejście odpowiedniego komunikatu, jeśli żaden komunikat nie jest natychmiast dostępny. Jeśli ten parametr zostanie pominięty, wywołanie zostanie zablokowane na czas nieokreślony do momentu pojawienia się odpowiedniego komunikatu. Jeśli aplikacja nie ma oczekiwać na komunikat, należy użyć metody Wait () receiveNo.

Metoda receive () zwraca komunikat określonego typu. Na przykład, gdy aplikacja odbiera komunikat tekstowy, obiektem zwróconym przez wywołanie metody receive () jest obiekt TextMessage .

Jednak zadeklarowany typ obiektu zwracany przez wywołanie metody receive () jest obiektem komunikatu. Dlatego w celu wyodrębnienia danych z treści odebranego komunikatu aplikacja musi rzutować z klasy Message do bardziej konkretnej podklasy, takiej jak TextMessage. Jeśli typ komunikatu nie jest znany, aplikacja może użyć operatora `instanceof` do określenia typu. Zaleca się, aby aplikacja zawsze określała typ komunikatu przed rzutowaniem, tak aby błędy mogły być obsługiwane poprawnie.

W poniższym kodzie użyto operatora `instanceof` i przedstawiono sposób wyodrębniania danych z treści komunikatu tekstowego:

```
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");
}
```

Jeśli aplikacja wysyła komunikat w ramach transakcji, komunikat nie jest dostarczany do miejsca docelowego, dopóki transakcja nie zostanie zatwierdzona. Oznacza to, że aplikacja nie może wysłać komunikatu i odebrać odpowiedzi na komunikat w ramach tej samej transakcji.

Jeśli konsument komunikatów odbiera komunikaty z miejsca docelowego skonfigurowanego do odczytu z wyprzedzeniem, wszystkie nietrwałe komunikaty znajdujące się w buforze odczytu z wyprzedzeniem po zakończeniu działania aplikacji są usuwane.

W domenie publikowania/subskrypcji produkt JMS identyfikuje dwa typy konsumentów komunikatów: nietrły subskrybent tematu i trwały subskrybent tematu, które zostały opisane w poniższych dwóch sekcjach.

## Nietrwałe subskrybenty tematów

Nietrwały subskrybent tematu odbiera tylko te komunikaty, które są publikowane, gdy subskrybent jest aktywny. Subskrypcja nietrwała jest uruchamiana, gdy aplikacja tworzy nietrwałego subskrybenta tematu i kończy się, gdy aplikacja zamyka subskrybent lub gdy subskrybent wykracza poza zasięg. Jako rozszerzenie w produkcie IBM MQ classes for JMS, nietrły subskrybent tematu również odbiera zachowane publikacje.

Aby utworzyć nietrwałego subskrybenta tematów, aplikacja może użyć niezależnej od domeny metody createConsumer(), określając obiekt Topic (temat) jako miejsce docelowe. Alternatywnie aplikacja może użyć metody createSubscriber() specyficznej dla domeny, jak pokazano w poniższym przykładzie:

```
TopicSubscriber subscriber = session.createSubscriber(topic);
```

Parametr `topic` jest obiektem tematu, który został wcześniej utworzony przez aplikację.

## Trwałe subskrybenci tematu

**Ograniczenie:** Aplikacja nie może tworzyć trwałych subskrybentów tematów podczas używania połączenia w czasie rzeczywistym z brokerem.

Trwały subskrybent tematu odbiera wszystkie komunikaty publikowane w czasie trwania trwałej subskrypcji. Te komunikaty obejmują wszystkie te, które zostały opublikowane, gdy subskrybent nie był aktywny. Jako rozszerzenie w produkcie IBM MQ classes for JMS, trwały subskrybent tematu również odbiera zachowane publikacje.

Aby utworzyć trwały subskrybent tematu, aplikacja używa metody createDurableobiektu Session, jak to pokazano w poniższym przykładzie:

```
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001");
```

W wywołaniu metody createDurableSubscriber () pierwszy parametr to obiekt Topic utworzony wcześniej przez aplikację, a drugi parametr to nazwa używana do zidentyfikowania trwałej subskrypcji.

Sesja używana do tworzenia trwałego subskrybenta tematów musi mieć powiązany identyfikator klienta. Identyfikator klienta powiązany z sesją jest taki sam, jak identyfikator klienta dla połączenia używanego do utworzenia sesji. Identyfikator klienta można określić, ustawiając właściwość CLIENTID obiektu ConnectionFactory . Alternatywnie aplikacja może określić identyfikator klienta, wywołując metodę setClientID () obiektu połączenia.

Nazwa używana do identyfikowania trwałej subskrypcji musi być unikalna tylko w obrębie identyfikatora klienta, dlatego identyfikator klienta stanowi część pełnego, unikalnego identyfikatora trwałej subskrypcji. Aby kontynuować korzystanie z trwałej subskrypcji, która została wcześniej utworzona, aplikacja musi utworzyć trwałego subskrybenta tematów przy użyciu sesji o tym samym identyfikatorze klienta, który jest powiązany z subskrypcją trwałą, i przy użyciu tej samej nazwy subskrypcji.

Subskrypcja trwała rozpoczyna się, gdy aplikacja tworzy trwałego subskrybenta tematu przy użyciu identyfikatora klienta i nazwy subskrypcji, dla których obecnie nie istnieje subskrypcja trwała. Jednak subskrypcja trwała nie kończy się, gdy aplikacja zamyka trwały subskrybent tematu. Aby zakończyć trwałą subskrypcję, aplikacja musi wywołać metodę unsubscribe () obiektu Session, który ma ten sam identyfikator klienta, co powiązany z trwałą subskrypcją. Parametr w wywołaniu unsubscribe () jest nazwą subskrypcji, jak pokazano w poniższym przykładzie:

```
session.unsubscribe("D_SUB_000001");
```

Zasięgiem trwałej subskrypcji jest menedżer kolejek. Jeśli w jednym menedżerze kolejek istnieje subskrypcja trwała, a aplikacja połączona z innym menedżerem kolejek tworzy subskrypcję trwałą o takim samym identyfikatorze klienta i nazwie subskrypcji, dwie subskrypcje trwałe są całkowicie niezależne.

## Selektory komunikatów

Aplikacja może określić, że tylko te komunikaty, które spełniają określone kryteria, są zwracane przez kolejne wywołania funkcji receive (). Podczas tworzenia obiektu MessageConsumer aplikacja może określić wyrażenie SQL (Structured Query Language), które określa pobierane komunikaty. To wyrażenie SQL jest nazywane *selektorem komunikatów*. Selektor komunikatów może zawierać nazwy pól nagłówka komunikatu JMS i właściwości komunikatu. Informacje na temat tworzenia selektora komunikatów zawiera sekcja “Selektory komunikatów w produkcie JMS” na stronie 150.

W poniższym przykładzie przedstawiono, w jaki sposób aplikacja może wybrać komunikaty na podstawie właściwości zdefiniowanej przez użytkownika o nazwie myProp:

```
MessageConsumer consumer;
.
consumer = session.createConsumer(destination, "myProp = 'blue'");
```

Specyfikacja JMS nie zezwala aplikacji na zmianę selektora komunikatów konsumenta komunikatów. Po utworzeniu przez aplikację konsumenta komunikatów z selektorem komunikatów selektor komunikatów

pozostaje przez cały czas życia tego konsumenta. Jeśli aplikacja wymaga więcej niż jednego selektora komunikatów, musi utworzyć konsument komunikatów dla każdego selektora komunikatów.

Należy zauważyć, że gdy aplikacja jest połączona z menedżerem kolejek w wersji 7, właściwość MSGSELECTION fabryki połączeń nie ma wpływu. Aby zoptymalizować wydajność, wszystkie komunikaty są wybierane przez menedżera kolejek.

## Blokowanie publikacji lokalnych

Aplikacja może utworzyć konsument komunikatów, który ignoruje publikacje opublikowane we własnym połączeniu konsumenta. W tym celu aplikacja ustawia trzeci parametr wywołania metody createConsumer() na wartość `true`, jak pokazano w poniższym przykładzie:

```
MessageConsumer consumer = session.createConsumer(topic, null, true);
```

W przypadku wywołania metody createDurableSubscriber () aplikacja robi to, ustawiając czwarty parametr na wartość `true`, jak pokazano w poniższym przykładzie.

```
String selector = "company = 'IBM'";
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001",
                                                              selector, true);
```

## Asynchroniczne dostarczanie komunikatów

Aplikacja może odbierać komunikaty asynchronicznie, rejestrując obiekt nasłuchiwania komunikatów w konsumencie komunikatów. Obiekt nasłuchiwania komunikatów ma metodę o nazwie onMessage, która jest wywoływana asynchronicznie, gdy dostępny jest odpowiedni komunikat i której celem jest przetworzenie komunikatu. Poniższy kod ilustruje mechanizm:

V 9.3.0 JM 3.0 V 9.3.0

```
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
```

Aplikacja może używać sesji do synchronicznego odbierania komunikatów przy użyciu wywołań funkcji receive () lub do asynchronicznego odbierania komunikatów przy użyciu funkcji nasłuchiwania komunikatów, ale nie do obu tych funkcji. Jeśli aplikacja musi odbierać komunikaty synchronicznie i asynchronicznie, musi utworzyć osobne sesje.

Po skonfigurowaniu sesji do odbierania komunikatów w trybie asynchronicznym nie można wywoływać następujących metod dla tej sesji ani dla obiektów utworzonych z tej sesji:

- MessageConsumer.receive ()
- MessageConsumer.receive (długa)
- MessageConsumer.receiveNoWait ()
- Session.acknowledge()
- MessageProducer.send (miejsce docelowe, komunikat)
- MessageProducer.send (miejsce docelowe, komunikat, int, int, long)
- MessageProducer.send (komunikat)
- MessageProducer.send (komunikat, int, int, long)
- MessageProducer.send (miejsce docelowe, komunikat, CompletionListener)
- MessageProducer.send (miejsce docelowe, komunikat, int, int, long, CompletionListener)
- MessageProducer.send (komunikat, CompletionListener)
- MessageProducer.send (Message, int, int, long, CompletionListener)
- Session.commit()
- Session.createBrowser(Kolejka)
- Session.createBrowser(kolejka, łańcuch)
- Session.createBytesMessage()
- Session.createConsumer(Miejsce docelowe)
- Session.createConsumer(miejsce docelowe, łańcuch, wartość boolowska)
- Session.createDurableSubscriber(Temat, Łańcuch)
- Session.createDurableSubscriber(Temat, Łańcuch, Łańcuch, Wartość boolowska)
- Session.createMapMessage()
- Session.createMessage()
- Session.createObjectMessage()
- Session.createObjectMessage(przekształcalny do postaci szeregowej)
- Session.createProducer(miejsce docelowe)
- Session.createQueue(Łańcuch)
- Session.createStreamMessage()

- Session.createTemporaryQueue()
- Session.createTemporaryTopic()
- Session.createTextMessage()
- Session.createTextMessage(Łańcuch)
- Session.createTopic()
- Session.getAcknowledgeMode()
- Session.getMessageListener()
- Session.getTransacted()
- Session.rollback()
- Session.unsubscribe(Łańcuch)

Jeśli zostanie wywołana dowolna z tych metod, zostanie zgłoszony wyjątek JMSException zawierający komunikat:

```
JMSCC0033: Wywołanie metody synchronicznej nie jest dozwolone, gdy sesja jest używana
asynchronicznie: 'nazwa metody'
```

jest zgłaszany.

## Odbieranie komunikatów nieprzetwarzalnych

Aplikacja może odebrać komunikat, którego nie można przetworzyć. Istnieje kilka powodów, dla których nie można przetworzyć komunikatu, na przykład komunikat może mieć niepoprawny format. Takie komunikaty są opisane jako komunikaty nieprzetwarzalne i wymagają specjalnej obsługi, aby zapobiec ich rekurencyjnemu przetwarzaniu.

Szczegółowe informacje na temat obsługi komunikatów nieprzetwarzalnych zawiera sekcja “Obsługa komunikatów nieprzetwarzalnych w produkcie IBM MQ classes for JMS” na stronie 240.

## Dostosowywanie wielkości buforu do odbieranych komunikatów

Po odebraniu komunikatu z produktu IBM MQ przez aplikację inną niżJMS , aby komunikat mógł zostać zapisany, aplikacja musi dostarczyć bufor komunikatów. Aplikacje JMS nie muszą ręcznie tworzyć buforu. Produkt IBM MQ classes for JMS automatycznie tworzy i powiększa bufory komunikatów, aby dopasować je do wielkości odbieranych komunikatów. W przypadku większości aplikacji automatycznie zarządzane bufory zapewniają odpowiednią równowagę między wydajnością i wygodą dla programisty aplikacji. W pewnych okolicznościach korzystne może być ręczne określenie początkowej wielkości buforu komunikatów. Domyślna początkowa wielkość buforu odbiorczego IBM MQ JMS wynosi 4 kB. Jeśli aplikacja zawsze będzie odbierać komunikaty o wielkości 256 kB, zaleca się skonfigurowanie początkowego rozmiaru buforu na 256 kB. Pozwala to uniknąć konieczności podejmowania przez IBM MQ classes for JMS prób i niepowodzeń odbierania komunikatu do buforu o wielkości 4 kB przed zmianą jego wielkości na 256 kB i pomyślnym odebraniem komunikatu. W przypadku aplikacji połączonej z klientem pozwala to uniknąć potencjalnie zmarnowanego obiegu w sieci, podczas gdy IBM MQ classes for JMS określa poprawną wielkość buforu, która ma być używana.

Początkową wielkość buforu można skonfigurować, ustawiając właściwość com.ibm.mq.jmqi.defaultMaxMsgSize Java na wybraną wartość (w bajtach). Należy zauważyć, że ta właściwość ma wpływ na wszystkie aplikacje produktu IBM MQ JMS działające w obrębie serwera Java Virtual Machine, dlatego należy uważać, aby nie wpłynąć negatywnie na innych konsumentów komunikatów, którzy odbierają komunikaty o innej wielkości.

IBM MQ classes for JMS nadal próbuje automatycznie zmniejszyć wielkość buforu, jeśli zostanie odebranych kilka komunikatów mniejszych niż skonfigurowana wielkość. Domyślnie dzieje się tak, jeśli odebrano 10 komunikatów, które są mniejsze niż wielkość buforu. Na przykład, jeśli 10 komunikatów zostanie odebranych w wierszu o wielkości 128 kB, bufor zostanie zmniejszony z 256 kB do 128 kB. Następnie jest on zwiększany ponownie po odebraniu większych komunikatów. Można skonfigurować liczbę komunikatów, które muszą zostać odebrane, zanim bufor zostanie zmniejszony. Może to być przydatne na przykład wtedy, gdy wiadomo, że aplikacja odbiera pięć dużych komunikatów,

po których następuje 10 mniejszych komunikatów, a następnie kolejne pięć dużych komunikatów. W przypadku ustawień domyślnych bufor zostanie zmniejszony po odebraniu 10 mniejszych komunikatów i będzie wymagał ponownego zwiększenia dla większych komunikatów. Właściwość systemową Java com.ibm.mq.jmqi.smallMsgBufferReductionThreshold można ustawić na liczbę komunikatów, które muszą zostać odebrane, zanim wielkość buforu zostanie zmniejszona. W tym przykładzie można ustawić wartość 20, aby zapobiec zmniejszeniu wielkości buforu przez 10 mniejszych komunikatów.

Właściwości mogą być ustawiane niezależnie od siebie. Na przykład można pozostawić domyślną wielkość buforu początkowego (4 kB), ale zwiększyć wartość parametru com.ibm.mq.jmqi.smallMsgBufferReductionThreshold , aby po zwiększeniu wielkości buforu pozostało ona dłużej.

Jeśli dla aplikacji JMS w rekordach statystyki MQI widoczna jest duża liczba kodów powrotu MQRC_TRUNCATED_MSG_FAILED (2080), może to oznaczać, że korzystne byłoby skonfigurowanie wyższej początkowej wielkości buforu dla tych aplikacji lub zmniejszenie częstotliwości zmniejszania wielkości buforów. Należy jednak zauważyć, że w przypadku długo działającej aplikacji prawdopodobnie będzie wyświetlana tylko niewielka liczba kodów powrotu MQRC_TRUNCATED_MSG_FAILED. Jest to spowodowane tym, że bufor jest zwiększany do poprawnej wielkości bezpośrednio po odebraniu pierwszego dużego komunikatu i nie jest zmniejszany, chyba że zostanie odebrana mniejsza liczba komunikatów. Dlatego możliwe jest, że duża liczba komunikatów MQRC_TRUNCATED_MSG_FAILED wskazuje inne słabe procedury aplikacji, takie jak nawiązywanie połączenia z produktem IBM MQ w celu odebrania tylko jednego lub dwóch komunikatów przed rozłączeniem.

### *Pobieranie danych użytkownika subskrypcji*

Jeśli komunikaty, które aplikacja IBM MQ classes for JMS wykorzystuje z kolejki, są umieszczane w trwałej subskrypcji zdefiniowanej administracyjnie, aplikacja musi uzyskać dostęp do informacji o danych użytkownika powiązanych z subskrypcją. Ta informacja jest dodawana do komunikatu jako właściwość.

Gdy komunikat jest pobierany z kolejki zawierającej nagłówek RFH2 z folderem MQPS, wartość powiązana z kluczem Sud (jeśli istnieje) jest dodawana jako właściwość łańcuchowa do obiektu komunikatu JMS zwracanego do aplikacji IBM MQ classes for JMS . Aby włączyć pobieranie tej właściwości z komunikatu, można użyć stałej JMS_IBM_SUBSCRIPTION_USER_DATA w interfejsie JmsConstants z następującą metodą pobierania danych użytkownika subskrypcji:

- V 9.3.0 JM 3.0 V 9.3.0 jakarta.jms.Message.getStringProperty(java.lang.String)
- JMS 2.0 javax.jms.Message.getStringProperty(java.lang.String)

W poniższym przykładzie trwała subskrypcja administracyjna jest definiowana za pomocą komendy MQSC **DEFINE SUB**:

```
DEFINE SUB('MY.SUBCRIPTION') TOPICSTR('PUBLIC') DEST('MY.SUBSCRIPTION.Q')
USERDATA('Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q')
```

Kopie komunikatów, które są publikowane w łańcuchu tematu PUBLIC , są umieszczane w kolejce MY.SUBSCRIPTION.Q. Dane użytkownika powiązane z subskrypcją trwałą są następnie dodawane jako właściwość do komunikatu, który jest przechowywany w folderze MQPS nagłówka RFH2 z kluczem Sud.

Aplikacja IBM MQ classes for JMS może wywoływać:

V 9.3.0 JM 3.0 V 9.3.0
```
jakarta.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

JMS 2.0
```
javax.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

Następnie zwracany jest następujący łańcuch:

```
Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q
```

**Pojęcia pokrewne**

"Nagłówek MQRFH2 i JMS" na stronie 154

**Zadania pokrewne**

Definiowanie subskrypcji administracyjnej

**Odsyłacze pokrewne**

DEFINE SUB

Interfejs JmsConstants

## *Zamykanie aplikacji IBM MQ classes for JMS*

Ważne jest, aby aplikacja IBM MQ classes for JMS jawnie zamkła niektóre obiekty JMS przed zatrzymaniem. Metody finalizujące mogą nie być wywoływane, dlatego nie należy polegać na nich, aby zwalniały zasoby. Nie zezwalaj aplikacji na zakończenie ze skompresowanym aktywnym śledzeniem.

Sam proces czyszczenia pamięci nie może zwolnić wszystkich zasobów IBM MQ classes for JMS i IBM MQ w odpowiednim czasie, zwłaszcza jeśli aplikacja tworzy wiele obiektów JMS o krótkim czasie życia na poziomie sesji lub niższym. Dlatego ważne jest, aby aplikacja zamykała obiekt Connection, Session, MessageConsumerlub MessageProducer , gdy nie jest on już wymagany.

Jeśli aplikacja zakończy działanie bez zamknięcia połączenia, nastąpi niejawne wycofanie zmian dla wszystkich sesji transakcyjnych połączenia. Aby upewnić się, że wszystkie zmiany wprowadzone przez aplikację zostały zatwierdzone, przed zamknięciem aplikacji należy jawnie zamknąć połączenie.

Nie należy używać metody finalizującej w aplikacji do zamykania obiektów JMS . Ponieważ metody finalizujące mogą nie być wywoływane, zasoby mogą nie zostać zwolnione. Zamknięcie połączenia powoduje zamknięcie wszystkich sesji, które zostały na jego podstawie utworzone. Podobnie, klasy MessageConsumers i MessageProducers utworzone na podstawie sesji są zamykane, gdy sesja jest zamknięta. Należy jednak rozważyć zamknięcie sesji, MessageConsumersi MessageProducers w sposób jawny, aby zapewnić terminowe zwolnienie zasobów.

Jeśli włączona jest kompresja śledzenia, zamknięcie metody System.Halt() i nieprawidłowe, niekontrolowane zakończenia działania maszyny JVM prawdopodobnie spowodują uszkodzenie pliku śledzenia. Jeśli to możliwe, po zebraniu potrzebnych informacji śledzenia należy wyłączyć narzędzie śledzenia. Jeśli aplikacja jest śledzona do nieprawidłowego zakończenia, należy użyć nieskompresowanych danych wyjściowych śledzenia.

**Uwaga:** Aby rozłączyć się z menedżerem kolejek, aplikacja JMS wywołuje metodę close () dla obiektu połączenia.

## *Obsługa komunikatów nieprzetwarzalnych w produkcie IBM MQ classes for JMS*

Komunikat nieprzetwarzalny to taki, który nie może być przetwarzany przez aplikację odbierającą. Jeśli komunikat nieprzetwarzalny zostanie dostarczony do aplikacji i wycofany określoną liczbę razy, IBM MQ classes for JMS może przenieść go do kolejki wycofania.

Komunikat nieprzetwarzalny to komunikat, który nie może zostać przetworzony przez aplikację odbierającą. Komunikat może mieć nieoczekiwany typ lub zawierać informacje, które nie mogą być obsługiwane przez logikę aplikacji. Jeśli komunikat nieprzetwarzalny zostanie dostarczony do aplikacji, aplikacja nie będzie mogła go przetworzyć i wycofa go do kolejki, z której pochodzi. Domyślnie IBM MQ classes for JMS będzie wielokrotnie ponownie dostarczać komunikat do aplikacji. Może to spowodować, że aplikacja utknie w pętli, stale próbując przetworzyć komunikat nieprzetwarzalny i wycofać go.

Aby temu zapobiec, IBM MQ classes for JMS może wykryć komunikaty nieprzetwarzalne i przenieść je do alternatywnego miejsca docelowego. W tym celu IBM MQ classes for JMS korzysta z następujących właściwości:

- Wartość pola BackoutCount w obrębie deskryptora MQMD wykrytego komunikatu.
- IBM MQ Atrybuty kolejki **BOTHRESH** (próg wycofania) i **BOQNAME** (kolejka wycofania) dla kolejki wejściowej zawierającej komunikat.

Za każdym razem, gdy komunikat jest wycofywany przez aplikację, menedżer kolejek automatycznie zwiększa wartość pola BackoutCount dla komunikatu.

Jeśli program IBM MQ classes for JMS wykryje komunikat, który ma wartość BackoutCount większą od zera, porównuje wartość atrybutu BackoutCount z wartością atrybutu **BOTHRESH** .

- Jeśli wartość BackoutCount jest mniejsza niż wartość atrybutu **BOTHRESH** , IBM MQ classes for JMS dostarcza ją do aplikacji w celu przetworzenia.
- Jeśli jednak wartość parametru BackoutCount jest większa lub równa **BOTHRESH**, komunikat jest traktowany jako nieprzetwarzalny. W takiej sytuacji IBM MQ classes for JMS przenosi komunikat do kolejki określonej przez atrybut **BOQNAME** . Jeśli komunikatu nie można umieścić w kolejce wycofanych komunikatów, jest on przenoszony do kolejki niedostarczonych komunikatów menedżera kolejek lub usuwany, w zależności od opcji raportu komunikatu.

**Uwaga:**

- Jeśli atrybut **BOTHRESH** ma wartość domyślną 0, obsługa komunikatów nieprzetwarzalnych jest wyłączona. Oznacza to, że wszystkie komunikaty nieprzetwarzalne są umieszczane z powrotem w kolejce wejściowej.
- Należy również zauważyć, że IBM MQ classes for JMS wysyła zapytanie do atrybutów **BOTHRESH** i **BOQNAME** dla kolejki przy pierwszym wykryciu komunikatu, który ma wartość BackoutCount większą od zera. Wartości tych atrybutów są następnie buforowane i używane za każdym razem, gdy IBM MQ classes for JMS napotka komunikat, który ma wartość BackoutCount większą od zera.

## Konfigurowanie systemu do obsługi komunikatów nieprzetwarzalnych

Kolejka, której IBM MQ classes for JMS używa podczas uzyskiwania informacji o atrybutach **BOTHRESH** i **BOQNAME** , zależy od stylu wykonywanego przesyłania komunikatów:

- W przypadku przesyłania komunikatów w trybie punkt z punktem jest to bazowa kolejka lokalna. Jest to ważne, gdy aplikacja JMS konsumuje komunikaty z kolejek aliasowych lub kolejek klastra.
- W przypadku przesyłania komunikatów w trybie publikowania i subskrypcji tworzona jest kolejka zarządzana, w której przechowywane są komunikaty dla aplikacji. IBM MQ classes for JMS wysyła zapytanie do kolejki zarządzanej w celu określenia wartości atrybutów **BOTHRESH** i **BOQNAME** .

  Kolejka zarządzana jest tworzona na podstawie kolejki modelowej powiązanej z obiektem tematu, który został zasubskrybowany przez aplikację, i dziedziczy wartości atrybutów **BOTHRESH** i **BOQNAME** z kolejki modelowej. Używana kolejka modelowa zależy od tego, czy aplikacja odbierająca odebrała trwałą, czy nietrwałą subskrypcję:

  – Kolejka modelowa używana na potrzeby trwałych subskrypcji jest określona przez atrybut **MDURMDL** tematu. Wartością domyślną tego atrybutu jest SYSTEM.DURABLE.MODEL.QUEUE.

  – W przypadku subskrypcji nietrwałych używana kolejka modelowa jest określona przez atrybut **MNDURMDL** . Wartością domyślną atrybutu **MNDURMDL** jest SYSTEM.NDURABLE.MODEL.QUEUE.

Podczas odpytywania atrybutów **BOTHRESH** i **BOQNAME** IBM MQ classes for JMS:

- Otwórz kolejkę lokalną lub kolejkę docelową dla kolejki aliasowej.
- Sprawdź atrybuty **BOTHRESH** i **BOQNAME** .
- Zamknij kolejkę lokalną lub kolejkę docelową dla kolejki aliasowej.

Opcje otwierania, które są używane podczas otwierania kolejki lokalnej lub kolejki docelowej dla kolejki aliasowej, zależą od używanej wersji programu IBM MQ classes for JMS :

- W przypadku produktu IBM MQ classes for JMS w wersji IBM MQ 9.1.0 Fix Pack 1 lub wcześniejszej albo produktu IBM MQ 9.1.1, jeśli kolejka lokalna lub kolejka docelowa dla kolejki aliasowej jest kolejką klastrową, IBM MQ classes for JMS otwórz kolejkę za pomocą opcji MQOO_INPUT_AS_Q_DEF, MQOO_INQUIRE i MQOO_FAIL_IF_QUIESCING . Oznacza to, że użytkownik uruchamiający aplikację odbierającą musi mieć dostęp do zapytań i uzyskiwania dostępu do lokalnej instancji kolejki klastra.

  IBM MQ classes for JMS otwiera wszystkie inne typy kolejek lokalnych z opcjami otwarcia MQOO_INQUIRE i MQOO_FAIL_IF_QUIESCING. Aby program IBM MQ classes for JMS mógł wysyłać zapytania o wartości atrybutów, użytkownik uruchamiający aplikację odbierającą musi mieć dostęp do zapytań w kolejce lokalnej.

- W przypadku korzystania z serwera IBM MQ classes for JMS w systemie IBM MQ 9.1.0 Fix Pack 2 lub nowszym albo w systemie IBM MQ 9.1.2 lub nowszym użytkownik uruchamiający aplikację odbierającą musi mieć dostęp do zapytań w kolejce lokalnej, niezależnie od typu kolejki.

Aby przenieść komunikaty nieprzetwarzalne do kolejki wycofanych komunikatów lub kolejki niedostarczonych komunikatów menedżera kolejek, należy nadać użytkownikowi uruchamiający aplikację uprawnienia `put` i `passall` .

## Przetwarzanie komunikatów nieprzetwarzalnych dla aplikacji synchronicznych

Jeśli aplikacja odbiera komunikaty synchronicznie, wywołując jedną z następujących metod, program IBM MQ classes for JMS ponownie wyświetla komunikat nieprzetwarzalny w jednostce pracy, która była aktywna podczas próby pobrania komunikatu przez aplikację:

- JMSConsumer.receive()
- JMSConsumer.receive(długi limit czasu)
- JMSConsumer.receiveBody(Klasa < T> c)
- JMSConsumer.receiveBody(Klasa < T> c, długi limit czasu)
- JMSConsumer.receiveBodyNoWait Klasa < T> c)
- JMSConsumer.receiveNoWait()
- MessageConsumer.receive ()
- MessageConsumer.receive (długi limit czasu)
- MessageConsumer.receiveNoWait ()
- QueueReceiver.receive ()
- QueueReceiver.receive (długi limit czasu)
- QueueReceiver.receiveNoWait ()
- Klasa TopicSubscriber.receive ()
- TopicSubscriber.receive (długi limit czasu)
- TopicSubscriber.receiveNoCzekaj ()

Oznacza to, że jeśli aplikacja używa kontekstu lub sesji JMS z transakcją, przeniesienie komunikatu do kolejki wycofania nie zostanie zatwierdzone do momentu zatwierdzenia transakcji.

Jeśli atrybut **BOTHRESH** jest ustawiony na wartość inną niż zero, należy również ustawić atrybut **BOQNAME** . Jeśli parametr **BOTHRESH** jest ustawiony na wartość większą niż zero, a parametr **BOQNAME** nie został ustawiony, zachowanie jest określane przez opcje raportu komunikatu:

- Jeśli komunikat ma ustawioną opcję raportu `MQRO_DISCARD_MSG` , komunikat jest odrzucany.
- Jeśli dla komunikatu określono opcję raportu `MQRO_DEAD_LETTER_Q` , program IBM MQ classes for JMS próbuje przenieść komunikat do kolejki niedostarczonych komunikatów menedżera kolejek.
- Jeśli komunikat nie ma ustawionej opcji `MQRO_DISCARD_MSG` lub `MQRO_DEAD_LETTER_Q` , IBM MQ classes for JMS podejmie próbę umieszczenia komunikatu w kolejce niedostarczonych komunikatów dla menedżera kolejek.

W przypadku, gdy próba umieszczenia komunikatu w kolejce niedostarczonych komunikatów nie powiedzie się z jakiegoś powodu, to, co stanie się z komunikatem, jest określane na podstawie tego, czy aplikacja odbierająca używa kontekstu lub sesji JMS z transakcją lub bez transakcji:

- Jeśli aplikacja odbierająca używa kontekstu lub sesji JMS z transakcją i transakcja jest zatwierdzona, komunikat jest odrzucany.
- Jeśli aplikacja odbierająca używa transakcyjnego kontekstu lub sesji JMS i wycofa transakcję, komunikat zostanie zwrócony do kolejki wejściowej.
- Jeśli aplikacja odbierająca utworzyła nietransakcyjne JMS kontekst lub sesję, komunikat jest odrzucany.

## Przetwarzanie komunikatów nieprzetwarzalnych dla aplikacji asynchronicznych

Jeśli aplikacja odbiera komunikaty asynchronicznie za pośrednictwem obiektu MessageListener, komunikaty nieprzetwarzalne w kolejce produktu IBM MQ classes for JMS nie mają wpływu na dostarczanie komunikatów. Proces requeue odbywa się poza jakąkolwiek jednostką pracy powiązaną z rzeczywistym dostarczaniem komunikatów do aplikacji.

Jeśli parametr **BOTHRESH** jest ustawiony na wartość większą niż zero, a parametr **BOQNAME** nie został ustawiony, zachowanie jest określane przez opcje raportu komunikatu:

- Jeśli komunikat ma ustawioną opcję raportu `MQRO_DISCARD_MSG` , komunikat jest odrzucany.
- Jeśli dla komunikatu określono opcję raportu `MQRO_DEAD_LETTER_Q` , program IBM MQ classes for JMS próbuje przenieść komunikat do kolejki niedostarczonych komunikatów menedżera kolejek.
- Jeśli komunikat nie ma ustawionej opcji `MQRO_DISCARD_MSG` lub `MQRO_DEAD_LETTER_Q` , IBM MQ classes for JMS podejmie próbę umieszczenia komunikatu w kolejce niedostarczonych komunikatów dla menedżera kolejek.

Jeśli próba umieszczenia komunikatu w kolejce niedostarczonych komunikatów nie powiedzie się z jakiegoś powodu, program IBM MQ classes for JMS zwróci komunikat do kolejki wejściowej.

Informacje na temat sposobu, w jaki specyfikacje aktywowania i klasa ConnectionConsumers obsługują komunikaty nieprzetwarzalne, zawiera sekcja Usuwanie komunikatów z kolejki w narzędziu ASF.

## Co się dzieje z komunikatem, gdy jest on przenoszony do kolejki wycofania

Gdy komunikat nieprzetwarzalny jest umieszczany w kolejce wycofanych komunikatów, IBM MQ classes for JMS dodaj do niego nagłówek RFH2 (jeśli jeszcze go nie ma) i zaktualizuj niektóre pola w deskryptorze komunikatu (MQMD).

Jeśli komunikat nieprzetwarzalny zawiera nagłówek RFH2 (na przykład dlatego, że był to komunikat JMS ), IBM MQ classes for JMS podczas przenoszenia komunikatu do kolejki wycofanych komunikatów zmienia następujące pola w strukturze MQMD:

- Wartość w polu BackoutCount zostanie zresetowana do zera.
- Pole Utrata ważności komunikatu jest aktualizowane w celu odzwierciedlenia pozostałej utraty ważności w momencie odebrania komunikatu nieprzetwarzalnego przez aplikację JMS .

Jeśli komunikat nieprzetwarzalny nie zawiera nagłówka RFH2 , IBM MQ classes for JMS dodaje go i aktualizuje następujące pola w strukturze MQMD w ramach przetwarzania wycofania:

- Wartość w polu BackoutCount zostanie zresetowana do zera.
- Pole Utrata ważności komunikatu jest aktualizowane w celu odzwierciedlenia pozostałej utraty ważności w momencie odebrania komunikatu nieprzetwarzalnego przez aplikację JMS .
- Pole Format komunikatu zostanie zmienione na MQHRF2.
- Wartość w polu CCSID jest zmieniana na 1208.
- Wartość w polu Kodowanie zostanie zmieniona na 273.

Oprócz tego pola CCSID i Encoding z komunikatu nieprzetwarzalnego są kopiowane do pól CCSID i Encoding nagłówka RFH2 , aby upewnić się, że łańcuch nagłówków komunikatu w kolejce wycofanych komunikatów jest poprawny.

**Pojęcia pokrewne**

“Obsługa komunikatów nieprzetwarzalnych w ASF” na stronie 348
W ramach narzędzi serwera aplikacji obsługa komunikatów nieprzetwarzalnych jest obsługiwana nieco inaczej niż w innych miejscach w produkcie IBM MQ classes for JMS.

## *Wyjątki w IBM MQ classes for JMS*

Aplikacja IBM MQ classes for JMS musi obsługiwać wyjątki zgłaszane przez wywołania funkcji API języka JMS lub dostarczane do procedury obsługi wyjątków.

IBM MQ classes for JMS zgłasza problemy w czasie wykonywania, zgłaszając wyjątki. Typ zgłaszanych wyjątków oraz sposób obsługi tych wyjątków zależą od wersji specyfikacji JMS używanej przez aplikację:

- Metody w interfejsach zdefiniowanych w usłudze JMS 1.1 i wcześniejszych zgłaszają sprawdzone wyjątki. Klasą bazową tych wyjątków jest `JMSException`. Więcej informacji na temat obsługi sprawdzonych wyjątków zawiera sekcja “Obsługa sprawdzonych wyjątków” na stronie 244.
- Metody w interfejsach dodanych w JMS 2.0 zgłaszają niesprawdzone wyjątki. Klasą bazową dla tych wyjątków jest `JMSRuntimeException`. Więcej informacji na temat obsługi niekontrolowanych wyjątków zawiera sekcja “Obsługa niesprawdzonych wyjątków” na stronie 247.

Można również zarejestrować `ExceptionListener` za pomocą usługi JMS `Connection` lub `JMSContext`. Klasy MQ dla usługi JMS następnie powiadamiają `ExceptionListener` , jeśli wykryto problem z połączeniem z menedżerem kolejek lub jeśli wystąpił problem podczas próby asynchronicznego dostarczenia komunikatu. Aby uzyskać więcej informacji, zapoznaj się z sekcją: “ExceptionListeners” na stronie 250.

**Pojęcia pokrewne**

Klasy produktu IBM MQ dla usługi JMS

**Odsyłacze pokrewne**

WYJĄTEK ASYNCEXCEPTION

*Obsługa sprawdzonych wyjątków*

Metody w interfejsach zdefiniowanych w usłudze JMS 1.1 lub wcześniejszej zgłaszają sprawdzone wyjątki. Klasą bazową dla tych wyjątków jest `JMSException`. Dlatego przechwytywanie `JMSExceptions` udostępnia ogólny sposób obsługi tych typów wyjątków.

Każdy `JMSException` zawiera następujące informacje:

- Komunikat wyjątku specyficzny dla dostawcy, który aplikacja może uzyskać, wywołując metodę `Throwable.getMessage()` .
- Kod błędu specyficzny dla dostawcy, który aplikacja może uzyskać, wywołując metodę `JMSException.getErrorCode()` .
- Powiązany wyjątek. Wyjątek zgłaszany przez wywołanie funkcji API JMS 1.1 jest często wynikiem problemu niższego poziomu, który jest zgłaszany przez inny wyjątek powiązany z tym wyjątkiem. Aplikacja może uzyskać powiązany wyjątek, wywołując metodę `JMSException.getLinkedException()` lub metodę `Throwable.getCause()` .

Jeśli używany jest interfejs API JMS 1.1 , większość wyjątków zgłaszanych przez IBM MQ classes for JMS to instancje podklas klasy `JMSException`. Te podklasy implementują interfejs `com.ibm.msg.client.jms.JmsExceptionDetail` , który udostępnia następujące informacje dodatkowe:

- Wyjaśnienie komunikatu o wyjątku. Aplikacja może uzyskać ten komunikat, wywołując metodę `JmsExceptionDetail.getExplanation()` .
- Zalecana odpowiedź użytkownika na wyjątek. Aplikacja może uzyskać ten komunikat, wywołując metodę `JmsExceptionDetail.getUserAction()` .
- Klucze dla komunikatu wstawianego do komunikatu o wyjątku. Aplikacja może uzyskać iterator dla wszystkich kluczy, wywołując metodę `JmsExceptionDetail.getKeys()` .
- Komunikat jest wstawiany do komunikatu o wyjątku. Na przykład wstawienie komunikatu może być nazwą kolejki, która spowodowała wyjątek, i może być przydatne dla aplikacji w celu uzyskania dostępu do tej nazwy. Aplikacja może uzyskać komunikat wstawiany odpowiadający określonemu kluczowi, wywołując metodę `JmsExceptionDetail.getValue()` .

Wszystkie metody w interfejsie `JmsExceptionDetail` zwracają wartość NULL, jeśli nie są dostępne żadne szczegóły.

Jeśli na przykład aplikacja próbuje utworzyć producenta komunikatów dla kolejki IBM MQ , która nie istnieje, zgłaszany jest wyjątek z następującymi informacjami:

```
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.
```

Zgłaszany wyjątek `com.ibm.msg.client.jms.DetailedInvalidDestinationException`jest podklasą następującej klasy i implementuje interfejs `com.ibm.msg.client.jms.JmsExceptionDetail` .

- V 9.3.0 JM 3.0 V 9.3.0 `jakarta.jms.InvalidDestinationException`
- JMS 2.0 `javax.jms.InvalidDestinationException`

## Powiązane wyjątki

Powiązany wyjątek udostępnia dodatkowe informacje na temat problemu ze środowiskiem wykonawczym. Dlatego dla każdego zgłaszanego wyjątku `JMSException` aplikacja powinna sprawdzić dołączony wyjątek.

Sam powiązany wyjątek może mieć inny powiązany wyjątek, dlatego połączone wyjątki tworzą łańcuch, który prowadzi z powrotem do pierwotnego problemu bazowego. Powiązany wyjątek jest implementowany przy użyciu mechanizmu połączonych wyjątków klasy `java.lang.Throwable` , a aplikacja może uzyskać powiązany wyjątek, wywołując metodę `Throwable.getCause()` . W przypadku metody `JMSException`metoda `getLinkedException()` deleguje do metody `Throwable.getCause()` .

Jeśli na przykład podczas nawiązywania połączenia z menedżerem kolejek aplikacja określa niepoprawny numer portu, wyjątki tworzą następujący łańcuch:

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

Zwykle każdy wyjątek w łańcuchu jest zgłaszany z innej warstwy w kodzie. Na przykład wyjątki w poprzednim łańcuchu są zgłaszane przez następujące warstwy:

- Pierwszy wyjątek, instancja podklasy `JMSException`, jest zgłaszany przez wspólną warstwę w produkcie IBM MQ classes for JMS.
- Następny wyjątek, instancja `com.ibm.mq.MQException`, jest zgłaszany przez dostawcę przesyłania komunikatów produktu IBM MQ .
- Kolejne dwa wyjątki, które są instancjami interfejsu `com.ibm.mq.jmqi.JmqiException`, są zgłaszane przez interfejs JMQI ( Java Message Queueing Interface). JMQI jest komponentem używanym przez program IBM MQ classes for JMS do komunikowania się z menedżerem kolejek.
- Ostatni wyjątek, instancja klasy `java.net.ConnectionException`, jest zgłaszany przez bibliotekę klas Java .

Więcej informacji na temat architektury warstwowej produktu IBM MQ classes for JMSzawiera sekcja IBM MQ classes for JMS architecture.

Aplikację można zakodować tak, aby przechodziła przez ten łańcuch w celu wyodrębnienia wszystkich odpowiednich informacji, jak pokazano w poniższym przykładzie:

V 9.3.0 JM 3.0 V 9.3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
```

```
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

Należy zauważyć, że aplikacja powinna zawsze sprawdzać typ każdego wyjątku w łańcuchu, ponieważ typ wyjątku może być różny, a wyjątki różnych typów mogą zawierać różne informacje.

## Uzyskiwanie informacji specyficznych dla systemu IBM MQ dotyczących problemu

Instancje `com.ibm.mq.MQException` i `com.ibm.mq.jmqi.JmqiException` hermetyzują IBM MQ konkretne informacje o problemie.

`MQException` hermetyzuje następujące informacje:

- Kod zakończenia, który aplikacja może uzyskać, wywołując metodę `getCompCode()` .
- Kod przyczyny, który aplikacja może uzyskać, wywołując metodę `getReason()` .

Przykłady użycia tych metod można znaleźć w przykładowym kodzie w sekcji Dowiązane wyjątki.

`JmqiException` hermetyzuje również kod zakończenia i kod przyczyny. Oprócz tego `JmqiException` zawiera informacje w komunikacie AMQ *nnnn* lub CSQ *nnnn* , jeśli są one powiązane z wyjątkiem. Aplikacja może uzyskać różne komponenty tego komunikatu, wywołując następujące metody:

- Metoda `getWmqMsgExplanation()` zwraca wyjaśnienie komunikatu AMQ *nnnn* lub CSQ *nnnn* .
- Metoda `getWmqMsgSeverity()` zwraca istotność komunikatu AMQ *nnnn* lub CSQ *nnnn* .
- Metoda `getWmqMsgSummary()` zwraca podsumowanie komunikatu AMQ *nnnn* lub CSQ *nnnn* .
- Metoda `getWmqMsgUserResponse()` zwraca odpowiedź użytkownika powiązaną z komunikatem AMQ *nnnn* lub CSQ *nnnn* .

*Obsługa niesprawdzonych wyjątków*
Metody w interfejsach zdefiniowanych w usłudze JMS 2.0 zgłaszają niesprawdzone wyjątki. Klasą bazową dla tych wyjątków jest `JMSRuntimeException`. Dlatego przechwytywanie `JMSRuntimeExceptions` udostępnia ogólny sposób obsługi tych typów wyjątków.

Każdy `JMSRuntimeException` zawiera następujące informacje:

- Komunikat wyjątku specyficzny dla dostawcy, który aplikacja może uzyskać, wywołując metodę `JMSRuntimeException.getMessage()` .
- Kod błędu specyficzny dla dostawcy, który aplikacja może uzyskać, wywołując metodę `JMSRuntimeException.getErrorCode()` .
- Powiązany wyjątek. Wyjątek zgłaszany przez wywołanie funkcji API JMS 2.0 jest często wynikiem problemu niższego poziomu, który jest zgłaszany przez inny wyjątek powiązany z tym wyjątkiem. Aplikacja może uzyskać powiązany wyjątek, wywołując metodę `JMSRuntimeException.getCause()` .

Podczas wywoływania metod w interfejsach udostępnianych przez interfejs API języka JMS 2.0 większość wyjątków zgłaszanych przez interfejs IBM MQ classes for JMS to instancje podklas klasy `JMSRuntimeException`. Te podklasy implementują interfejs `com.ibm.msg.client.jms.JmsExceptionDetail` , który udostępnia następujące informacje dodatkowe:

- Wyjaśnienie komunikatu o wyjątku. Aplikacja może uzyskać ten komunikat, wywołując metodę `JmsExceptionDetail.getExplanation()` .
- Zalecana odpowiedź użytkownika na wyjątek. Aplikacja może uzyskać ten komunikat, wywołując metodę `JmsExceptionDetail.getUserAction()` .

- Klucze dla komunikatu wstawianego do komunikatu o wyjątku. Aplikacja może uzyskać iterator dla wszystkich kluczy, wywołując metodę `JmsExceptionDetail.getKeys()`.
- Komunikat jest wstawiany do komunikatu o wyjątku. Na przykład wstawienie komunikatu może być nazwą kolejki, która spowodowała wyjątek, i może być przydatne dla aplikacji w celu uzyskania dostępu do tej nazwy. Aplikacja może uzyskać komunikat wstawiany odpowiadający określonemu kluczowi, wywołując metodę `JmsExceptionDetail.getValue()`.

Wszystkie metody w interfejsie `JmsExceptionDetail` zwracają wartość NULL, jeśli nie są dostępne żadne szczegóły.

Jeśli na przykład aplikacja próbuje utworzyć kolejkę `JMSProducer` dla kolejki IBM MQ , która nie istnieje, zgłaszany jest wyjątek z następującymi informacjami:

```
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.
```

Zgłaszany wyjątek `com.ibm.msg.client.jms.DetailedInvalidDestinationException`jest podklasą następującej klasy i implementuje interfejs `com.ibm.msg.client.jms.JmsExceptionDetail`.

- V 9.3.0 JM 3.0 V 9.3.0 `jakarta.jms.InvalidDestinationException`
- JMS 2.0 `javax.jms.InvalidDestinationException`

## Połączone wyjątki

Zazwyczaj wyjątki są powodowane przez inne wyjątki. Dlatego dla każdego zgłaszanego wyjątku `JMSRuntimeException` aplikacja powinna sprawdzić powiązany wyjątek.

Przyczyną `JMSRuntimeException` może być inny wyjątek. Te wyjątki tworzą łańcuch, który prowadzi do pierwotnego problemu bazowego. Przyczyna wyjątku jest implementowana przy użyciu mechanizmu połączonych wyjątków klasy `java.lang.Throwable` , a aplikacja może uzyskać powiązany wyjątek, wywołując metodę `Throwable.getCause()`.

Jeśli na przykład podczas nawiązywania połączenia z menedżerem kolejek aplikacja określa niepoprawny numer portu, wyjątki tworzą następujący łańcuch:

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

Zwykle każdy wyjątek w łańcuchu jest zgłaszany z innej warstwy w kodzie. Na przykład wyjątki w poprzednim łańcuchu są zgłaszane przez następujące warstwy:

- Pierwszy wyjątek, instancja podklasy `JMSRuntimeException`, jest zgłaszany przez wspólną warstwę w produkcie IBM MQ classes for JMS.
- Następny wyjątek, instancja `com.ibm.mq.MQException`, jest zgłaszany przez dostawcę przesyłania komunikatów produktu IBM MQ .

- Kolejne dwa wyjątki, które są instancjami interfejsu `com.ibm.mq.jmqi.JmqiException`, są zgłaszane przez interfejs JMQI ( Java Message Queueing Interface). JMQI jest komponentem używanym przez program IBM MQ classes for JMS do komunikowania się z menedżerem kolejek.
- Ostatni wyjątek, instancja klasy `java.net.ConnectionException`, jest zgłaszany przez bibliotekę klas Java .

Więcej informacji na temat architektury warstwowej produktu IBM MQ classes for JMSzawiera sekcja IBM MQ classes for JMS architecture.

Aplikację można zakodować tak, aby przechodziła przez ten łańcuch w celu wyodrębnienia wszystkich odpowiednich informacji, jak pokazano w poniższym przykładzie:

V 9.3.0 JM 3.0 V 9.3.0

```
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();
  }
}
```

Należy zauważyć, że aplikacja powinna zawsze sprawdzać typ każdego wyjątku w łańcuchu, ponieważ typ wyjątku może być różny, a wyjątki różnych typów mogą zawierać różne informacje.

## Uzyskiwanie informacji specyficznych dla systemu IBM MQ dotyczących problemu

Instancje `com.ibm.mq.MQException` i `com.ibm.mq.jmqi.JmqiException` hermetyzują IBM MQ konkretne informacje o problemie.

`MQException` hermetyzuje następujące informacje:

- Kod zakończenia, który aplikacja może uzyskać, wywołując metodę `getCompCode()` .
- Kod przyczyny, który aplikacja może uzyskać, wywołując metodę `getReason()` .

Przykłady użycia tych metod można znaleźć w przykładowym kodzie w sekcji Wyjątki w łańcuchu.

`JmqiException` hermetyzuje również kod zakończenia i kod przyczyny. Oprócz tego `JmqiException` zawiera informacje w komunikacie AMQ *nnnn* lub CSQ *nnnn* , jeśli są one powiązane z wyjątkiem. Aplikacja może uzyskać różne komponenty tego komunikatu, wywołując następujące metody:

- Metoda `getWmqMsgExplanation()` zwraca wyjaśnienie komunikatu AMQ *nnnn* lub CSQ *nnnn* .
- Metoda `getWmqMsgSeverity()` zwraca istotność komunikatu AMQ *nnnn* lub CSQ *nnnn* .
- Metoda `getWmqMsgSummary()` zwraca podsumowanie komunikatu AMQ *nnnn* lub CSQ *nnnn* .
- Metoda `getWmqMsgUserResponse()` zwraca odpowiedź użytkownika powiązaną z komunikatem AMQ *nnnn* lub CSQ *nnnn* .

*ExceptionListeners*
Obiekty JMS `Connection` i `JMSContext` mają powiązane połączenie z menedżerem kolejek. Aplikacja może zarejestrować `ExceptionListener` w usłudze JMS `Connection` lub `JMSContext`. Jeśli wystąpi problem, który powoduje, że połączenie powiązane z `Connection` lub `JMSContext` jest bezużyteczne, IBM MQ classes for JMS dostarcza wyjątek do metody `ExceptionListener` , wywołując metodę `onException()` . Aplikacja będzie miała możliwość ponownego nawiązania połączenia.

Produkt IBM MQ classes for JMS może również dostarczyć wyjątek do obiektu nasłuchiwania wyjątków, jeśli wystąpi problem podczas próby asynchronicznego dostarczenia komunikatu.

## Obiekty nasłuchiwania wyjątków

W produkcie IBM MQ 8.0.0 Fix Pack 2, aby zachować zachowanie dla bieżących aplikacji JMS, które konfigurują JMS `MessageListener` i JMS `ExceptionListener`, oraz aby zapewnić spójność IBM MQ classes for JMS ze specyfikacją JMS , wartość domyślna właściwości `ConnectionFactory` ASYNCEXCEPTION jest zmieniana na ASYNC_EXCEPTIONS_CONNECTIONBROKEN. W rezultacie tylko

wyjątki, które odpowiadają kodom błędów zerwanego połączenia, są dostarczane do serwera `ExceptionListener` aplikacji.

Raport APAR IT14820dołączony do produktu IBM MQ 9.0.0 Fix Pack 1aktualizuje plik IBM MQ classes for JMS , tak aby:

- Serwer `ExceptionListener` , który jest rejestrowany przez aplikację, jest wywoływany dla wszystkich wyjątków zerwania połączenia, niezależnie od tego, czy aplikacja używa synchronicznych czy asynchronicznych konsumentów komunikatów.
- Wyjątki niezwiązane z przerwaniem połączenia (na przykład MQRC_GET_INHIBITED), które występują podczas dostarczania komunikatów, są dostarczane do serwera `ExceptionListener` aplikacji, gdy aplikacja używa asynchronicznych konsumentów komunikatów, a właściwość JMS ConnectionFactory używana przez aplikację ma właściwość ASYNC_EXCEPTION ustawioną na wartość ASYNC_EXCEPTIONS_ALL.

**Uwaga:** `ExceptionListener` jest wywoływana tylko raz dla wyjątku zerwanego połączenia, nawet jeśli dwa połączenia TCP/IP (jedno używane przez połączenie JMS i jedno używane przez sesję JMS ) są zerwane.

W przypadku każdego innego typu problemu bieżące wywołanie funkcji API JMS powoduje zgłoszenie wyjątku. Typ zgłaszanego wyjątku zależy od wersji interfejsu API języka JMS używanego przez aplikację:

- Jeśli aplikacja używa interfejsów, które są udostępniane przez specyfikację JMS 1.1 , wyjątkiem jest `JMSException`. Więcej informacji na temat obsługi tych wyjątków zawiera sekcja “Obsługa sprawdzonych wyjątków” na stronie 244.
- Jeśli aplikacja używa interfejsów JMS 2.0 , wyjątkiem jest `JMSRuntimeException`. Więcej informacji na temat obsługi tych wyjątków zawiera sekcja “Obsługa niesprawdzonych wyjątków” na stronie 247.

Jeśli aplikacja nie zarejestruje obiektu nasłuchiwania wyjątków w systemie `Connection` lub `JMSContext`, wszystkie wyjątki, które zostałyby dostarczone do obiektu nasłuchiwania wyjątków, zostaną zapisane w dzienniku IBM MQ classes for JMS .

## *Uzyskiwanie dostępu do funkcji programu IBM MQ z poziomu aplikacji IBM MQ classes for JMS*

IBM MQ classes for JMS udostępnia narzędzia do korzystania z wielu funkcji produktu IBM MQ.

**Ostrzeżenie:** Te funkcje nie są zgodne ze specyfikacją JMS lub, w niektórych przypadkach, naruszają specyfikację JMS . W przypadku ich użycia aplikacja nie będzie kompatybilna z innymi dostawcami JMS . Te funkcje, które nie są zgodne ze specyfikacją JMS , są oznaczone etykietą z powiadomieniem typu Uwaga (Attention).

*Odczytywanie i zapisywanie deskryptora komunikatu z aplikacji IBM MQ classes for JMS*

Możliwość dostępu do deskryptora komunikatu (MQMD) można kontrolować, ustawiając właściwości miejsca docelowego i komunikatu.

Niektóre aplikacje IBM MQ wymagają ustawienia konkretnych wartości w strukturze MQMD wysyłanych do nich komunikatów. Produkt IBM MQ classes for JMS udostępnia atrybuty komunikatów, które umożliwiają aplikacjom JMS ustawianie pól MQMD, dzięki czemu aplikacje JMS mogą "kierować" aplikacjami IBM MQ .

Aby ustawienie właściwości MQMD przyniosło jakikolwiek efekt, należy ustawić właściwość obiektu docelowego WMQ_MQMD_WRITE_ENABLED na wartość true. Następnie można użyć metod ustawiania właściwości komunikatu (na przykład właściwości setString), aby przypisać wartości do pól MQMD. Wszystkie pola MQMD są prezentowane z wyjątkiem StrucId i Version; BackoutCount może być odczytywane, ale nie zapisywane.

W tym przykładzie komunikat jest umieszczany w kolejce lub temacie za pomocą programu MQMD.UserIdentifier ustawiony na "JoeBloggs".

```
// Create a ConnectionFactory, connection, session, producer, message
// ...
```

```
// Create a destination
// ...

// Enable MQMD write
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_WRITE_ENABLED, true);

// Optionally, set a message context if applicable for this MD field
dest.setIntProperty(WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT,
  WMQConstants.WMQ_MDCTX_SET_IDENTITY_CONTEXT);

// On the message, set property to provide custom UserId
msg.setStringProperty("JMS_IBM_MQMD_UserIdentifier", "JoeBloggs");

// Send the message
// ...
```

Konieczne jest ustawienie właściwości WMQ_MQMD_MESSAGE_CONTEXT przed ustawieniem właściwości JMS_IBM_MQMD_UserIdentifier. Więcej informacji na temat używania właściwości WMQ_MQMD_MESSAGE_CONTEXT zawiera sekcja “Właściwości obiektu komunikatu JMS” na stronie 254.

Podobnie można wyodrębnić zawartość pól MQMD, ustawiając właściwość WMQ_MQMD_READ_ENABLED na wartość true przed odebraniem komunikatu, a następnie przy użyciu metod pobierania komunikatu, takich jak właściwość getString. Wszystkie odebrane właściwości są tylko do odczytu.

W tym przykładzie wynikiem jest pole *value* zawierające wartość MQMD.ApplIdentityData komunikatu uzyskanego z kolejki lub tematu.

```
// 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");
```

*Właściwości obiektu docelowego JMS*
Dwie właściwości obiektu docelowego sterują dostępem do deskryptora MQMD z programu JMS, a trzecia steruje kontekstem komunikatu.

*Tabela 36. Nazwy i opisy właściwości*

| **Właściwość** | **Postać krótka** | **Opis** |
|---|---|---|
| WMQ_MQMD_WRITE_ENABLED | MDW | Określa, czy aplikacja JMS może ustawiać wartości pól MQMD |
| WMQ_MQMD_READ_ENABLED | MDR | Określa, czy aplikacja JMS może wyodrębniać wartości pól MQMD |
| WMQ_MQMD_MESSAGE_ KONTEKST | MDCTX | Poziom kontekstu komunikatu, który ma zostać ustawiony przez aplikację JMS . Aby ta właściwość odniosła skutek, aplikacja musi być uruchomiona z odpowiednimi uprawnieniami kontekstu. |

*Tabela 37. Nazwy, wartości i metody ustawiania właściwości*

| Właściwość | Poprawne wartości w narzędziu administracyjnym (wartości domyślne pogrubione) | Poprawne wartości w programach | Ustaw metodę |
|---|---|---|---|
| WMQ_MQMD_WRITE_PLABLED | • **Nie**<br>Wszystkie właściwości JMS_IBM_MQMD* są ignorowane, a ich wartości nie są kopiowane do bazowej struktury MQMD.<br>• YES<br>Właściwości JMS_IBM_MQMD* są przetwarzane. Ich wartości są kopiowane do bazowej struktury MQMD. | • **Fałsz**<br>• Prawda | setMQMDWritewłączone |
| WMQ_MQMD_READ_PLABLED | • **Nie**<br>Podczas wysyłania komunikatów właściwości JMS_IBM_MQMD* wysłanego komunikatu nie są aktualizowane w celu odzwierciedlenia zaktualizowanych wartości pól w strukturze MQMD.<br>Podczas odbierania komunikatów żadna właściwość JMS_IBM_MQMD* nie jest dostępna w odebranym komunikacie, nawet jeśli nadawca ustawił niektóre lub wszystkie z nich.<br>• YES<br>Podczas wysyłania komunikatów wszystkie właściwości JMS_IBM_MQMD* wysłanego komunikatu są aktualizowane w celu odzwierciedlenia zaktualizowanych wartości pól w strukturze MQMD, w tym również te, które nie zostały jawnie ustawione przez nadawcę.<br>Podczas odbierania komunikatów wszystkie właściwości JMS_IBM_MQMD* są dostępne w odebranym komunikacie (w tym także te, które nie zostały jawnie ustawione przez nadawcę). | • **Fałsz**<br>• Prawda | setMQMDReadwłączone |

| *Tabela 37. Nazwy, wartości i metody ustawiania właściwości (kontynuacja)* | | | |
|---|---|---|---|
| **Właściwość** | **Poprawne wartości w narzędziu administracyjnym (wartości domyślne pogrubione)** | **Poprawne wartości w programach** | **Ustaw metodę** |
| WMQ_MQMD _MESSAGE_CONTEXT | • **Domyślnie**<br>Wywołanie API MQOPEN i struktura MQPMO nie określają jawnych opcji kontekstu komunikatu<br>• SET_IDENTITY_CONTEXT,<br>Wywołanie funkcji API MQOPEN określa opcję kontekstu komunikatu MQOO_SET_IDENTITY_CONTEXT, a struktura MQPMO określa opcję MQPMO_SET_IDENTITY_CONTEXT.<br>• SET_ALL_CONTEXT<br>Wywołanie funkcji API MQOPEN określa opcję kontekstu komunikatu MQOO_SET_ALL_CONTEXT, a struktura MQPMO określa opcję MQPMO_SET_ALL_CONTEXT. | • **WMQ_MDCTX_DEFAULT**<br>• WMQ_MDCTX_SET_IDENTITY_CONTEXT<br>• WMQ_MDCTX_SET_ALL_CONTEXT | Kontekst setMQMDMessage |

*Właściwości obiektu komunikatu JMS*
Właściwości obiektu komunikatu z przedrostkiem JMS_IBM_MQMD umożliwiają ustawienie lub odczytanie odpowiedniego pola MQMD.

## Wysyłanie komunikatów

Reprezentowane są wszystkie pola MQMD z wyjątkiem StrucId i Version. Te właściwości odnoszą się tylko do pól MQMD. W przypadku, gdy właściwość występuje zarówno w deskryptorze MQMD, jak i w nagłówku MQRFH2 , wersja w nagłówku MQRFH2 nie jest ustawiana ani wyodrębniana.

Można ustawić dowolną z tych właściwości, z wyjątkiem właściwości JMS_IBM_MQMD_BackoutCount. Każda wartość ustawiona dla JMS_IBM_MQMD_BackoutCount jest ignorowana.

Jeśli właściwość ma maksymalną długość i zostanie podana wartość, która jest zbyt długa, wartość zostanie obcięta.

W przypadku niektórych właściwości należy również ustawić właściwość WMQ_MQMD_MESSAGE_CONTEXT w obiekcie docelowym. Aby ta właściwość miała zastosowanie, aplikacja musi być uruchamiana z odpowiednimi uprawnieniami dotyczącymi kontekstu. Jeśli właściwość WMQ_MQMD_MESSAGE_CONTEXT nie zostanie ustawiona na odpowiednią wartość, zostanie ona zignorowana. Jeśli właściwość WMQ_MQMD_MESSAGE_CONTEXT zostanie ustawiona na odpowiednią wartość, ale użytkownik nie ma wystarczających uprawnień kontekstowych dla menedżera kolejek, zostanie zgłoszony wyjątek JMSException. Właściwości wymagające konkretnych wartości właściwości WMQ_MQMD_MESSAGE_CONTEXT są następujące.

Następujące właściwości wymagają ustawienia właściwości WMQ_MQMD_MESSAGE_CONTEXT na wartość WMQ_MDCTX_SET_IDENTITY_CONTEXT lub WMQ_MDCTX_SET_ALL_CONTEXT:

- JMS_IBM_MQMD_UserIdentifier
- JMS_IBM_MQMD_AccountingToken
- Dane JMS_IBM_MQMD_ApplIdentity

Następujące właściwości wymagają, aby właściwość WMQ_MQMD_MESSAGE_CONTEXT była ustawiona na wartość WMQ_MDCTX_SET_ALL_CONTEXT:

- JMS_IBM_MQMD_PutApplTyp
- JMS_IBM_MQMD_PutApplNazwa
- JMS_IBM_MQMD_PutDate
- JMS_IBM_MQMD_PutTime
- Dane JMS_IBM_MQMD_ApplOrigin

## Odbieranie komunikatów

Wszystkie te właściwości są dostępne w odebranym komunikacie, jeśli właściwość WMQ_MQMD_READ_ENABLED ma wartość true, niezależnie od rzeczywistych właściwości ustawionych przez aplikację generującą. Aplikacja nie może modyfikować właściwości odebranego komunikatu, chyba że wszystkie właściwości zostaną najpierw wyczyszczone zgodnie ze specyfikacją JMS . Odebrany komunikat można przekazać bez modyfikowania właściwości.

**Ostrzeżenie:** Jeśli aplikacja odbierze komunikat z miejsca docelowego z właściwością WMQ_MQMD_READ_ENABLED ustawioną na wartość true i przekaże go do miejsca docelowego z właściwością WMQ_MQMD_WRITE_ENABLED ustawioną na wartość true, spowoduje to skopiowanie wszystkich wartości pól MQMD odebranego komunikatu do przekazanego komunikatu.

## Tabela właściwości

W tej tabeli przedstawiono właściwości obiektu Message reprezentującego pola MQMD. Pełne opisy pól i ich dozwolonych wartości można znaleźć w odsyłaczach.

*Tabela 38. Nazwy, opisy i typy właściwości*

| Właściwość | Opis | Java Typ | Odsyłacz do pełnego opisu |
|---|---|---|---|
| Raport JMS_IBM_MQMD_Report | Opcje dla komunikatów raportu | Liczba całkowita | Raport |
| JMS_IBM_MQMD_MsgType | Typ komunikatu | Liczba całkowita | MsgType |
| JMS_IBM_MQMD_Utrata ważności | Czas życia komunikatu | Liczba całkowita | Utrata ważności |
| Opinia JMS_IBM_MQMD_Feedback | Informacja zwrotna lub kod przyczyny | Liczba całkowita | Opinie |
| Kodowanie JMS_IBM_MQMD_Encoding | Kodowanie liczbowe danych komunikatu | Liczba całkowita | Kodowanie |
| JMS_IBM_MQMD_CodedCharSetId | Identyfikator zestawu znaków danych komunikatu | Liczba całkowita | CodedCharSetId |
| Format JMS_IBM_MQMD_Format | Nazwa formatu danych komunikatu | Łańcuch | Formatowanie |
| JMS_IBM_MQMD_Priority [1] | Priorytet komunikatu | Liczba całkowita | Priorytet |
| Trwałość JMS_IBM_MQMD_Persistence | Trwałość komunikatu | Liczba całkowita | Trwałość |
| JMS_IBM_MQMD_MsgId [2] | Identyfikator komunikatu | Obiekt (byte []) [4] | MsgId |
| JMS_IBM_MQMD_CorrelId [3] | Identyfikator korelacji | Obiekt (byte []) [4] | CorrelId |

*Tabela 38. Nazwy, opisy i typy właściwości (kontynuacja)*

| Właściwość | Opis | Java Typ | Odsyłacz do pełnego opisu |
|---|---|---|---|
| JMS_IBM_MQMD_BackoutCount | Liczba wycofanych | Liczba całkowita | BackoutCount |
| JMS_IBM_MQMD_ReplyToQ | Nazwa kolejki odpowiedzi | Łańcuch | Kolejka_zwrotna |
| Menedżer kolejek JMS_IBM_MQMD_ReplyTo | Nazwa menedżera kolejek odpowiedzi | Łańcuch | ReplyToQMgr |
| JMS_IBM_MQMD_UserIdentifier | Identyfikator użytkownika | Łańcuch | UserIdentifier |
| JMS_IBM_MQMD_AccountingToken | Token rozliczania | Obiekt (byte []) [4] | AccountingToken |
| Dane JMS_IBM_MQMD_ApplIdentity | Dane aplikacji odnoszące się do tożsamości | Łańcuch | Dane_tożsamości_aplikacji |
| JMS_IBM_MQMD_PutApplTyp | Typ aplikacji, która umieściła komunikat | Liczba całkowita | Typ_aplikacji_wstawiającej |
| JMS_IBM_MQMD_PutApplNazwa | Nazwa aplikacji, która umieściła komunikat | Łańcuch | Nazwa_aplikacji_wstawiającej |
| JMS_IBM_MQMD_PutDate | Data umieszczenia komunikatu | Łańcuch | PutDate |
| JMS_IBM_MQMD_PutTime | Czas umieszczenia komunikatu | Łańcuch | PutTime |
| Dane JMS_IBM_MQMD_ApplOrigin | Dane aplikacji odnoszące się do pochodzenia | Łańcuch | Dane_pochodzenia_aplikacji |
| JMS_IBM_MQMD_GroupId | Identyfikator grupy | Obiekt (byte []) [4] | GroupId |
| Numer JMS_IBM_MQMD_MsgSeq | Numer kolejny komunikatu logicznego w grupie | Liczba całkowita | Numer_kolejny_komunikatu |
| Przesunięcie JMS_IBM_MQMD_Offset | Przesunięcie danych w komunikacie fizycznym od początku komunikatu logicznego | Liczba całkowita | Depozycja |
| JMS_IBM_MQMD_MsgFlags | Flagi komunikatów | Liczba całkowita | MsgFlags |
| JMS_IBM_MQMD_OriginalLength | Długość oryginalnego komunikatu | Liczba całkowita | OriginalLength |

1. **Ostrzeżenie:** Jeśli do JMS_IBM_MQMD_Priority zostanie przypisana wartość spoza zakresu 0-9, narusza to specyfikację JMS .
2. **Ostrzeżenie:** Specyfikacja JMS określa, że identyfikator komunikatu musi być ustawiony przez dostawcę JMS i musi być unikalny lub mieć wartość NULL. Jeśli wartość zostanie przypisana do JMS_IBM_MQMD_MsgId, zostanie ona skopiowana do identyfikatora JMSMessageID. Dlatego

nie jest on ustawiany przez dostawcę JMS i może nie być unikalny: narusza to specyfikację JMS .

3. **Ostrzeżenie:** Jeśli do JMS_IBM_MQMD_CorrelId zostanie przypisana wartość rozpoczynająca się od łańcucha 'ID:', jest to niezgodne ze specyfikacją JMS .

4. **Ostrzeżenie:** Użycie właściwości tablicy bajtów w komunikacie jest niezgodne ze specyfikacją JMS .

*Uzyskiwanie dostępu do danych komunikatu IBM MQ z aplikacji za pomocą programu IBM MQ classes for JMS*
Dostęp do pełnych danych komunikatu IBM MQ w aplikacji można uzyskać za pomocą programu IBM MQ classes for JMS. Aby uzyskać dostęp do wszystkich danych, komunikat musi być JMSBytesMessage. Treść pliku JMSBytesMessage zawiera dowolny nagłówek MQRFH2 , wszystkie inne nagłówki IBM MQ oraz następujące dane komunikatu.

Ustaw właściwość WMQ_MESSAGE_BODY miejsca docelowego na wartość WMQ_MESSAGE_BODY_MQ, aby odbierać wszystkie dane treści komunikatu w JMSBytesMessage.

Jeśli właściwość WMQ_MESSAGE_BODY jest ustawiona na wartość WMQ_MESSAGE_BODY_JMS lub WMQ_MESSAGE_BODY_UNSPECIFIED, treść komunikatu jest zwracana bez nagłówka JMS MQRFH2 , a właściwości JMSBytesMessage odzwierciedlają właściwości ustawione w pliku RFH2.

Niektóre aplikacje nie mogą korzystać z funkcji opisanych w tym temacie. Jeśli aplikacja jest połączona z menedżerem kolejek produktu IBM MQ V6 lub jeśli właściwość PROVIDERVERSION ma wartość 6, funkcje nie są dostępne.

## Wysyłanie komunikatu

Podczas wysyłania komunikatów właściwość miejsca docelowego WMQ_MESSAGE_BODYma pierwszeństwo przed właściwością WMQ_TARGET_CLIENT.

Jeśli właściwość WMQ_MESSAGE_BODY ma wartość WMQ_MESSAGE_BODY_JMS, produkt IBM MQ classes for JMS automatycznie generuje nagłówek MQRFH2 na podstawie ustawień właściwości JMSMessage i pól nagłówka.

Jeśli właściwość WMQ_MESSAGE_BODY ma wartość WMQ_MESSAGE_BODY_MQ, do treści komunikatu nie jest dodawany żaden dodatkowy nagłówek.

Jeśli właściwość WMQ_MESSAGE_BODY ma wartość WMQ_MESSAGE_BODY_UNSPECIFIED, produkt IBM MQ classes for JMS wysyła nagłówek MQRFH2 , chyba że właściwość WMQ_TARGET_CLIENT ma wartość WMQ_TARGET_DEST_MQ. Po odebraniu ustawienie właściwości WMQ_TARGET_CLIENT na wartość WMQ_TARGET_DEST_MQ powoduje usunięcie dowolnego MQRFH2 z treści komunikatu.

**Uwaga:** Systemy JMSBytesMessage i JMSTextMessage nie wymagają systemu MQRFH2, natomiast systemy JMSStreamMessage, JMSMapMessagei JMSObjectMessage nie.

WMQ_MESSAGE_BODY_UNSPECIFIED jest domyślnym ustawieniem dla WMQ_MESSAGE_BODY, a WMQ_TARGET_DEST_JMS jest domyślnym ustawieniem dla WMQ_TARGET_CLIENT.

Jeśli zostanie wysłany komunikat JMSBytesMessage, można przesłonić ustawienia domyślne dla treści komunikatu JMS podczas tworzenia komunikatu IBM MQ . Użyj następujących właściwości:

- JMS_IBM_Format lub JMS_IBM_MQMD_Format: Ta właściwość określa format nagłówka IBM MQ lub ładunku aplikacji, który uruchamia treść komunikatu JMS , jeśli nie istnieje wcześniejszy nagłówek WebSphere MQ .
- JMS_IBM_Character_Set lub JMS_IBM_MQMD_CodedCharSetId: Ta właściwość określa wartość CCSID nagłówka IBM MQ lub ładunku aplikacji, która uruchamia treść komunikatu JMS , jeśli nie ma poprzedzającego nagłówka WebSphere MQ .
- JMS_IBM_Encoding lub JMS_IBM_MQMD_Encoding: ta właściwość określa kodowanie nagłówka IBM MQ lub ładunku aplikacji, które uruchamia treść komunikatu JMS , jeśli nie istnieje wcześniejszy nagłówek WebSphere MQ .

Jeśli określono oba typy właściwości, właściwości JMS_IBM_MQMD_* nadpisują odpowiednie właściwości JMS_IBM_* , o ile właściwość miejsca docelowego WMQ_MQMD_WRITE_ENABLED ma wartość true.

Różnice w działaniu między ustawieniem właściwości komunikatu za pomocą parametrów JMS_IBM_MQMD_* i JMS_IBM_* są istotne:

1. Właściwości JMS_IBM_MQMD_* są specyficzne dla dostawcy IBM MQ JMS .
2. Właściwości JMS_IBM_MQMD_* są ustawiane tylko w pliku MQMD. Właściwości JMS_IBM_* są ustawiane w pliku MQMD tylko wtedy, gdy komunikat nie zawiera nagłówka MQRFH2 JMS . W przeciwnym razie są one ustawiane w nagłówku JMS RFH2 .
3. Właściwości JMS_IBM_MQMD_* nie mają wpływu na kodowanie tekstu i liczb zapisanych w JMSMessage.

   Aplikacja odbierająca prawdopodobnie przyjmuje wartości MQMD.Encoding i MQMD.CodedCharSetId odpowiadające kodowaniu i zestawowi znaków liczb i tekstu w treści komunikatu. Jeśli używane są właściwości JMS_IBM_MQMD_* , odpowiedzialność za ich użycie spoczywa na aplikacji wysyłającej. Kodowanie i zestaw znaków dla liczb i tekstu w treści komunikatu są ustawiane przez właściwości JMS_IBM_* .

   Niepoprawnie zakodowany fragment kodu w pliku Rysunek 39 na stronie 258 wysyła komunikat zakodowany w zestawie znaków 1208 z wartością MQMD.CodedCharSetId ustawioną na 37.

   a. Wyślij błędnie zakodowany komunikat

   ```
   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. Odbieranie komunikatu w oparciu o wartość JMS_IBM_CHARACTER_SET ustawioną przez wartość MQMD.CodedCharSetId:

   ```
   TextMessage tmi = (TextMessage) cons.receive();
   System.out.println("Message is \"" + tmi.getText() + "\"");
   ```

   c. Wynik:

   ```
   Message is "éÈÊ'>...??>?"
   ```

*Rysunek 39. Niespójnie zakodowane dane MQMD i dane komunikatu*

Jeden z fragmentów kodu w pliku Rysunek 40 na stronie 259 powoduje umieszczenie komunikatu w kolejce lub temacie, a jego treść zawiera ładunek aplikacji bez automatycznie generowanego nagłówka MQRFH2 .

1. Ustawienie WMQ_MESSAGE_BODY_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

2. Ustawienie WMQ_TARGET_DEST_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);
((MQDestination) destination).
                setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
```

*Rysunek 40. Wyślij komunikat z treścią komunikatu MQ .*

## Odbieranie komunikatu

Jeśli właściwość WMQ_MESSAGE_BODY ma wartość WMQ_MESSAGE_BODY_JMS, typ i treść komunikatu przychodzącego JMS są określane przez treść odebranego komunikatu WebSphere MQ . Typ i treść komunikatu są określane przez pola w nagłówku MQRFH2 lub w pliku MQMD, jeśli nie ma elementu MQRFH2.

Jeśli opcja WMQ_MESSAGE_BODY ma wartość WMQ_MESSAGE_BODY_MQ, typem komunikatu przychodzącego JMS jest JMSBytesMessage. Treść komunikatu JMS to dane komunikatu zwracane przez bazowe wywołanie funkcji API MQGET . Długość treści komunikatu jest równa długości zwróconej przez wywołanie MQGET . Zestaw znaków i kodowanie danych w treści komunikatu są określane przez pola CodedCharSetId i Encoding w pliku MQMD. Format danych w treści komunikatu jest określany przez pole Format w pliku MQMD .

Jeśli właściwość WMQ_MESSAGE_BODY jest ustawiona na wartość WMQ_MESSAGE_BODY_UNSPECIFIED, wartość domyślna jest ustawiana przez parametr IBM MQ classes for JMS na wartość WMQ_MESSAGE_BODY_JMS.

Po otrzymaniu kodu JMSBytesMessagemożna go zdekodować, odwołując się do następujących właściwości:

- JMS_IBM_Format lub JMS_IBM_MQMD_Format: Ta właściwość określa format nagłówka IBM MQ lub ładunku aplikacji, który uruchamia treść komunikatu JMS , jeśli nie istnieje wcześniejszy nagłówek WebSphere MQ .
- JMS_IBM_Character_Set lub JMS_IBM_MQMD_CodedCharSetId: Ta właściwość określa wartość CCSID nagłówka IBM MQ lub ładunku aplikacji, która uruchamia treść komunikatu JMS , jeśli nie ma poprzedzającego nagłówka WebSphere MQ .
- JMS_IBM_Encoding lub JMS_IBM_MQMD_Encoding: ta właściwość określa kodowanie nagłówka IBM MQ lub ładunku aplikacji, które uruchamia treść komunikatu JMS , jeśli nie istnieje wcześniejszy nagłówek WebSphere MQ .

Poniższy fragment kodu powoduje wyświetlenie odebranego komunikatu, którym jest JMSBytesMessage. Niezależnie od treści odebranego komunikatu i pola formatu odebranego komunikatu MQMDkomunikat ma postać JMSBytesMessage.

```
((MQDestination)destination).setMessageBodyStyle
                            (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

*Właściwość miejsca docelowego WMQ_MESSAGE_BODY*
WMQ_MESSAGE_BODY określa, czy aplikacja JMS przetwarza MQRFH2 komunikatu IBM MQ jako część ładunku komunikatu (czyli jako część treści komunikatu JMS ).

*Tabela 39. Nazwy i opisy właściwości*

| Właściwość | Postać krótka | Opis |
|---|---|---|
| WMQ_MESSAGE_BODY | MBODY | Określa, czy aplikacja JMS przetwarza MQRFH2 komunikatu IBM MQ jako część ładunku komunikatu (czyli jako część treści komunikatu JMS ). |

*Tabela 40. Nazwy, wartości i metody ustawiania właściwości*

| Właściwość | Poprawne wartości w narzędziu administracyjnym (wartości domyślne pogrubione) | Poprawne wartości w programach | Ustaw metodę |
|---|---|---|---|
| KOMUNIKAT WMQ_MESSAGE _BODY | • **Nieokreślone**<br>Podczas wysyłania produkt IBM MQ classes for JMS generuje lub nie generuje i dołącza nagłówek MQRFH2 , w zależności od wartości właściwości WMQ_TARGET_CLIENT.<br>Podczas odbierania działa jako wartość JMS.<br>• JMS<br>Podczas wysyłania produkt IBM MQ classes for JMS automatycznie generuje nagłówek MQRFH2 i dołącza go do komunikatu IBM MQ .<br>Podczas odbierania produkt IBM MQ classes for JMS ustawia właściwości komunikatu JMS zgodnie z wartościami w nagłówku MQRFH2 (jeśli istnieje). Nie przedstawia on nagłówka MQRFH2 jako części treści komunikatu JMS .<br>• MQ<br>Podczas wysyłania produkt IBM MQ classes for JMS nie generuje nagłówka MQRFH2.<br>Podczas odbierania produkt IBM MQ classes for JMS prezentuje MQRFH2 jako część treści komunikatu JMS . | • **WMQ_MESSAGE_ BODY_UNSPECIFIED**<br>• WMQ_MESSAGE_BODY_JMS<br>• WMQ_MESSAGE_BODY_MQ | setMessageBodySty le |

*Trwałe komunikaty JMS*
Aplikacje IBM MQ classes for JMS mogą używać atrybutu kolejki **NonPersistentMessageClass** w celu zapewnienia lepszej wydajności dla trwałych komunikatów JMS , kosztem pewnej niezawodności.

Kolejka IBM MQ ma atrybut o nazwie **NonPersistentMessageClass**. Wartość tego atrybutu określa, czy nietrwałe komunikaty w kolejce są usuwane podczas restartowania menedżera kolejek.

Atrybut kolejki lokalnej można ustawić za pomocą komendy skryptowej IBM MQ (MQSC) DEFINE QLOCAL z jednym z następujących parametrów:

**NPMCLASS (NORMALNY)**
Nietrwałe komunikaty w kolejce są usuwane podczas restartowania menedżera kolejek. Jest to wartość domyślna.

**NPMCLASS (WYSOKI)**
Nietrwałe komunikaty w kolejce nie są usuwane podczas restartowania menedżera kolejek po wyciszeniu lub natychmiastowym zamknięciu systemu. Komunikaty nietrwałe mogą być jednak usuwane po zamknięciu systemu z wywłaszczaniem lub niepowodzeniu.

W tym temacie opisano, w jaki sposób aplikacje IBM MQ classes for JMS mogą używać tego atrybutu kolejki w celu zapewnienia lepszej wydajności dla trwałych komunikatów JMS .

Właściwość PERSISTENCE obiektu kolejki lub tematu może mieć wartość HIGH. Do ustawienia tej wartości można użyć narzędzia administracyjnego produktu IBM MQ JMS . Aplikacja może również wywołać metodę Destination.setPersistence(), przekazując jako parametr wartość WMQConstants.WMQ_PER_NPHIGH .

Jeśli aplikacja wysyła komunikat trwały JMS lub komunikat nietrwały JMS do miejsca docelowego, w którym właściwość PERSISTENCE ma wartość HIGH, a bazowa kolejka IBM MQ jest ustawiona na NPMCLASS (HIGH), komunikat jest umieszczany w kolejce jako komunikat nietrwały IBM MQ . Jeśli właściwość PERSISTENCE miejsca docelowego nie ma wartości HIGH lub jeśli kolejka bazowa jest ustawiona na NPMCLASS (NORMAL), komunikat trwały JMS jest umieszczany w kolejce jako komunikat trwały IBM MQ , a komunikat nietrwały JMS jest umieszczany w kolejce jako komunikat nietrwały IBM MQ .

Jeśli komunikat trwały JMS jest umieszczany w kolejce jako komunikat nietrwały IBM MQ , a użytkownik chce się upewnić, że komunikat nie zostanie usunięty po wyciszeniu lub natychmiastowym zamknięciu menedżera kolejek, wszystkie kolejki, przez które komunikat może być kierowany, muszą być ustawione na wartość NPMCLASS (HIGH). W domenie publikowania/subskrypcji kolejki te obejmują kolejki subskrybentów. Jako pomoc w wymuszaniu tej konfiguracji, IBM MQ classes for JMS zgłasza wyjątek InvalidDestination, jeśli aplikacja próbuje utworzyć konsument komunikatów dla miejsca docelowego, w którym właściwość PERSISTENCE ma wartość HIGH, a bazowa kolejka IBM MQ jest ustawiona na NPMCLASS (NORMAL).

Ustawienie właściwości PERSISTENCE miejsca docelowego na wartość HIGH nie ma wpływu na sposób odbierania komunikatu z tego miejsca docelowego. Komunikat wysłany jako komunikat trwały systemu JMS jest odbierany jako komunikat trwały systemu JMS , a komunikat wysłany jako komunikat nietrwały systemu JMS jest odbierany jako komunikat nietrwały systemu JMS .

Gdy aplikacja wysyła pierwszy komunikat do miejsca docelowego, w którym właściwość PERSISTENCE ma wartość HIGH, lub gdy aplikacja tworzy pierwszy konsument komunikatów dla miejsca docelowego, w którym właściwość PERSISTENCE ma wartość HIGH, IBM MQ classes for JMS wysyła wywołanie MQINQ w celu określenia, czy w bazowej kolejce IBM MQ ustawiono wartość NPMCLASS (HIGH). Dlatego aplikacja musi mieć uprawnienie do odpytywania kolejki. Ponadto produkt IBM MQ classes for JMS zachowuje wynik wywołania MQINQ do momentu usunięcia miejsca docelowego i nie wywołuje więcej wywołań MQINQ. Dlatego w przypadku zmiany ustawienia NPMCLASS w bazowej kolejce, gdy aplikacja nadal używała miejsca docelowego, produkt IBM MQ classes for JMS nie zauważa nowego ustawienia.

Zezwalając na umieszczanie komunikatów trwałych JMS w kolejkach systemu IBM MQ jako komunikatów nietrwałych IBM MQ , można uzyskać wydajność kosztem pewnej niezawodności. Jeśli wymagana jest maksymalna niezawodność komunikatów trwałych produktu JMS , nie należy wysyłać komunikatów do miejsca docelowego, w którym właściwość PERSISTENCE ma wartość HIGH.

Warstwa JMS może korzystać z systemu SYSTEM.JMS.TEMPQ.MODEL, a nie SYSTEM.DEFAULT.MODEL.QUEUE. SYSTEM SYSTEM.JMS.TEMPQ.MODEL tworzy trwałe kolejki dynamiczne, które akceptują komunikaty trwałe, ponieważ SYSTEM.DEFAULT.MODEL.QUEUE nie może zaakceptować trwałych komunikatów. Aby użyć kolejek tymczasowych do akceptowania komunikatów

trwałych, należy użyć systemu SYSTEM.JMS.TEMPQ.MODELlub zmień kolejkę modelową na wybraną kolejkę alternatywną.

*Używanie protokołu TLS z produktem IBM MQ classes for JMS*
Aplikacje IBM MQ classes for JMS mogą używać szyfrowania TLS (Transport Layer Security). W tym celu wymagany jest dostawca JSSE.

Połączenia IBM MQ classes for JMS używające protokołu TRANSPORT (CLIENT) obsługują szyfrowanie TLS. Protokół TLS zapewnia szyfrowanie komunikacji, uwierzytelnianie i integralność komunikatów. Jest on zwykle używany do zabezpieczania komunikacji między dwoma węzłami w sieci Internet lub w obrębie intranetu.

Produkt IBM MQ classes for JMS używa rozszerzenia Java Secure Socket Extension (JSSE) do obsługi szyfrowania TLS i dlatego wymaga dostawcy JSSE. JSE v1.4 Maszyny JVM mają wbudowanego dostawcę JSSE. Szczegóły dotyczące sposobu zarządzania certyfikatami i ich przechowywania mogą być różne w zależności od dostawcy. Więcej informacji na ten temat zawiera dokumentacja dostawcy JSSE.

W tej sekcji przyjęto założenie, że dostawca JSSE jest poprawnie zainstalowany i skonfigurowany oraz że odpowiednie certyfikaty zostały zainstalowane i udostępnione dostawcy JSSE. Teraz można użyć narzędzia JMSAdmin do ustawienia wielu właściwości administracyjnych.

Jeśli aplikacja IBM MQ classes for JMS używa tabeli definicji kanału klienta (CCDT) do nawiązania połączenia z menedżerem kolejek, należy zapoznać się z sekcją “Używanie tabeli definicji kanału klienta z produktem IBM MQ classes for JMS” na stronie 292.

*Właściwość obiektu SSLCIPHERSUITE*
Ustaw parametr SSLCIPHERSUITE, aby włączyć szyfrowanie TLS w obiekcie ConnectionFactory .

Aby włączyć szyfrowanie TLS w obiekcie ConnectionFactory , należy użyć narzędzia JMSAdmin do ustawienia właściwości SSLCIPHERSUITE na wartość CipherSuite obsługiwaną przez dostawcę JSSE. Wartość ta musi być zgodna z wartością CipherSpec ustawioną w kanale docelowym. Jednak CipherSuites różnią się od CipherSpecs i dlatego mają różne nazwy. Plik “TLS CipherSpecs i CipherSuites w produkcie IBM MQ classes for JMS” na stronie 265 zawiera tabelę z odwzorowaniem CipherSpecs obsługiwanych przez produkt IBM MQ na odpowiadające im CipherSuites znane w środowisku JSSE. Więcej informacji na temat CipherSpecs i CipherSuites z produktem IBM MQzawiera sekcja Zabezpieczanie produktu IBM MQ.

Na przykład, aby skonfigurować obiekt ConnectionFactory , który może być używany do tworzenia połączenia przez kanał MQI z włączoną obsługą TLS z wartością CipherSpec ustawioną na TLS_RSA_WITH_AES_128_CBC_SHA, należy wydać następującą komendę dla JMSAdmin:

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_WITH_AES_128_CBC_SHA)
```

Można ją również ustawić z poziomu aplikacji przy użyciu metody setSSLCipherSuite () w obiekcie MQConnectionFactory .

Dla wygody, jeśli we właściwości SSLCIPHERSUITE zostanie podana wartość CipherSpec , JMSAdmin podejmie próbę odwzorowania CipherSpec na odpowiedni CipherSuite i zgłosi ostrzeżenie. Ta próba odwzorowania nie jest podejmowana, jeśli właściwość jest określona przez aplikację.

Alternatywnie można użyć tabeli definicji kanału klienta (CCDT). Więcej informacji na ten temat zawiera “Używanie tabeli definicji kanału klienta z produktem IBM MQ classes for JMS” na stronie 292.

*Właściwość obiektu SSLFIPSREQUIRED*
Jeśli wymagane jest połączenie z użyciem pakietu CipherSuite obsługiwanego przez dostawcę JSSE FIPS IBM Java (IBMJSSEFIPS), należy ustawić właściwość SSLFIPSREQUIRED fabryki połączeń na wartość YES.

**Uwaga:** W systemie AIX, Linux, and Windows IBM MQ zapewnia zgodność ze standardem FIPS 140-2 za pośrednictwem modułu szyfrującego IBM Crypto for C (ICC) . Certyfikat dla tego modułu został przeniesiony do statusu historycznego. Klienci powinni zapoznać się z informacjami w sekcji Certyfikat IBM Crypto for C (ICC) i zapoznać się z poradami NIST. Zastępczy moduł FIPS 140-3 jest obecnie w toku, a jego status można wyświetlić, wyszukując go na liście Moduły NIST CMVP na liście procesów.

Wartością domyślną tej właściwości jest NO, co oznacza, że połączenie może korzystać z dowolnego CipherSuite obsługiwanego przez produkt IBM MQ.

Jeśli aplikacja używa więcej niż jednego połączenia, wartość SSLFIPSREQUIRED, która jest używana podczas tworzenia pierwszego połączenia, określa wartość, która jest używana podczas tworzenia kolejnego połączenia przez aplikację. Oznacza to, że wartość właściwości SSLFIPSREQUIRED fabryki połączeń, która jest używana do tworzenia kolejnego połączenia, jest ignorowana. Aby użyć innej wartości SSLFIPSREQUIRED, należy zrestartować aplikację.

Aplikacja może ustawić tę właściwość, wywołując metodę setSSLFipsRequired () obiektu ConnectionFactory . Ta właściwość jest ignorowana, jeśli nie ustawiono opcji CipherSuite .

**Zadania pokrewne**

Określenie, że w czasie wykonywania na kliencie MQI będą używane tylko CipherSpecs z certyfikatem FIPS.

**Odsyłacze pokrewne**

Standardy FIPS (Federal Information Processing Standards) dla AIX, Linux, and Windows

*Właściwość obiektu SSLPEERNAME*

Użyj parametru SSLPEERNAME, aby określić wzorzec nazwy wyróżniającej w celu zapewnienia, że aplikacja JMS nawiąże połączenie z poprawnym menedżerem kolejek.

Aplikacja JMS może zapewnić, że nawiąże połączenie z poprawnym menedżerem kolejek, określając wzorzec nazwy wyróżniającej (DN). Połączenie powiedzie się tylko wtedy, gdy menedżer kolejek przedstawi nazwę wyróżniającą (DN) zgodną ze wzorcem. Więcej informacji na temat formatu tego wzorca zawierają tematy pokrewne.

Nazwa wyróżniająca jest ustawiana przy użyciu właściwości SSLPEERNAME obiektu ConnectionFactory . Na przykład następująca komenda JMSAdmin ustawia obiekt ConnectionFactory tak, aby menedżer kolejek mógł się zidentyfikować przy użyciu nazwy zwykłej rozpoczynającej się od znaków `QMGR.` i z co najmniej dwiema nazwami jednostek organizacyjnych, z których pierwsza musi mieć wartość IBM , a druga musi mieć wartość WEBSPHERE:

```
ALTER CF(my.cf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

Podczas sprawdzania nie jest rozróżniana wielkość liter, a zamiast przecinków można używać średników. Parametr SSLPEERNAME można również ustawić z poziomu aplikacji przy użyciu metody setSSLPeerName () obiektu MQConnectionFactory . Jeśli ta właściwość nie jest ustawiona, nie jest wykonywane sprawdzanie nazwy wyróżniającej podanej przez menedżer kolejek. Ta właściwość jest ignorowana, jeśli nie ustawiono właściwości CipherSuite .

*Właściwość obiektu SSLCERTSTORES*

Użyj komendy SSLCERTSTORES, aby określić listę serwerów LDAP, które mają być używane do sprawdzania listy odwołań certyfikatów (CRL).

Powszechnie używa się listy odwołań certyfikatów (CRL) do identyfikowania certyfikatów, które nie są już zaufane. Listy CRL są zwykle udostępniane na serwerach LDAP. JMS umożliwia określenie serwera LDAP na potrzeby sprawdzania listy CRL w wersji Java 2 v1.4 lub nowszej. W poniższym przykładzie JMSAdmin nakazuje programowi JMS użycie listy CRL udostępnianej na serwerze LDAP o nazwie crl1.ibm.com:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

**Uwaga:** Aby pomyślnie użyć CertStore z listą CRL udostępnioną na serwerze LDAP, upewnij się, że pakiet Java Software Development Kit (SDK) jest zgodny z listą CRL. Niektóre pakiety SDK wymagają, aby lista CRL była zgodna z dokumentem RFC 2587, który definiuje schemat dla LDAP v2. Większość serwerów LDAP v3 używa standardu RFC 2256.

Jeśli serwer LDAP nie działa na domyślnym porcie 389, można podać port, dodając dwukropek (:) i numer portu do nazwy hosta. Jeśli certyfikat przedstawiony przez menedżera kolejek znajduje się na liście CRL udostępnionej w produkcie crl1.ibm.com, połączenie nie jest zakończone. Aby uniknąć pojedynczego

punktu awarii, produkt JMS umożliwia podanie wielu serwerów LDAP przez podanie listy serwerów LDAP rozdzielonych znakiem spacji. Oto przykład:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

Jeśli określono wiele serwerów LDAP, program JMS próbuje każdy z nich po kolei, dopóki nie znajdzie serwera, za pomocą którego może pomyślnie zweryfikować certyfikat menedżera kolejek. Każdy serwer musi zawierać identyczne informacje.

Łańcuch w tym formacie może zostać dostarczony przez aplikację w metodzie MQConnectionFactory.setSSLCertStores (). Alternatywnie aplikacja może utworzyć jeden lub więcej obiektów java.security.cert.CertStore , umieścić je w odpowiednim obiekcie kolekcji i dostarczyć ten obiekt kolekcji do metody setSSLCertStores (). W ten sposób aplikacja może dostosować sprawdzanie listy CRL. Szczegółowe informacje na temat tworzenia i używania obiektów CertStore można znaleźć w dokumentacji środowiska JSSE.

Poprawność certyfikatu prezentowanego przez menedżer kolejek podczas konfigurowania połączenia jest sprawdzana w następujący sposób:

1. Pierwszy obiekt CertStore w kolekcji identyfikowany przez magazyny sslCertjest używany do identyfikacji serwera CRL.
2. Podjęto próbę skontaktowania się z serwerem CRL.
3. Jeśli próba się powiedzie, serwer zostanie wyszukany pod kątem zgodności certyfikatu.
   a. Jeśli certyfikat zostanie unieważniony, proces wyszukiwania zostanie zakończony, a żądanie połączenia zakończy się niepowodzeniem z kodem przyczyny MQRC_SSL_CERTIFICATE_ODWOŁANE.
   b. Jeśli certyfikat nie zostanie znaleziony, proces wyszukiwania zostanie zakończony i połączenie będzie kontynuowane.
4. Jeśli próba nawiązania kontaktu z serwerem nie powiedzie się, następny obiekt CertStore zostanie użyty do zidentyfikowania serwera CRL, a proces zostanie powtórzony w kroku 2.

   Jeśli była to ostatnia CertStore w kolekcji lub jeśli kolekcja nie zawiera żadnych obiektów CertStore , proces wyszukiwania nie powiódł się i żądanie połączenia zakończyło się niepowodzeniem z kodem przyczyny MQRC_SSL_CERT_STORE_ERROR.

Obiekt Collection określa kolejność, w jakiej używane są CertStores .

Jeśli aplikacja używa metody setSSLCertStores () do ustawiania kolekcji obiektów CertStore , fabryki MQConnectionFactory nie można już powiązać w przestrzeni nazw JNDI. Próba wykonania tej czynności powoduje wystąpienie wyjątku. Jeśli właściwość magazynu sslCertnie jest ustawiona, nie jest wykonywane sprawdzanie odwołań dla certyfikatu udostępnionego przez menedżer kolejek. Ta właściwość jest ignorowana, jeśli nie ustawiono właściwości CipherSuite .

*Właściwość obiektu SSLRESETCOUNT*
Ta właściwość reprezentuje łączną liczbę bajtów wysłanych i odebranych przez połączenie przed ponownym negocjowaniem klucza tajnego używanego do szyfrowania.

Liczba wysłanych bajtów jest liczbą przed szyfrowaniem, a liczba odebranych bajtów jest liczbą po deszyfrowaniu. Liczba bajtów obejmuje również informacje sterujące wysyłane i odbierane przez IBM MQ classes for JMS.

Na przykład, aby skonfigurować obiekt ConnectionFactory , który może być używany do tworzenia połączenia przez kanał MQI z włączoną obsługą TLS z kluczem tajnym, który jest renegocjowany po przekazaniu 4 MB danych, należy wydać następującą komendę w narzędziu JMSAdmin:

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

Aplikacja może ustawić tę właściwość, wywołując metodę setSSLResetCount () obiektu ConnectionFactory .

Jeśli wartość tej właściwości wynosi zero, co jest wartością domyślną, klucz tajny nigdy nie jest renegocjowany. Ta właściwość jest ignorowana, jeśli nie ustawiono opcji CipherSuite .

*Właściwość obiektu SSLSocketFactory*
Aby dostosować inne aspekty połączenia TLS dla aplikacji, należy utworzyć fabrykę SSLSocketFactory i skonfigurować ją w produkcie JMS .

Istnieje możliwość dostosowania innych aspektów połączenia TLS dla aplikacji. Na przykład można zainicjować sprzęt szyfrujący lub zmienić używany magazyn kluczy i magazyn zaufanych certyfikatów. W tym celu aplikacja musi najpierw utworzyć obiekt javax.net.ssl.SSLSocketFactory , który został odpowiednio dostosowany. Informacje na ten temat można znaleźć w dokumentacji JSSE, ponieważ opcje dostosowywalne różnią się w zależności od dostawcy. Po uzyskaniu odpowiedniego obiektu SSLSocketFactory należy użyć metody MQConnectionFactory.setSSLSocketFactory () do skonfigurowania produktu JMS w celu użycia dostosowanego obiektu SSLSocketFactory .

Jeśli aplikacja używa metody setSSLSocketFactory () do ustawienia dostosowanego obiektu SSLSocketFactory , obiektu MQConnectionFactory nie można już powiązać z przestrzenią nazw JNDI. Próba wykonania tej czynności powoduje wystąpienie wyjątku. Jeśli ta właściwość nie jest ustawiona, używany jest domyślny obiekt SSLSocketFactory . Szczegółowe informacje na temat zachowania domyślnego obiektu SSLSocketFactory można znaleźć w dokumentacji rozszerzenia JSSE. Ta właściwość jest ignorowana, jeśli nie ustawiono właściwości CipherSuite .

**Ważne:** Nie należy zakładać, że użycie właściwości SSL zapewnia bezpieczeństwo podczas pobierania obiektu ConnectionFactory z przestrzeni nazw JNDI, która sama nie jest bezpieczna. W szczególności standardowa implementacja LDAP interfejsu JNDI nie jest bezpieczna. Atakujący może naśladować serwer LDAP, myląc aplikację JMS w celu nawiązania połączenia z niewłaściwym serwerem bez powiadomienia. Dzięki odpowiednim mechanizmom zabezpieczeń inne implementacje interfejsu JNDI (takie jak implementacja fscontext) są bezpieczne.

*Wprowadzanie zmian w magazynie kluczy lub magazynie zaufanych certyfikatów JSSE*
Po wprowadzeniu zmian w magazynie kluczy lub magazynie zaufanych certyfikatów należy podjąć pewne działania, aby zmiany zostały uwzględnione.

Jeśli zawartość magazynu kluczy lub magazynu zaufanych certyfikatów JSSE zostanie zmieniona lub zostanie zmienione położenie pliku kluczy lub magazynu zaufanych certyfikatów, aplikacje IBM MQ classes for JMS , które są uruchomione w tym czasie, nie będą automatycznie wprowadzać zmian. Aby zmiany odniosły skutek, należy wykonać następujące czynności:

- Aplikacje muszą zamknąć wszystkie swoje połączenia i zniszczyć nieużywane połączenia w pulach połączeń.
- Jeśli dostawca JSSE buforuje informacje z magazynu kluczy i magazynu zaufanych certyfikatów, należy je odświeżyć.

Po wykonaniu tych działań aplikacje mogą ponownie utworzyć swoje połączenia.

W zależności od sposobu projektowania aplikacji i funkcji udostępnianej przez dostawcę JSSE, może być możliwe wykonanie tych działań bez zatrzymywania i restartowania aplikacji. Jednak zatrzymanie i zrestartowanie aplikacji może być najprostszym rozwiązaniem.

*TLS CipherSpecs i CipherSuites w produkcie IBM MQ classes for JMS*
Możliwość nawiązywania połączeń z menedżerem kolejek przez aplikacje produktu IBM MQ classes for JMS zależy od parametru CipherSpec określonego na końcu kanału MQI serwera i parametru CipherSuite określonego na końcu klienta.

Poniższa tabela zawiera listę CipherSpecs obsługiwanych przez produkt IBM MQ oraz ich odpowiedniki w sekcji CipherSuites.

Deprecated Należy zapoznać się z tematem Nieaktualne specyfikacje szyfrowania CipherSpecs , aby sprawdzić, czy którekolwiek z CipherSpecswymienionych w poniższej tabeli zostały uznane za nieaktualne przez produkt IBM MQ , a jeśli tak, to aktualizacja specyfikacji szyfrowania CipherSpec stała się nieaktualna.

**Ważne:** Wymienione CipherSuites są obsługiwane przez środowisko wykonawcze IBM Java Runtime Environment (JRE) dostarczane z produktem IBM MQ. Wymienione CipherSuites obejmują te, które są obsługiwane przez środowisko Oracle Java JRE. Więcej informacji na temat konfigurowania aplikacji do używania środowiska Oracle Java JRE zawiera sekcja Konfigurowanie aplikacji do używania odwzorowań IBM Java lub Oracle Java CipherSuite.

Tabela zawiera również informacje o protokole używanym do komunikacji oraz o tym, czy pakiet CipherSuite jest zgodny ze standardem FIPS 140-2.

**Uwaga:** W systemie AIX, Linux, and Windows IBM MQ zapewnia zgodność ze standardem FIPS 140-2 za pośrednictwem modułu szyfrującego IBM Crypto for C (ICC) . Certyfikat dla tego modułu został przeniesiony do statusu historycznego. Klienci powinni zapoznać się z informacjami w sekcji Certyfikat IBM Crypto for C (ICC) i zapoznać się z poradami NIST. Zastępczy moduł FIPS 140-3 jest obecnie w toku, a jego status można wyświetlić, wyszukując go na liście Moduły NIST CMVP na liście procesów.

Zestawy algorytmów szyfrowania oznaczone jako zgodne ze standardem FIPS 140-2 mogą być używane, jeśli aplikacja nie została skonfigurowana do wymuszania zgodności ze standardem FIPS 140-2, ale jeśli dla aplikacji została skonfigurowana zgodność ze standardem FIPS 140-2 (patrz poniższe uwagi dotyczące konfiguracji), można skonfigurować tylko te CipherSuites , które są oznaczone jako zgodne ze standardem FIPS 140-2. Próba użycia innych CipherSuites spowoduje wystąpienie błędu.

**Uwaga:** Każde środowisko JRE może mieć wielu dostawców zabezpieczeń szyfrowania, z których każdy może wnosić implementację tego samego pakietu CipherSuite. Jednak nie wszyscy dostawcy zabezpieczeń mają certyfikat FIPS 140-2. Jeśli zgodność ze standardem FIPS 140-2 nie jest wymuszana w przypadku aplikacji, możliwe jest użycie niecertyfikowanej implementacji CipherSuite . Implementacje niecertyfikowane mogą nie działać zgodnie ze standardem FIPS 140-2, nawet jeśli zestaw algorytmów szyfrowania CipherSuite teoretycznie spełnia minimalny poziom zabezpieczeń wymagany przez standard. Więcej informacji na temat konfigurowania wymuszania FIPS 140-2 w aplikacjach IBM MQ JMS zawierają poniższe uwagi.

Więcej informacji na temat zgodności ze standardami FIPS 140-2 i Suite-B dla CipherSpecs i CipherSuiteszawiera sekcja Określanie specyfikacji szyfrowania CipherSpecs. Konieczne może być również zapoznanie się z informacjami dotyczącymi Federal Information Processing Standardsw Stanach Zjednoczonych.

Aby używać pełnego zestawu algorytmów szyfrowania CipherSuites i działać z certyfikowanym standardem FIPS 140-2 i/lub Suite-B, wymagane jest odpowiednie środowisko JRE. IBM Java 7 Service Refresh 4 z pakietem poprawek 2 lub nowszym IBM JRE zapewnia odpowiednią obsługę TLS 1.2 CipherSuites wymienionych w sekcji Tabela 41 na stronie 267.

Aby można było używać szyfrów TLS 1.3 , środowisko JRE, w którym działa aplikacja, musi obsługiwać protokół TLS 1.3.

**Uwaga:** Aby użyć niektórych CipherSuites, w środowisku JRE należy skonfigurować pliki nieograniczonej strategii. Więcej informacji na temat konfigurowania plików strategii w środowisku SDK lub JRE zawiera temat *IBM Pliki strategii SDK* w publikacji *Security Reference for IBM SDK, Java Technology Edition* dla używanej wersji.

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites*

| **CipherSpec** **"1" na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| ECDHE_ECDSA_NULL_SHA256 | SSL_ECDHE_ECDSA_WITH_NULL_SHA | TLS_ECDHE_ECDSA_WITH_NULL_SHA | TLS 1.2 | no |
| ECDHE_ECDSA_RC4_128_SHA256 | SSL_ECDHE_ECDSA_WITH_RC4_128_SHA | TLS_ECDHE_ECDSA_WITH_RC4_128_SHA | TLS 1.2 | no |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| ECDHE_RSA_NULL_SHA256 | SSL_ECDHE_RSA_WITH_NULL_SHA | TLS_ECDHE_RSA_WITH_NULL_SHA | TLS 1.2 | no |
| ECDHE_RSA_RC4_128_SHA256 | SSL_ECDHE_RSA_WITH_RC4_128_SHA | TLS_ECDHE_RSA_WITH_RC4_128_SHA | TLS 1.2 | no |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_RSA_WITH_3DES_EDE_CBC_SHA “2” na stronie 287 | SSL_RSA_WITH_3DES_EDE_CBC_SHA | TLS_RSA_WITH_3DES_EDE_CBC_SHA | TLS 1.0 | nie “4” na stronie 287 |
| TLS_RSA_WITH_AES_128_CBC_SHA | SSL_RSA_WITH_AES_128_CBC_SHA | TLS_RSA_WITH_AES_128_CBC_SHA | TLS 1.0 | nie “4” na stronie 287 |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_RSA_WITH_AES_128_CBC_SHA256 | SSL_RSA_WITH_AES_128_CBC_SHA256 | TLS_RSA_WITH_AES_128_CBC_SHA256 | TLS 1.2 | nie "4" na stronie 287 |
| TLS_RSA_WITH_AES_128_GCM_SHA256 | SSL_RSA_WITH_AES_128_GCM_SHA256 | TLS_RSA_WITH_AES_128_GCM_SHA256 | TLS 1.2 | nie "4" na stronie 287 |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_RSA_WITH_AES_256_CBC_SHA | SSL_RSA_WITH_AES_256_CBC_SHA | TLS_RSA_WITH_AES_256_CBC_SHA | TLS 1.0 | nie "4" na stronie 287 |
| TLS_RSA_WITH_AES_256_CBC_SHA256 | SSL_RSA_WITH_AES_256_CBC_SHA256 | TLS_RSA_WITH_AES_256_CBC_SHA256 | TLS 1.2 | nie "4" na stronie 287 |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_RSA_WITH_AES_256_GCM_SHA384 | SSL_RSA_WITH_AES_256_GCM_SHA384 | TLS_RSA_WITH_AES_256_GCM_SHA384 | TLS 1.2 | nie "4" na stronie 287 |
| TLS_RSA_WITH_DES_CBC_SHA | SSL_RSA_WITH_DES_CBC_SHA | SSL_RSA_WITH_DES_CBC_SHA | TLS 1.0 | no |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_RSA_WITH_NULL_SHA256 | SSL_RSA_WITH_NULL_SHA256 | TLS_RSA_WITH_NULL_SHA256 | TLS 1.2 | no |
| TLS_RSA_WITH_RC4_128_SHA256 | SSL_RSA_WITH_RC4_128_SHA | SSL_RSA_WITH_RC4_128_SHA | TLS 1.2 | no |
| ANY_TLS12 | *TLS12 | *TLS12 | TLS 1.2 | yes |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_AES_128_GCM_SHA256 "3" na stronie 287 | TLS_AES_128_GCM_SHA256 | TLS_AES_128_GCM_SHA256 | TLS 1.3 | no |
| TLS_AES_256_GCM_SHA384 "3" na stronie 287 | TLS_AES_256_GCM_SHA384 | TLS_AES_256_GCM_SHA384 | TLS 1.3 | no |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_CHACHA20_POLY1305_SHA256 “3” na stronie 287 | TLS_CHACHA20_POLY1305_SHA256 | TLS_CHACHA20_POLY1305_SHA256 | TLS 1.3 | no |
| TLS_AES_128_CCM_SHA256 “3” na stronie 287 | TLS_AES_128_CCM_SHA256 | TLS_AES_128_CCM_SHA256 | TLS 1.3 | no |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 287** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_AES_128_CCM_8_SHA256 "3" na stronie 287 | TLS_AES_128_CCM_8_SHA256 | TLS_AES_128_CCM_8_SHA256 | TLS 1.3 | no |
| Dowolna"3" na stronie 287 | *ANY | *ANY | Wielokrotny | no |
| ANY_TLS13 "3" na stronie 287 | *TLS13 | *TLS13 | TLS V13 | no |
| ANY_TLS12_OR_HIGHER "3" na stronie 287 | *TLS12ORHIGHER | *TLS12ORHIGHER | TLS 1.2 i nowsze | no |

*Tabela 41. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** “1” na stronie 287 | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| ANY_TLS13_OR_HIGHER “3” na stronie 287 | *TLS13ORHIGHER | *TLS13ORHIGHER | TLS 1.3 i nowsze | no |

**Uwagi:**

1. Jest to wartość skonfigurowana w kanale w pliku IBM MQ, w tym w tabeli CCDT (binarnej lub JSON).
2. Deprecated CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA jest nieaktualna. Jednak nadal może być używany do przesyłania do 32 GB danych, zanim połączenie zostanie przerwane i zostanie zgłoszony błąd AMQ9288. Aby uniknąć tego błędu, należy unikać używania potrójnego algorytmu szyfrowania DES lub włączyć resetowanie klucza tajnego podczas korzystania z tej CipherSpec.
3. Aby można było używać szyfrów TLS v1.3 , środowisko Java runtime environment (JRE), na którym działa aplikacja, musi obsługiwać protokół TLS v1.3.
4. V 9.3.0.17 V 9.3.5.1 W systemach IBM MQ 9.3.5 CSU 1 i IBM MQ 9.3.0 CSU 17środowisko IBM Java 8 JRE usuwa obsługę wymiany kluczy RSA podczas pracy w trybie FIPS.

## Konfigurowanie zestawów algorytmów szyfrowania i zgodności ze standardami FIPS w aplikacji IBM MQ classes for JMS

- Aplikacja, która używa produktu IBM MQ classes for JMS , może użyć jednej z dwóch metod ustawienia CipherSuite dla połączenia:
  - Wywołaj metodę setSSLCipherSuite obiektu ConnectionFactory .
  - Użyj narzędzia administracyjnego IBM MQ JMS , aby ustawić właściwość SSLCIPHERSUITE dla obiektu ConnectionFactory .
- Aplikacja, która używa IBM MQ classes for JMS , może użyć jednej z dwóch metod w celu wymuszenia zgodności ze standardem FIPS 140-2:
  - Wywołaj metodę setSSLFipswymaganą dla obiektu ConnectionFactory .

– Użyj narzędzia administracyjnego IBM MQ JMS , aby ustawić właściwość SSLFIPSREQUIRED obiektu ConnectionFactory .

## Konfigurowanie aplikacji do korzystania z odwzorowań IBM Java lub Oracle Java CipherSuite

**Uwaga:** V 9.3.3 W przypadku produktu Continuous Delivery z produktu IBM MQ 9.3.3 Java właściwość systemowa `com.ibm.mq.cfg.useIBMCipherMappings`, która steruje wykorzystywanym odwzorowaniem, jest usuwana z produktu. W produkcie IBM MQ 9.3.3szyfr może być zdefiniowany jako nazwa CipherSpec lub CipherSuite i jest poprawnie obsługiwany przez produkt IBM MQ. Następujące trzy pliki JAR Jacksona są wymagane, jeśli aplikacja IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging tworzy bezpieczne połączenia TLS z menedżerem kolejek:

- `jackson-annotations.jar`
- `jackson-core.jar`
- `jackson-databind.jar`

**Ważne:** Poniższe informacje o systemie `com.ibm.mq.cfg.useIBMCipherMappings` dotyczą tylko systemów Long Term Support i Continuous Delivery przed IBM MQ 9.3.3 .

Istnieje możliwość określenia, czy aplikacja ma używać domyślnych odwzorowań IBM Java CipherSuite na IBM MQ CipherSpec , czy odwzorowań Oracle CipherSuite na IBM MQ CipherSpec . Dlatego można użyć CipherSuites TLS, niezależnie od tego, czy aplikacja używa środowiska JRE firmy IBM , czy środowiska JRE firmy Oracle . Java Właściwość systemowa `com.ibm.mq.cfg.useIBMCipherMappings` określa, które odwzorowania są używane. Właściwość może mieć jedną z następujących wartości:

**Prawda**
Użyj odwzorowań IBM Java CipherSuite na IBM MQ CipherSpec .

Jest to wartość domyślna.

**Fałsz**
Użyj odwzorowań Oracle CipherSuite na IBM MQ CipherSpec .

Więcej informacji na temat używania szyfrów IBM MQ Java i TLS zawiera wpis na blogu MQdev MQ Java, TLS Ciphers, inne niżIBM JRE i APAR IT06775, IV66840, IT09423, IT10837.

## Ograniczenia współdziałania

Niektóre CipherSuites mogą być zgodne z więcej niż jedną specyfikacją szyfrowania produktu IBM MQ CipherSpec, w zależności od używanego protokołu. Jednak obsługiwana jest tylko kombinacja CipherSuite/CipherSpec , która używa wersji TLS określonej w tabeli 1. Próba użycia nieobsługiwanych kombinacji zestawów algorytmów szyfrowania ( CipherSuites i CipherSpecs ) nie powiedzie się i zostanie zgłoszony odpowiedni wyjątek. Instalacje używające dowolnej z tych kombinacji CipherSuite/CipherSpec powinny zostać przeniesione do obsługiwanej kombinacji.

W poniższej tabeli przedstawiono CipherSuites , do których ma zastosowanie to ograniczenie.

*Tabela 42. CipherSuites oraz ich obsługiwane i nieobsługiwane specyfikacje szyfrowania CipherSpecs*

| CipherSuite | Obsługiwany TLS CipherSpec | Nieobsługiwana CipherSpec SSL |
|---|---|---|
| SSL_RSA_WITH_3DES_EDE_CBC_SHA | TLS_RSA_WITH_3DES_EDE_CBC_SHA "1" na stronie 289 | 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 |

**Uwaga:**

1. Deprecated Ta CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA jest nieaktualna. Jednak nadal może być używany do przesyłania do 32 GB danych, zanim połączenie zostanie przerwane i zostanie zgłoszony błąd AMQ9288. Aby uniknąć tego błędu, należy unikać używania potrójnego algorytmu szyfrowania DES lub włączyć resetowanie klucza tajnego podczas korzystania z tej CipherSpec.

*Zapisywanie wyjść kanałów w produkcie Java for IBM MQ classes for JMS*
Wyjścia kanału są tworzone przez zdefiniowanie klas Java , które implementują określone interfejsy.

Wprowadzenie do wyjść zabezpieczeń można rozpocząć od tematu Programy obsługi wyjścia zabezpieczeń kanału .

W pakiecie com.ibm.mq.exits zdefiniowano trzy interfejsy:

- WMQSendExitdla wyjścia wysyłania
- WMQReceiveExitdla wyjścia odbierania
- WMQSecurityExitdla wyjścia zabezpieczeń

Poniższy kod przykładowy definiuje klasę, która implementuje wszystkie trzy interfejsy:

```
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
  }
}
```

Każde wyjście odbiera jako parametry obiekt MQCXP i obiekt MQCD. Te obiekty reprezentują struktury MQCXP i MQCD zdefiniowane w interfejsie proceduralnym.

Po wywołaniu wyjścia wysyłania parametr agentBuffer zawiera dane, które mają zostać wysłane do menedżera kolejek serwera. Parametr długości nie jest wymagany, ponieważ wyrażenie agentBuffer.limit () określa długość danych. Wyjście wysyłania zwraca jako wartość dane, które mają zostać wysłane do menedżera kolejek serwera. Jeśli jednak wyjście wysyłania nie jest ostatnim wyjściem wysyłania w sekwencji wyjść wysyłania, zwracane dane są przekazywane do następnego wyjścia wysyłania w sekwencji. Wyjście wysyłania może zwrócić zmodyfikowaną wersję danych odebranych w parametrze agentBuffer lub może zwrócić dane bez zmian. Najprostszym możliwym organem wyjścia jest zatem:

```
{ return agentBuffer; }
```

Po wywołaniu wyjścia odbierania parametr agentBuffer zawiera dane odebrane z menedżera kolejek serwera. Wyjście odbierania zwraca jako swoją wartość dane, które mają zostać przekazane do aplikacji przez program IBM MQ classes for JMS. Jeśli jednak wyjście odbierania nie jest ostatnim wyjściem

odbierania w sekwencji wyjść odbierania, zwracane dane są przekazywane do następnego wyjścia odbierania w sekwencji.

Po wywołaniu wyjścia zabezpieczeń parametr agentBuffer zawiera dane, które zostały odebrane w przepływie zabezpieczeń z wyjścia zabezpieczeń na końcu połączenia serwera. Wyjście zabezpieczeń zwraca jako wartość dane, które mają zostać wysłane w przepływie zabezpieczeń do wyjścia zabezpieczeń serwera.

Wyjścia kanału są wywoływane z buforem, który ma macierz bazową. Aby uzyskać najlepszą wydajność, wyjście powinno zwrócić bufor z tablicą bazową.

Do 32 znaków danych użytkownika można przekazać do wyjścia kanału podczas jego wywołania. Wyjście uzyskuje dostęp do danych użytkownika, wywołując metodę getExitData () obiektu MQCXP. Mimo że wyjście może zmienić dane użytkownika, wywołując metodę setExitData (), dane użytkownika są odświeżane przy każdym wywołaniu wyjścia. Wszelkie zmiany wprowadzone w danych użytkownika zostaną utracone. Jednak wyjście może przekazywać dane z jednego wywołania do następnego za pomocą obszaru użytkownika wyjścia obiektu MQCXP. Wyjście uzyskuje dostęp do obszaru użytkownika wyjścia przez wywołanie metody getExitUserArea().

Każda klasa wyjścia musi mieć konstruktor. Konstruktor może być konstruktorem domyślnym, jak pokazano w poprzednim przykładzie, lub konstruktorem z parametrem łańcuchowym. Konstruktor jest wywoływany w celu utworzenia instancji klasy wyjścia dla każdego wyjścia zdefiniowanego w klasie. Dlatego w poprzednim przykładzie utworzono instancję klasy MyMQExits dla wyjścia wysyłania, utworzono inną instancję dla wyjścia odbierania i utworzono trzecią instancję dla wyjścia zabezpieczeń. Po wywołaniu konstruktora z parametrem łańcuchowym parametr zawiera te same dane użytkownika, które są przekazywane do wyjścia kanału, dla którego tworzona jest instancja. Jeśli klasa wyjścia ma zarówno konstruktor domyślny, jak i konstruktor pojedynczego parametru, pierwszeństwo ma konstruktor pojedynczego parametru.

Nie zamykaj połączenia z poziomu wyjścia kanału.

Gdy dane są wysyłane do serwera po zakończeniu połączenia, szyfrowanie TLS jest wykonywane *po* wywołaniu wszystkich wyjść kanałów. Podobnie, gdy dane są odbierane z serwera po zakończeniu połączenia, deszyfrowanie TLS jest wykonywane *przed* wywołaniem dowolnego wyjścia kanału.

W wersjach produktu IBM MQ classes for JMS wcześniejszych niż IBM WebSphere MQ 7.0wyjścia kanału były implementowane przy użyciu interfejsów MQSendExit, MQReceiveExiti MQSecurityExit. Nadal można używać tych interfejsów, ale nowe interfejsy są preferowane w celu poprawy funkcji i wydajności.

*Konfigurowanie produktu IBM MQ classes for JMS do korzystania z wyjść kanałów*
Aplikacja produktu IBM MQ classes for JMS może korzystać z zabezpieczeń kanału, wyjść wysyłania i odbierania w kanale MQI, które są uruchamiane, gdy aplikacja łączy się z menedżerem kolejek. Aplikacja może używać wyjść napisanych w języku Java, C lub C + +. Aplikacja może również używać sekwencji wyjść wysyłania lub odbierania, które są uruchamiane po sobie.

Następujące właściwości są używane do określania wyjścia wysyłania lub sekwencji wyjść wysyłania używanych przez połączenie JMS :

- Właściwość **SENDEXIT** obiektu MQConnectionFactory .
- Właściwość **sendexit** w specyfikacji aktywowania używanej przez adapter zasobów IBM MQ na potrzeby komunikacji przychodzącej.
- Właściwość **sendexit** w obiekcie ConnectionFactory używana przez adapter zasobów IBM MQ do komunikacji wyjściowej.

Wartość właściwości jest łańcuchem składającym się z jednego lub większej liczby elementów oddzielonych przecinkami. Każdy element identyfikuje wyjście wysyłania w jeden z następujących sposobów:

- Nazwa klasy, która implementuje interfejs `WMQSendExit` dla wyjścia wysyłania napisanego w języku Java.
- Łańcuch w formacie *libraryName* (*entryPoint*) dla wyjścia wysyłania zapisanego w języku C lub C + +.

W podobny sposób następujące właściwości określają wyjście odbierania lub sekwencję wyjść odbierania używanych przez połączenie:

- Właściwość **RECEXIT** obiektu MQConnectionFactory .
- Właściwość **receiveexit** w specyfikacji aktywowania używanej przez adapter zasobów IBM MQ na potrzeby komunikacji przychodzącej.
- Właściwość **receiveexit** w obiekcie ConnectionFactory używana przez adapter zasobów IBM MQ do komunikacji wyjściowej.

Następujące właściwości określają wyjście zabezpieczeń używane przez połączenie:

- Właściwość **SECEXIT** obiektu MQConnectionFactory .
- Właściwość **securityexit** w specyfikacji aktywowania używanej przez adapter zasobów IBM MQ na potrzeby komunikacji przychodzącej.
- Właściwość **securityexit** w obiekcie ConnectionFactory używana przez adapter zasobów IBM MQ do komunikacji wyjściowej.

W przypadku MQConnectionFactoriesmożna ustawić właściwości **SENDEXIT**, **RECEXIT** i **SECEXIT** za pomocą narzędzia administracyjnego produktu IBM MQ JMS lub pliku IBM MQ Explorer. Alternatywnie aplikacja może ustawić właściwości, wywołując metody `setSendExit()`, `setReceiveExit()`i `setSecurityExit()` .

Wyjścia kanału są ładowane przez własny program ładujący klasy. Aby znaleźć wyjście kanału, program ładujący klasy przeszukuje następujące miejsca w podanej kolejności.

1. Ścieżka klasy określona przez właściwość **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** lub atrybut **JavaExitsClassPath** w sekcji Channels pliku konfiguracyjnego klienta IBM MQ .
2. Deprecated Ścieżka klasy określona przez Java właściwość systemową **com.ibm.mq.exitClasspath**. Należy zauważyć, że ta właściwość jest teraz nieaktualna.
3. Katalog wyjść IBM MQ , jak to pokazano na rysunku ( Tabela 43 na stronie 291). Program ładujący klasy najpierw wyszukuje w katalogu pliki klas, które nie są spakowane w plikach archiwum Java (JAR). Jeśli wyjście kanału nie zostanie znalezione, program ładujący klasy przeszukuje pliki JAR w katalogu.

*Tabela 43. Katalog wyjść IBM MQ*

| Platforma | Katalog |
|---|---|
| Linux AIX AIX and Linux | `/var/mqm/exits` (32-bitowe wyjścia kanału)<br>`/var/mqm/exits64` (64-bitowe wyjścia kanału) |
| Windows Windows | *katalog_danych_instalacji*\exits<br>gdzie *katalog_danych_instalacji* jest katalogiem wybranym dla plików danych IBM MQ podczas instalacji. Katalog domyślny to C:\ProgramData\IBM\MQ. |

**Uwaga:** Jeśli wyjście kanału istnieje w więcej niż jednym położeniu, program IBM MQ classes for JMS ładuje pierwszą znalezioną instancję.

Programem macierzystym programu ładującego klasy jest program ładujący klasy, który jest używany do ładowania pliku IBM MQ classes for JMS. W związku z tym możliwe jest załadowanie wyjścia kanału przez macierzysty program ładujący klasy, jeśli nie można go znaleźć w żadnym z powyższych miejsc. Jeśli jednak plik IBM MQ classes for JMS jest używany w środowisku, takim jak serwer aplikacji JEE , prawdopodobnie nie będzie można wpłynąć na wybór nadrzędnego programu ładującego klasy, dlatego należy skonfigurować program ładujący klasy, ustawiając Java właściwość systemową **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** na serwerze aplikacji.

Jeśli aplikacja jest uruchamiana z włączoną opcją Java security manager , plik konfiguracyjny strategii używany przez środowisko wykonawcze Java , w którym aplikacja jest uruchomiona, musi mieć

uprawnienia do ładowania klasy wyjścia kanału. Więcej informacji na ten temat zawiera sekcja Uruchamianie klas IBM MQ dla aplikacji JMS w programie Java Security Manager.

Interfejsy `MQSendExit`, `MQReceiveExit`i `MQSecurityExit` dostarczone z wersjami wcześniejszymi niż IBM WebSphere MQ 7.0 są nadal obsługiwane. Jeśli używane są wyjścia kanału, które implementują te interfejsy, w ścieżce klasy musi znajdować się łańcuch `com.ibm.mq.jar` .

Informacje na temat zapisywania wyjść kanałów w języku C zawiera sekcja "Kanał-programy obsługi wyjścia dla kanałów przesyłania komunikatów" na stronie 990. Należy zapisać programy obsługi wyjścia kanału napisane w języku C lub C++ w katalogu wyświetlonego na rysunku ( Tabela 43 na stronie 291).

Jeśli aplikacja używa tabeli definicji kanału klienta (CCDT) do nawiązywania połączenia z menedżerem kolejek, należy zapoznać się z sekcją "Używanie tabeli definicji kanału klienta z produktem IBM MQ classes for JMS" na stronie 292.

*Określanie danych użytkownika, które mają być przekazywane do wyjść kanałów przy użyciu funkcji IBM MQ classes for JMS*
Do 32 znaków danych użytkownika można przekazać do wyjścia kanału podczas jego wywołania.

Właściwość SENDEXITINIT obiektu MQConnectionFactory określa dane użytkownika, które są przekazywane do każdego wyjścia wysyłania podczas jego wywołania. Wartość właściwości jest łańcuchem składającym się z jednego lub większej liczby elementów danych użytkownika oddzielonych przecinkami. Pozycja każdego elementu danych użytkownika w łańcuchu określa, do którego wyjścia wysyłania, w sekwencji wyjść wysyłania, są przekazywane dane użytkownika. Na przykład pierwszy element danych użytkownika w łańcuchu jest przekazywany do pierwszego wyjścia wysyłania w sekwencji wyjść wysyłania.

Właściwość SENDEXITINIT można ustawić za pomocą narzędzia administracyjnego IBM MQ JMS lub IBM MQ Explorer. Alternatywnie aplikacja może ustawić tę właściwość, wywołując metodę setSendExitInit().

W podobny sposób właściwość RECEXITINIT obiektu ConnectionFactory określa dane użytkownika przekazywane do każdego wyjścia odbierania, a właściwość SECEXITINIT określa dane użytkownika przekazywane do wyjścia zabezpieczeń. Właściwości te można ustawić za pomocą narzędzia administracyjnego IBM MQ JMS lub pliku IBM MQ Explorer. Alternatywnie aplikacja może ustawić właściwości, wywołując metody setReceiveExitInit() i setSecurityExitInit().

Podczas określania danych użytkownika przekazywanych do wyjść kanału należy pamiętać o następujących regułach:

- Jeśli liczba elementów danych użytkownika w łańcuchu jest większa niż liczba wyjść w sekwencji, nadmiarowe elementy danych użytkownika są ignorowane.
- Jeśli liczba elementów danych użytkownika w łańcuchu jest mniejsza niż liczba wyjść w sekwencji, każdy nieokreślony element danych użytkownika jest ustawiany na pusty łańcuch. Dwa kolejne przecinki w łańcuchu lub przecinek na początku łańcucha oznaczają również nieokreślony element danych użytkownika.

Jeśli aplikacja używa tabeli definicji kanału klienta (CCDT) do nawiązywania połączenia z menedżerem kolejek, wszystkie dane użytkownika określone w definicji kanału połączenia klienta są przekazywane do wyjść kanału podczas ich wywoływania. Więcej informacji na temat używania tabeli definicji kanału klienta zawiera sekcja "Używanie tabeli definicji kanału klienta z produktem IBM MQ classes for JMS" na stronie 292.

*Używanie tabeli definicji kanału klienta z produktem IBM MQ classes for JMS*
Aplikacja IBM MQ classes for JMS może używać definicji kanału połączenia klienckiego przechowywanych w tabeli definicji kanału klienta (CCDT). Obiekt ConnectionFactory należy skonfigurować w taki sposób, aby korzystał z tabeli CCDT. Istnieją pewne ograniczenia dotyczące jego stosowania.

Zamiast tworzyć definicję kanału połączenia klienckiego, ustawiając określone właściwości obiektu ConnectionFactory , aplikacja IBM MQ classes for JMS może korzystać z definicji kanału połączenia klienckiego, które są przechowywane w tabeli definicji kanału klienta. Definicje te są tworzone za pomocą komend skryptowych IBM MQ (MQSC) lub komend IBM MQ Programmable Command Format (PCF). Gdy aplikacja tworzy obiekt połączenia, program IBM MQ classes for JMS wyszukuje w tabeli definicji kanału klienta odpowiednią definicję kanału połączenia klienta i używa definicji kanału do uruchomienia kanału

MQI. Więcej informacji na temat tabel definicji kanału klienta oraz sposobu ich tworzenia zawiera sekcja Tabela definicji kanału klienta.

Aby użyć tabeli definicji kanału klienta, właściwość CCDTURL obiektu ConnectionFactory musi być ustawiona na obiekt URL . IBM MQ classes for JMS nie należy odczytywać informacji o tabeli definicji kanału klienta z pliku konfiguracyjnego IBM MQ MQI client , chociaż z tego pliku są używane inne wartości (patrz sekcja “Plik konfiguracyjny IBM MQ classes for JMS/Jakarta Messaging” na stronie 102 , dla których wartości mają zastosowanie). Obiekt URL hermetyzuje adres URL (URL) identyfikujący nazwę i położenie pliku zawierającego tabelę definicji kanału klienta oraz określający sposób dostępu do pliku. Właściwość CCDTURL można ustawić za pomocą narzędzia administracyjnego IBM MQ JMS lub aplikacji, tworząc obiekt URL i wywołując metodę setCCDTURL() obiektu ConnectionFactory .

Na przykład, jeśli plik ccdt1.tab zawiera tabelę definicji kanału klienta i jest przechowywany w tym samym systemie, w którym działa aplikacja, aplikacja może ustawić właściwość CCDTURL w następujący sposób:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
factory.setCCDTURL(chanTab1);
```

Inny przykład: załóżmy, że plik ccdt2.tab zawiera tabelę definicji kanału klienta i jest przechowywany w systemie innym niż ten, w którym działa aplikacja. Jeśli dostęp do pliku można uzyskać za pomocą protokołu FTP, aplikacja może ustawić właściwość CCDTURL w następujący sposób:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
factory.setCCDTURL(chanTab2);
```

Oprócz ustawienia właściwości CCDTURL obiektu ConnectionFactory , właściwość QMANAGER tego samego obiektu musi być ustawiona na jedną z następujących wartości:

- Nazwa menedżera kolejek
- Gwiazdka (*), po której następuje nazwa grupy menedżerów kolejek

Są to te same wartości, które mogą być używane dla parametru **QMgrName** w wywołaniu MQCONN wywoływanym przez aplikację kliencką używającą interfejsu MQI (Message Queue Interface). Więcej informacji na temat znaczenia tych wartości zawiera sekcja MQCONN. Właściwość QMANAGER można ustawić za pomocą narzędzia administracyjnego IBM MQ JMS lub Eksploratora IBM MQ . Alternatywnie aplikacja może ustawić tę właściwość, wywołując metodę setQueueManager () obiektu ConnectionFactory .

Jeśli aplikacja utworzy następnie obiekt połączenia z obiektu ConnectionFactory , program IBM MQ classes for JMS uzyskuje dostęp do tabeli definicji kanału klienta identyfikowanej przez właściwość CCDTURL, używa właściwości QMANAGER do wyszukiwania w tabeli odpowiedniej definicji kanału połączenia klienckiego, a następnie używa definicji kanału do uruchamiania kanału MQI w menedżerze kolejek.

Należy zauważyć, że właściwości CCDTURL i CHANNEL obiektu ConnectionFactory nie mogą być ustawione jednocześnie, gdy aplikacja wywołuje metodę createConnection(). Jeśli obie właściwości są ustawione, metoda zgłasza wyjątek. Właściwość CCDTURL lub CHANNEL jest uważana za ustawioną, jeśli jej wartość jest inna niż NULL, pusty łańcuch lub łańcuch zawierający wszystkie znaki odstępu.

Gdy program IBM MQ classes for JMS znajdzie odpowiednią definicję kanału połączenia klienckiego w tabeli definicji kanału klienta, używa tylko informacji wyodrębnionych z tabeli do uruchomienia kanału MQI. Wszystkie właściwości powiązane z kanałem obiektu ConnectionFactory są ignorowane.

W szczególności należy zwrócić uwagę na następujące punkty, jeśli używany jest protokół TLS:

- Kanał MQI używa protokołu TLS tylko wtedy, gdy definicja kanału wyodrębniona z tabeli definicji kanału klienta określa nazwę CipherSpec obsługiwanej przez produkt IBM MQ classes for JMS.
- Tabela definicji kanału klienta zawiera również informacje o położeniu serwerów LDAP (Lightweight Directory Access Protocol), które przechowują listy odwołań certyfikatów (CRL). Program IBM MQ classes for JMS korzysta tylko z tych informacji w celu uzyskania dostępu do serwerów LDAP, które przechowują listy CRL.

- Tabela definicji kanału klienta może również zawierać położenie programu odpowiadającego OCSP. Produkt IBM MQ classes for JMS nie może używać informacji OCSP z pliku tabeli definicji kanału klienta. Można jednak skonfigurować protokół OCSP w sposób opisany w sekcji Online Certificate Status Protocol (OCSP) in Java and JMS client applications(Protokół OCSP).

Więcej informacji na temat używania protokołu TLS z tabelą definicji kanału klienta zawiera sekcja Używanie rozszerzonego klienta transakcyjnego z kanałami TLS.

Należy również zwrócić uwagę na następujące punkty, jeśli używane są wyjścia kanału:

- Kanał MQI używa tylko wyjść kanału i powiązanych danych użytkownika określonych w definicji kanału wyodrębnionej z tabeli definicji kanału klienta.
- Definicja kanału wyodrębniona z tabeli definicji kanału klienta może określać wyjścia kanału zapisane w pliku Java. Oznacza to na przykład, że parametr SCYEXIT komendy DEFINE CHANNEL w celu utworzenia definicji kanału połączenia klienckiego może określać nazwę klasy, która implementuje interfejs WMQSecurityExit . Podobnie parametr SENDEXIT może określać nazwę klasy implementującej interfejs WMQSendExit , a parametr RCVEXIT może określać nazwę klasy implementującej interfejs WMQReceiveExit . Więcej informacji na temat zapisywania wyjścia kanału w programie Javazawiera sekcja“Zapisywanie wyjść kanałów w produkcie Java for IBM MQ classes for JMS” na stronie 289.

  Obsługiwane jest również użycie wyjść kanałów napisanych w języku innym niż Java . Informacje na temat określania parametrów SCYEXIT, SENDEXIT i RCVEXIT w komendzie DEFINE CHANNEL dla wyjść kanałów napisanych w innym języku zawiera sekcja DEFINE CHANNEL.

*Automatyczne ponowne łączenie klienta JMS*
Skonfiguruj klienta JMS tak, aby automatycznie nawiązywał połączenie po awarii sieci, menedżera kolejek lub serwera.

Zwykle, jeśli autonomiczna aplikacja IBM MQ classes for JMS jest połączona z menedżerem kolejek przy użyciu transportu klienta i menedżer kolejek stanie się z jakiegoś powodu niedostępny (na przykład z powodu wyłączenia sieci, awarii menedżera kolejek lub zatrzymania menedżera kolejek), podczas następnej próby nawiązania komunikacji z menedżerem kolejek przez aplikację IBM MQ classes for JMS zgłosi wyjątek JMSException. Aplikacja musi wychwycić wyjątek JMSException i podjąć próbę ponownego nawiązania połączenia z menedżerem kolejek. Projekt aplikacji można uprościć, włączając automatyczne ponowne połączenie klienta. Gdy menedżer kolejek stanie się niedostępny, program IBM MQ classes for JMS próbuje automatycznie ponownie nawiązać połączenie z menedżerem kolejek w imieniu aplikacji. Oznacza to, że aplikacja nie musi zawierać logiki do ponownego nawiązania połączenia.

Użycie tej implementacji automatycznego ponownego połączenia klienta nie jest obsługiwane na serwerach aplikacji Java Platform, Enterprise Edition . Alternatywną implementację można znaleźć w sekcji “Używanie automatycznego ponownego połączenia klienta w środowiskach Java EE” na stronie 301 .

*Korzystanie z automatycznego ponownego połączenia klienta JMS*
Jeśli autonomiczna aplikacja IBM MQ classes for JMS używa fabryki połączeń, która ma ustawioną właściwość CONNECTIONNAMELIST lub CCDTURL, aplikacja może użyć automatycznego ponownego połączenia klienta.

Automatyczne ponowne połączenie klienta może być używane do ponownego nawiązywania połączeń z menedżerami kolejek, w tym z menedżerami, które są częścią konfiguracji wysokiej dostępności (HA). Konfiguracje wysokiej dostępności obejmują menedżery kolejek z wieloma instancjami, menedżery kolejek RDQM lub menedżery kolejek HA na urządzeniu IBM MQ .

Zachowanie funkcji automatycznego ponownego nawiązywania połączenia przez klienta, która jest udostępniana przez produkt IBM MQ classes for JMS , zależy od następujących właściwości:

**Właściwość TRANSPORT (nazwa skrócona TRAN) fabryki połączeń JMS**

Wartość TRANSPORT określa sposób, w jaki aplikacje używające fabryki połączeń łączą się z menedżerem kolejek. Ta właściwość musi być ustawiona na wartość CLIENT, aby można było używać automatycznego ponownego połączenia klienta. Automatyczne ponowne połączenie klienta nie jest dostępne dla aplikacji, które łączą się z menedżerem kolejek używającym fabryki połączeń z właściwością TRANSPORT ustawioną na wartość BIND, DIRECT lub DIRECTHTTP.

**Właściwość fabryki połączeń JMS QMANAGER (krótka nazwa QMGR)**

Właściwość QMANAGER określa nazwę menedżera kolejek, z którym łączy się fabryka połączeń.

**Właściwość fabryki połączeń JMS CONNECTIONNAMELIST (krótka nazwa CRHOSTS)**

Właściwość CONNECTIONNAMELIST jest listą rozdzielaną przecinkami, w której każda pozycja zawiera informacje na temat nazwy hosta i portu, które mają być używane do nawiązywania połączenia z menedżerem kolejek określonym przez właściwość QMANAGER w przypadku używania transportu CLIENT. Lista ma następujący format: nazwa hosta (port), nazwa hosta (port).

**Właściwość fabryki połączeń JMS CCDTURL (krótka nazwa CCDT)**

Właściwość CCDTURL wskazuje tabelę definicji kanału klienta używaną przez program IBM MQ classes for JMS podczas nawiązywania połączenia z menedżerem kolejek przy użyciu tabeli definicji kanału klienta.

**Właściwość fabryki połączeń JMS CLIENTRECONNECTOPTIONS (krótka nazwa CROPT)**

Parametr CLIENTRECONNECTOPTIONS określa, czy program IBM MQ classes for JMS będzie próbował automatycznie nawiązać połączenie z menedżerem kolejek w imieniu aplikacji, jeśli menedżer kolejek stanie się dostępny.

**Atrybut DefRecon w sekcji Kanały pliku konfiguracyjnego klienta**

Atrybut DefRecon udostępnia opcję administracyjną, która umożliwia automatyczne ponowne nawiązywanie połączeń przez wszystkie aplikacje lub wyłączenie automatycznego ponownego nawiązywania połączeń przez aplikacje, które zostały napisane w celu automatycznego ponownego nawiązywania połączeń.

Automatyczne ponowne nawiązywanie połączenia z klientem jest dostępne tylko wtedy, gdy aplikacja pomyślnie nawiąże połączenie z menedżerem kolejek.

Gdy aplikacja łączy się z menedżerem kolejek, który używa transportu CLIENT, IBM MQ classes for JMS używa wartości właściwości fabryki połączeń CLIENTRECONNECTOPTIONS, aby określić, czy ma być używane automatyczne ponowne połączenie klienta, jeśli menedżer kolejek, z którym aplikacja jest połączona, jest niedostępny. Tabela 1 przedstawia możliwe wartości właściwości CLIENTRECONNECTOPTIONS oraz zachowanie IBM MQ classes for JMS dla każdej z tych wartości:

*Tabela 44. Możliwe wartości właściwości CLIENTRECCECTOPTIONS.*

| **CLIENTRECONNECTOPTIONS** | **Zachowanie IBM MQ classes for JMS** |
|---|---|
| ANY | Jeśli parametr CONNECTIONNAMELIST jest ustawiony, należy użyć wartości właściwości CONNECTIONNAMELIST, aby otworzyć połączenie z kombinacją nazwy hosta i portu oraz nawiązać połączenie z dowolnym menedżerem kolejek. Aby użyć tej opcji automatycznego ponownego połączenia klienta, właściwość QMANAGER musi być ustawiona na wartość domyślną lub "*".<br>Jeśli parametr CCDTURL jest ustawiony, otwórz tabelę definicji kanału klienta określoną przez właściwość CCDTURL, wybierz pozycję w tabeli, a następnie użyj tej pozycji, aby uruchomić kanał połączenia klienta z menedżerem kolejek. Aby użyć tej opcji automatycznego ponownego połączenia klienta, właściwość QMANAGER musi być ustawiona na jedną z następujących wartości:<br>• Gwiazdka (*)<br>• Gwiazdka (*), po której następuje nazwa grupy menedżerów kolejek<br>• Pusty łańcuch lub łańcuch, który zawiera wszystkie znaki odstępu |
| ASDEF | Użyj wartości DefRecon , aby określić, czy jest dostępne automatyczne ponowne połączenie klienta. |
| WYŁĄCZONE | Nie wykonuj żadnego automatycznego ponownego połączenia klienta i zwróć do aplikacji wyjątek JMSException. |
| QMGR | Określa, że klient musi ponownie nawiązać połączenie z tym samym menedżerem kolejek. Ta opcja musi być używana w rozwiązaniach wysokiej dostępności, w których wymagane jest ponowne nawiązanie połączenia z inną instancją tego samego menedżera kolejek.<br>Jeśli parametr CONNECTIONNAMELIST jest ustawiony, należy użyć wartości właściwości CONNECTIONNAMELIST, aby otworzyć połączenie z kombinacją nazwy hosta i portu oraz nawiązać połączenie z menedżerem kolejek określonym przez właściwość QMANAGER.<br>Jeśli parametr CCDTURL jest ustawiony, otwórz tabelę definicji kanału klienta określoną przez właściwość CCDTURL, znajdź w tabeli pozycje zgodne z nazwą menedżera kolejek określoną przez właściwość QMANAGER, a następnie użyj tych pozycji, aby uruchomić kanał połączenia klienta z tym menedżerem kolejek. |

Jeśli parametr CONNECTIONNAMELIST jest ustawiony, podczas wykonywania automatycznego ponownego nawiązywania połączenia przez klient IBM MQ classes for JMS używa informacji z właściwości

CONNNECTIONNAMELIST fabryki połączeń, aby określić, z którym systemem ma zostać nawiązane ponowne połączenie.

Program IBM MQ classes for JMS początkowo próbuje ponownie nawiązać połączenie, używając nazwy hosta i portu określonych w pierwszej pozycji listy nazw połączeń (CONNECTIONNAMELIST). W przypadku nawiązania połączenia program IBM MQ classes for JMS podejmuje próbę nawiązania połączenia z menedżerem kolejek o nazwie określonej we właściwości QMANAGER. Jeśli można nawiązać połączenie z menedżerem kolejek, program IBM MQ classes for JMS ponownie otwiera wszystkie obiekty IBM MQ , które zostały otwarte przez aplikację przed automatycznym ponownym nawiązaniem połączenia przez klienta, i kontynuuje działanie tak jak poprzednio.

Jeśli nie można nawiązać połączenia z wymaganym menedżerem kolejek przy użyciu pierwszej pozycji na liście CONNECTIONNAMELIST, program IBM MQ classes for JMS próbuje wykonać drugą pozycję na liście CONNECTIONNAMELIST itd.

Po wypróbowaniu przez IBM MQ classes for JMS wszystkich pozycji na liście CONNECTIONNAMELIST, przed ponowną próbą nawiązania połączenia oczekują przez pewien czas. Aby wykonać nową próbę ponownego połączenia, IBM MQ classes for JMS rozpoczyna się od pierwszej pozycji na liście CONNECTIONNAMELIST. Następnie po kolei wypróbowują każdy wpis na liście CONNECTIONNAMELIST, dopóki nie nastąpi ponowne połączenie lub nie zostanie osiągnięty koniec listy CONNECTIONNAMELIST. W tym momencie program IBM MQ classes for JMS czeka przez pewien czas, zanim podejmie ponowną próbę.

Jeśli parametr CCDTURL jest ustawiony, podczas wykonywania automatycznego ponownego połączenia klienta IBM MQ classes for JMS używa tabeli definicji kanału klienta określonej we właściwości CCDTURL do określenia systemu, z którym ma zostać nawiązane ponowne połączenie.

Program IBM MQ classes for JMS początkowo analizuje tabelę definicji kanału klienta i znajduje odpowiednią pozycję, która jest zgodna z wartością właściwości QMANAGER. Po znalezieniu pozycji program IBM MQ classes for JMS próbuje ponownie nawiązać połączenie z wymaganym menedżerem kolejek przy użyciu tej pozycji. Jeśli można nawiązać połączenie z menedżerem kolejek, program IBM MQ classes for JMS ponownie otwiera wszystkie obiekty IBM MQ , które zostały otwarte przez aplikację przed automatycznym ponownym nawiązaniem połączenia przez klienta, i kontynuuje działanie tak jak poprzednio.

Jeśli nie można nawiązać połączenia z wymaganym menedżerem kolejek, program IBM MQ classes for JMS szuka innej odpowiedniej pozycji w tabeli definicji kanału klienta i próbuje jej użyć itd.

Gdy serwer IBM MQ classes for JMS próbował wszystkich odpowiednich pozycji w tabeli definicji kanału klienta, oczekiwał przez pewien czas przed ponowną próbą nawiązania połączenia. Aby wykonać nową próbę ponownego połączenia, program IBM MQ classes for JMS ponownie analizuje tabelę definicji kanału klienta i próbuje wykonać pierwszą odpowiednią pozycję. Następnie po kolei będą próbowane wszystkie odpowiednie pozycje w tabeli definicji kanału klienta, aż do momentu ponownego nawiązania połączenia lub wypróbowania ostatniej odpowiedniej pozycji w tabeli definicji kanału klienta. W tym momencie program IBM MQ classes for JMS czeka przez pewien czas przed ponowieniem próby.

Niezależnie od tego, czy używana jest lista CONNECTIONNAMELIST, czy CCDTURL, proces automatycznego ponownego nawiązywania połączenia przez klient jest kontynuowany do momentu pomyślnego ponownego nawiązania przez program IBM MQ classes for JMS połączenia z menedżerem kolejek określonym we właściwości QMANAGER.

Domyślnie próby ponownego połączenia są wykonywane w następujących odstępach czasu:

- Pierwsza próba jest podejmowana po początkowym opóźnieniu o 1 sekundę, plus element losowy o wartości do 250 milisekund.
- Druga próba jest wykonywana przez 2 sekundy, plus losowy odstęp czasu wynoszący maksymalnie 500 milisekund, po niepowodzeniu pierwszej próby.
- Trzecia próba jest wykonywana przez 4 sekundy, plus losowy odstęp czasu wynoszący maksymalnie 1 sekundę, po niepowodzeniu drugiej próby.
- Czwarta próba jest wykonywana przez 8 sekund, plus losowy odstęp czasu wynoszący maksymalnie 2 sekundy, po trzeciej próbie zakończonej niepowodzeniem.

- Piąta próba jest wykonywana 16 sekund, plus losowy przedział czasu do 4 sekund, po czwartej próbie zakończonej niepowodzeniem.
- Szósta próba i wszystkie kolejne próby są wykonywane po 25 sekundach, plus losowy odstęp czasu wynoszący maksymalnie 6 sekund i 250 milisekund po niepowodzeniu poprzedniej próby.

Próby ponownego połączenia są opóźnione o okresy, które są częściowo stałe i częściowo przypadkowe. Ma to na celu zapobieżenie równoczesnemu ponownemu nawiązywaniu połączenia przez wszystkie aplikacje produktu IBM MQ classes for JMS połączone z menedżerem kolejek, który nie jest już dostępny.

Jeśli konieczne jest zwiększenie wartości domyślnych, aby dokładniej odzwierciedlić czas wymagany do odtworzenia menedżera kolejek lub do aktywowania rezerwowego menedżera kolejek, należy zmodyfikować atrybut ReconDelay w sekcji Channel pliku konfiguracyjnego klienta. Więcej informacji na ten temat zawiera sekcja CHANNELS stanza of the client configuration file(Sekcja CHANNELS pliku konfiguracyjnego klienta).

To, czy aplikacja IBM MQ classes for JMS będzie działać poprawnie po ponownym połączeniu, zależy od jej projektu. Zapoznaj się z tematami pokrewnymi, aby dowiedzieć się, jak zaprojektować aplikacje, które mogą korzystać z funkcji automatycznego ponownego połączenia.

*Kody przyczyny wskazujące, że menedżer kolejek nie jest już dostępny*
Jakie kody przyczyny wskazują, że menedżer kolejek nie jest już dostępny lub jest nieosiągalny podczas próby automatycznego ponownego połączenia z bazą danych IBM MQ classes for JMS .

Tabela “Automatyczne ponowne łączenie klienta JMS” na stronie 294 zawiera przegląd wyjątków JMSException i sposobu automatycznego restartowania aplikacji, a informacje w sekcji “Korzystanie z automatycznego ponownego połączenia klienta JMS” na stronie 294 zawierają szczegółowe informacje o wymaganiach dotyczących automatycznego ponownego nawiązywania połączenia przez klient.

Poniższe informacje zawierają listę kodów przyczyny systemu IBM MQ , które aplikacja powinna sprawdzić:

**RC2009**
ZERWANE POŁĄCZENIE MQRC_CONNECTION_BROKEN

**RC2059**
MQRC_Q_MGR_NOT_AVAILABLE

**RC2161**
MQRC_Q_MGR_QUIESCING,

**RC2162**
MQRC_Q_MGR_ZATRZYMYWANIE

**RC2202**
MQRC_CONNECTION_QUIESCING,

**RC2203**
ZATRZYMANE_POŁĄCZENIA_MQRC

**RC2223**
MQRC_Q_MGR_NOT_ACTIVE (nieaktywna)

**RC2279**
MQRC_CHANNEL_STOPPED_BY_USER

**RC2537**
MQRC_CHANNEL_NOT_AVAILABLE

**RC2538**
MQRC_HOST_NOT_AVAILABLE (niedostępny)

Większość wyjątków JMSException zgłaszanych z powrotem do aplikacji korporacyjnych zawiera powiązany wyjątek MQException, który zawiera kod przyczyny. Aby zaimplementować logikę ponawiania dla kodów przyczyny z poprzedniej listy, aplikacje korporacyjne powinny sprawdzić ten powiązany wyjątek przy użyciu kodu podobnego do poniższego przykładu:

```
} catch (JMSException 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
            }
        }
    }
```

**Pojęcia pokrewne**

Klasy produktu IBM MQ dla usługi JMS

*Używanie automatycznego ponownego połączenia klienta w środowiskach Java SE i Java EE*

W środowisku Java SE i Java EE można wykorzystać automatyczne ponowne połączenie klienta IBM MQ , aby ułatwić korzystanie z różnych rozwiązań wysokiej dostępności (HA) i odtwarzania po awarii (DR).

Różne rozwiązania HA i DR są dostępne na różnych platformach:

- Multi Menedżery kolejek z wieloma instancjami są instancjami tego samego menedżera kolejek skonfigurowanego na różnych serwerach (patrz sekcja Menedżery kolejek z wieloma instancjami). Jedna instancja menedżera kolejek jest zdefiniowana jako instancja aktywna, a inna jako instancja rezerwowa. Jeśli aktywna instancja nie powiedzie się, menedżer kolejek z wieloma instancjami zostanie automatycznie zrestartowany na serwerze rezerwowym.

  Zarówno aktywny, jak i rezerwowy menedżer kolejek mają ten sam identyfikator menedżera kolejek (QMID). Aplikacje klienckie IBM MQ , które nawiązują połączenie z menedżerem kolejek z wieloma instancjami, można skonfigurować w taki sposób, aby automatycznie ponownie nawiązywał połączenie z instancją rezerwową menedżera kolejek przy użyciu automatycznego ponownego połączenia klienta.
- Linux RDQM (menedżer kolejek replikowanych danych) jest rozwiązaniem wysokiej dostępności dostępnym na platformach Linux (patrz RDQM o wysokiej dostępności). Konfiguracja RDQM składa się z trzech serwerów skonfigurowanych w grupie wysokiej dostępności (HA), z których każdy ma instancję menedżera kolejek. Jedna instancja jest działającym menedżerem kolejek, który synchronicznie replikuje swoje dane do pozostałych dwóch instancji. Jeśli działanie serwera, na którym działa ten menedżer kolejek, zakończy się niepowodzeniem, zostanie uruchomiona inna instancja menedżera kolejek, która będzie miała bieżące dane do działania. Trzy instancje menedżera kolejek współużytkują zmienny adres IP, dlatego klienty muszą być skonfigurowane tylko z jednym adresem IP. Aplikacje klienckie nawiązujące połączenie z menedżerem kolejek RDQM można skonfigurować w taki sposób, aby automatycznie ponownie nawiązywał połączenie z instancją rezerwową menedżera kolejek przy użyciu automatycznego ponownego połączenia klienta.
- MQ Appliance Rozwiązanie wysokiej dostępności może być również dostarczane przez parę urządzeń IBM MQ (patrz sekcja High Availability i Disaster Recovery w dokumentacji IBM MQ Appliance ). Menedżer kolejek wysokiej dostępności działa na jednym z urządzeń, jednocześnie synchronicznie replikując dane do instancji rezerwowej menedżera kolejek na innym urządzeniu. Jeśli działanie urządzenia podstawowego nie powiedzie się, menedżer kolejek zostanie automatycznie uruchomiony i uruchomiony na innym urządzeniu. Dwie instancje menedżera kolejek można skonfigurować w taki sposób, aby współużytkowała zmienny adres IP, dlatego klienty muszą być skonfigurowane tylko z pojedynczym adresem IP. Aplikacje klienckie nawiązujące połączenie z menedżerem kolejek wysokiej dostępności na urządzeniu IBM MQ Appliance mogą zostać skonfigurowane w taki sposób, aby automatycznie ponownie nawiązywać połączenie z instancją rezerwową menedżera kolejek przy użyciu automatycznego ponownego połączenia klienta.

**Uwaga:** W środowiskach Java EE , takich jak WebSphere Application Server, automatyczne ponowne łączenie klienta ze specyfikacjami aktywowania przy użyciu funkcji udostępnianych przez produkt IBM MQ classes for JMS nie jest obsługiwane. Adapter zasobów IBM MQ udostępnia własny mechanizm ponownego nawiązywania połączeń ze specyfikacjami aktywowania, jeśli menedżer kolejek, z którym łączyła się specyfikacja aktywowania, stanie się niedostępny. Aby uzyskać więcej informacji, zapoznaj się z sekcją: “Obsługa automatycznego ponownego połączenia klienta w środowiskach Java EE” na stronie 301.

**Pojęcia pokrewne**

Menedżery kolejek z wieloma instancjami

Automatyczne ponowne łączenie klienta

**Odsyłacze pokrewne**

Wysoka dostępność rdqm

*Używanie automatycznego ponownego połączenia klienta w środowiskach Java SE*

Aplikacje korzystające z produktu IBM MQ classes for JMS działającego w środowiskach Java SE mogą korzystać z funkcji automatycznego ponownego nawiązywania połączeń przez klienta za pośrednictwem właściwości fabryki połączeń **CLIENTRECONNECTOPTIONS**.

Właściwość fabryki połączeń **CLIENTRECONNECTOPTIONS** używa dwóch dodatkowych właściwości fabryki połączeń, **CONNECTIONNAMELIST** i **CCDTURL**, do określenia sposobu nawiązywania połączenia z serwerem, na którym działa menedżer kolejek.

## CONNECTIONNAMELIST właściwość

Właściwość **CONNECTIONNAMELIST** jest listą rozdzielaną przecinkami, która zawiera informacje o nazwie hosta i porcie, które mają być używane do nawiązywania połączenia z menedżerem kolejek w trybie klienta. Ta właściwość jest używana z wartościami **QMANAGER** i **CHANNEL** . Gdy aplikacja używa właściwości **CONNECTIONNAMELIST** do utworzenia połączenia klienta, serwer IBM MQ classes for JMS próbuje połączyć się z każdym hostem w kolejności listy. Jeśli pierwszy host menedżera kolejek jest niedostępny, program IBM MQ classes for JMS próbuje nawiązać połączenie z następnym hostem na liście. Jeśli koniec listy nazw połączeń zostanie osiągnięty bez tworzenia połączenia, IBM MQ classes for JMS zgłasza kod przyczyny `MQRC_QMGR_NOT_AVAILABLE` IBM MQ .

Jeśli działanie menedżera kolejek, z którym połączona jest aplikacja, zakończy się niepowodzeniem, wszystkie aplikacje, które używały programu **CONNECTIONNAMELIST** do nawiązania połączenia z tym menedżerem kolejek, otrzymają wyjątek wskazujący, że menedżer kolejek jest niedostępny. Aplikacja musi wychwycić wyjątek i usunąć wszystkie używane przez nią zasoby. Aby utworzyć połączenie, aplikacja musi używać fabryki połączeń. Fabryka połączeń próbuje ponownie nawiązać połączenie z każdym hostem w kolejności wyświetlania. Menedżer kolejek, który nie powiódł się, jest teraz niedostępny. Fabryka połączeń próbuje połączyć się z innym hostem na liście.

## CCDTURL właściwość

Właściwość **CCDTURL** zawiera adres URL (URL) wskazujący na tabelę definicji kanału klienta (CCDT). Ta właściwość jest używana z właściwością **QMANAGER** . Tabela CCDT zawiera listę kanałów klienta używanych do nawiązywania połączenia z menedżerem kolejek zdefiniowanym w systemie IBM MQ . Informacje na temat sposobu używania CDT przez IBM MQ classes for JMSzawiera sekcja “Używanie tabeli definicji kanału klienta z produktem IBM MQ classes for JMS” na stronie 292.

## Korzystanie z właściwości CLIENTRECONNECTOPTIONS w celu włączenia automatycznego ponownego nawiązywania połączenia przez klienta w obrębie IBM MQ classes for JMS

Właściwość **CLIENTRECONNECTOPTIONS** jest używana do włączenia automatycznego ponownego nawiązywania połączenia przez klient w obrębie IBM MQ classes for JMS. Możliwe wartości tej właściwości są następujące:

**ASDEF (ASDEF)**
: Zachowanie automatycznego ponownego połączenia klienta jest definiowane przez wartość domyślną, która jest określona w sekcji kanału pliku konfiguracyjnego klienta IBM MQ (`mqclient.ini`).

**WYŁĄCZONE**
: Automatyczne ponowne nawiązywanie połączenia z klientem jest wyłączone.

**QMGR**
Program IBM MQ classes for JMS podjął próbę nawiązania połączenia z menedżerem kolejek o tym samym identyfikatorze, co menedżer kolejek, z którym był połączony, przy użyciu jednej z następujących opcji:

- Właściwość **CONNECTIONNAMELIST** i kanał zdefiniowany we właściwości **CHANNEL** .
- Tabela CCDT zdefiniowana we właściwości **CCDTURL** .

**ANY**
Program IBM MQ classes for JMS próbuje ponownie nawiązać połączenie z menedżerem kolejek o tej samej nazwie, używając właściwości **CONNECTIONNAMELIST** lub **CCDTURL**.

**Informacje pokrewne**
Sekcja CHANNELS pliku konfiguracyjnego klienta

*Używanie automatycznego ponownego połączenia klienta w środowiskach Java EE*
Adapter zasobów IBM MQ , który można wdrożyć w środowiskach Java EE (Java Platform, Enterprise Edition), oraz dostawca przesyłania komunikatów produktu WebSphere Application Server IBM MQ używają pliku IBM MQ classes for JMS do komunikowania się z menedżerami kolejek systemu IBM MQ . Adapter zasobów IBM MQ i dostawca przesyłania komunikatów produktu WebSphere Application Server IBM MQ udostępniają wiele mechanizmów umożliwiających automatyczne ponowne nawiązanie połączenia z menedżerem kolejek przez specyfikacje aktywowania, porty nasłuchiwania produktu WebSphere Application Server i aplikacje działające w obrębie kontenerów klienta. Komponenty EJB (Enterprise JavaBeans ) i aplikacje z interfejsem WWW muszą implementować własną logikę ponownego połączenia.

**Uwaga:** Automatyczne ponowne łączenie klienta ze specyfikacjami aktywowania przy użyciu funkcji udostępnianych przez IBM MQ classes for JMS nie jest obsługiwane (patrz sekcja “Automatyczne ponowne łączenie klienta JMS” na stronie 294). Adapter zasobów IBM MQ udostępnia własny mechanizm ponownego nawiązywania połączeń ze specyfikacjami aktywowania, jeśli menedżer kolejek, z którym łączyła się specyfikacja aktywowania, stanie się niedostępny.

Mechanizm udostępniany przez adapter zasobów jest kontrolowany przez:

- IBM MQ Właściwość adaptera zasobów **reconnectionRetryCount**.
- IBM MQ Właściwość adaptera zasobów **reconnectionRetryInterval**.
- Właściwość specyfikacji aktywowania **connectionNameList**.

Więcej informacji na temat tych właściwości zawiera sekcja “Konfiguracja właściwości obiektu ResourceAdapter” na stronie 465.

Użycie automatycznego ponownego połączenia klienta w metodzie `onMessage()` aplikacji komponentu bean sterowanego komunikatami lub w dowolnej innej aplikacji działającej w środowisku Java Platform, Enterprise Edition nie jest obsługiwane. Aplikacja musi zaimplementować własną logikę ponownego połączenia, jeśli menedżer kolejek, z którym nawiązała połączenie, stanie się niedostępny. Więcej informacji na ten tematzawiera sekcja “Implementowanie logiki ponownego połączenia w aplikacji Java EE” na stronie 309.

*Obsługa automatycznego ponownego połączenia klienta w środowiskach Java EE*
W środowiskach Java EE , takich jak WebSphere Application Server, adapter zasobów IBM MQ i dostawca przesyłania komunikatów produktu WebSphere Application Server IBM MQ , udostępnia wiele mechanizmów, które umożliwiają aplikacjom automatyczne ponowne nawiązywanie połączeń z menedżerem kolejek. Jednak w niektórych przypadkach do tego wsparcia mają zastosowanie ograniczenia.

Adapter zasobów IBM MQ , który można wdrożyć w środowiskach Java EE i dostawcy przesyłania komunikatów produktu WebSphere Application Server IBM MQ , używa pliku IBM MQ classes for JMS do komunikowania się z menedżerami kolejek systemu IBM MQ .

Poniższa tabela zawiera podsumowanie informacji o obsłudze automatycznego ponownego nawiązywania połączenia przez adapter zasobów IBM MQ i dostawcę przesyłania komunikatów produktu WebSphere Application Server IBM MQ .

*Tabela 45. Podsumowanie obsługi opcji automatycznego ponownego połączenia klienta w środowiskach Java EE*

| **Opcje automatycznego ponownego połączenia** | **Właściwość CONNECTIONNAMELIST** | **CCDTURL, właściwość** | **Właściwość CLIENTRECONNECTOPTIONS** | **Alternatywne podejście do automatycznego ponownego połączenia klienta** |
|---|---|---|---|---|
| Specyfikacje aktywowania | Obsługiwane z ograniczeniami | Obsługiwane z ograniczeniami | Nieobsługiwane | Środowisko Java EE i specyfikacje aktywowania udostępniają własny mechanizm ponownego połączenia |
| Porty nasłuchiwania WebSphere Application Server | Obsługiwane z ograniczeniami | Obsługiwane z ograniczeniami | Nieobsługiwane | WebSphere Application Server udostępnia własny mechanizm ponownego połączenia |
| Komponenty EJB (Enterprise JavaBeans ) i aplikacje WWW | Obsługiwane z ograniczeniami | Obsługiwane z ograniczeniami | Nieobsługiwane | Aplikacja musi implementować własną logikę ponownego połączenia |
| Aplikacje działające wewnątrz kontenerów klienckich | Obsługiwany | Obsługiwany | Obsługiwany | Nie dotyczy |

Aplikacje bean sterowane komunikatami, które są zainstalowane w środowisku Java EE , na przykład IBM MQ classes for JMS, mogą używać specyfikacji aktywowania do przetwarzania komunikatów w systemie IBM MQ . Specyfikacje aktywowania są używane do wykrywania komunikatów przychodzących do systemu IBM MQ i dostarczania ich do komponentów bean sterowanych komunikatami w celu przetwarzania. Komponenty bean sterowane komunikatami mogą również nawiązywać więcej połączeń z systemami IBM MQ z poziomu metody **onMessage()** . Więcej informacji na temat sposobu, w jaki te połączenia mogą korzystać z automatycznego ponownego połączenia klienta, zawiera sekcja Komponenty EJB (Enterprise JavaBeans ) i aplikacje z interfejsem WWW.

*Specyfikacje aktywowania*
W przypadku specyfikacji aktywowania właściwości **CONNECTIONNAMELIST** i **CCDTURL** są obsługiwane z ograniczeniami, a właściwość **CLIENTRECONNECTOPTIONS** nie jest obsługiwana.

Aplikacje komponentu bean sterowanego komunikatami (MDB) zainstalowane w środowisku Java EE , takim jak WebSphere Application Server, mogą używać specyfikacji aktywowania do przetwarzania komunikatów w systemie IBM MQ .

Specyfikacje aktywowania są używane do wykrywania komunikatów przychodzących do systemu IBM MQ , a następnie dostarczania ich do komponentów MDB w celu przetworzenia. W tej sekcji opisano sposób monitorowania systemu IBM MQ przez specyfikację aktywowania.

Bazy danych MDB mogą również nawiązywać dodatkowe połączenia z systemami IBM MQ z poziomu swojej metody `onMessage()` .

Szczegółowe informacje o tym, w jaki sposób te połączenia mogą korzystać z automatycznego ponownego połączenia klienta, można znaleźć w sekcji “Komponenty EJB (Enterprise JavaBeans ) i aplikacje WWW” na stronie 307.

## CONNECTIONNAMELIST właściwość

Podczas uruchamiania specyfikacja aktywowania podejmuje próbę nawiązania połączenia z menedżerem kolejek przy użyciu następujących elementów:

- Jeden określony we właściwości **QMANAGER**
- Kanał wymieniony we właściwości **CHANNEL**
- Informacje o nazwie hosta i porcie z pierwszego wpisu w pliku **CONNECTIONNAMELIST**

Jeśli specyfikacja aktywowania nie może nawiązać połączenia z menedżerem kolejek przy użyciu pierwszej pozycji na liście, specyfikacja aktywowania jest przenoszona do drugiej pozycji itd. do momentu nawiązania połączenia z menedżerem kolejek lub osiągnięcia końca listy.

Jeśli specyfikacja aktywowania nie może nawiązać połączenia z określonym menedżerem kolejek przy użyciu żadnej z pozycji w pliku **CONNECTIONNAMELIST**, specyfikacja aktywowania zostaje zatrzymana i musi zostać zrestartowana.

Po uruchomieniu specyfikacji aktywowania specyfikacja aktywowania pobiera komunikaty z systemu IBM MQ i dostarcza je do komponentu MDB w celu przetworzenia.

Jeśli działanie menedżera kolejek zakończy się niepowodzeniem podczas przetwarzania komunikatu, środowisko Java EE wykryje niepowodzenie i podejmie próbę ponownego nawiązania połączenia ze specyfikacją aktywowania.

Specyfikacja aktywowania używa informacji z właściwości **CONNECTIONNAMELIST** tak jak wcześniej, gdy specyfikacja aktywowania wykonuje próby ponownego połączenia.

Jeśli specyfikacja aktywowania próbuje wszystkich pozycji w pliku **CONNECTIONNAMELIST** i nadal nie może nawiązać połączenia z menedżerem kolejek, to przed ponowieniem próby specyfikacja aktywowania czeka przez czas określony przez IBM MQ właściwość adaptera zasobów **reconnectionRetryInterval** .

IBM MQ Właściwość adaptera zasobów **reconnectionRetryCount** definiuje liczbę kolejnych prób ponownego nawiązania połączenia, które są wykonywane przed zatrzymaniem specyfikacji aktywowania i wymaga ręcznego zrestartowania.

Po ponownym połączeniu specyfikacji aktywowania z systemem IBM MQ środowisko Java EE wykonuje wszystkie wymagane procedury czyszczące wymagane dla transakcji, a następnie wznawia dostarczanie komunikatów do komponentów MDB w celu przetworzenia.

Aby procedura czyszcząca dla transakcji działała poprawnie, środowisko Java EE musi mieć dostęp do dzienników menedżera kolejek, który uległ awarii.

Jeśli specyfikacje aktywowania są używane z transakcyjnymi komponentami MDB, które uczestniczą w transakcjach XA i nawiązują połączenie z menedżerem kolejek z wieloma instancjami, parametr **CONNECTIONNAMELIST** musi zawierać wpis zarówno dla instancji aktywnego, jak i rezerwowego menedżera kolejek.

Oznacza to, że środowisko produktu Java EE może uzyskać dostęp do dzienników menedżera kolejek, jeśli środowisko wymaga wykonania odtwarzania transakcji, niezależnie od tego, z którym menedżerem kolejek środowisko ponownie nawiązuje połączenie po awarii.

Jeśli transakcyjne bazy danych MDB są używane z autonomicznymi menedżerami kolejek, właściwość **CONNECTIONNAMELIST** musi zawierać pojedynczy wpis, aby zapewnić, że specyfikacja aktywowania zawsze będzie ponownie nawiązywać połączenie z tym samym menedżerem kolejek działającym w tym samym systemie po awarii.

## CCDTURL właściwość

Podczas uruchamiania specyfikacja aktywowania próbuje nawiązać połączenie z menedżerem kolejek określonym we właściwości **QMANAGER** przy użyciu pierwszej pozycji w tabeli definicji kanału klienta (CCDT).

Jeśli specyfikacja aktywowania nie może nawiązać połączenia z menedżerem kolejek przy użyciu pierwszej pozycji w tabeli, specyfikacja aktywowania jest przenoszona do drugiej pozycji itd. do momentu nawiązania połączenia z menedżerem kolejek lub osiągnięcia końca tabeli.

Jeśli specyfikacja aktywowania nie może nawiązać połączenia z określonym menedżerem kolejek przy użyciu żadnej z pozycji w tabeli definicji kanału klienta, specyfikacja aktywowania zostaje zatrzymana i musi zostać zrestartowana.

Po uruchomieniu specyfikacji aktywowania specyfikacja aktywowania pobiera komunikaty z systemu IBM MQ i dostarcza je do komponentu MDB w celu przetworzenia.

Jeśli działanie menedżera kolejek zakończy się niepowodzeniem podczas przetwarzania komunikatu, środowisko Java EE wykryje niepowodzenie i podejmie próbę ponownego nawiązania połączenia ze specyfikacją aktywowania.

Specyfikacja aktywowania korzysta z informacji we właściwości CCDT, tak jak wcześniej, podczas próby ponownego nawiązania połączenia przez specyfikację aktywowania.

Jeśli specyfikacja aktywowania próbuje wszystkich pozycji w tabeli definicji kanału klienta i nadal nie może nawiązać połączenia z menedżerem kolejek, przed podjęciem kolejnej próby specyfikacja aktywowania oczekuje przez czas określony przez IBM MQ właściwość adaptera zasobów **reconnectionRetryInterval** .

IBM MQ Właściwość adaptera zasobów **reconnectionRetryCount** definiuje liczbę kolejnych prób ponownego nawiązania połączenia, które są wykonywane przed zatrzymaniem specyfikacji aktywowania i wymaga ręcznego zrestartowania.

Po ponownym połączeniu specyfikacji aktywowania z systemem IBM MQ środowisko Java EE wykonuje wszystkie wymagane procedury czyszczące wymagane dla transakcji, a następnie wznawia dostarczanie komunikatów do komponentów MDB w celu przetworzenia.

Aby procedura czyszcząca dla transakcji działała poprawnie, środowisko Java EE musi mieć dostęp do dzienników menedżera kolejek, który uległ awarii.

Jeśli specyfikacje aktywowania są używane z transakcyjnymi komponentami MDB uczestniczącymi w transakcjach XA i nawiązują połączenie z menedżerem kolejek z wieloma instancjami, tabela CCDT musi zawierać pozycję zarówno dla aktywnej, jak i rezerwowej instancji menedżera kolejek.

Oznacza to, że środowisko produktu Java EE może uzyskać dostęp do dzienników menedżera kolejek, jeśli środowisko wymaga wykonania odtwarzania transakcji, niezależnie od tego, z którym menedżerem kolejek środowisko ponownie nawiązuje połączenie po awarii.

Jeśli transakcyjne bazy danych MDB są używane z autonomicznymi menedżerami kolejek, tabela CCDT musi zawierać pojedynczą pozycję, aby zapewnić, że specyfikacja aktywowania zawsze ponownie nawiązuje połączenie z tym samym menedżerem kolejek działającym w tym samym systemie po awarii.

Upewnij się, że została ustawiona wartość domyślna *PREFERRED* dla właściwości **AFFINITY** w CCDT, używana ze specyfikacjami aktywowania, tak aby połączenia były nawiązywane z tym samym aktywnym menedżerem kolejek.

## CLIENTRECONNECTOPTIONS właściwość

Specyfikacje aktywowania udostępniają własne funkcje ponownego połączenia. Udostępniona funkcja umożliwia specyfikacji automatyczne ponowne nawiązanie połączenia z systemem IBM MQ , jeśli menedżer kolejek, z którym nawiązano połączenie, zakończy się niepowodzeniem.

Z tego powodu funkcja automatycznego ponownego połączenia klienta udostępniana przez IBM MQ classes for JMS nie jest obsługiwana.

Dla wszystkich specyfikacji aktywowania używanych w Java EEnależy ustawić właściwość **CLIENTRECONNECTOPTIONS** na wartość *DISABLED* .

*Porty nasłuchiwania WebSphere Application Server*
Aplikacje komponentów bean sterowanych komunikatami (MDB) zainstalowane w produkcie WebSphere Application Server mogą również używać portów nasłuchiwania do przetwarzania komunikatów w systemie IBM MQ .

Porty nasłuchiwania są używane do wykrywania komunikatów przychodzących do systemu IBM MQ , a następnie dostarczania ich do komponentów MDB w celu przetworzenia. W tym temacie opisano sposób monitorowania systemu IBM MQ przez port nasłuchiwania.

Bazy danych MDB mogą również nawiązywać dodatkowe połączenia z systemami IBM MQ z poziomu swojej metody onMessage() .

Więcej informacji na temat sposobu, w jaki te połączenia mogą korzystać z automatycznego ponownego połączenia klienta, zawiera sekcja “Komponenty EJB (Enterprise JavaBeans ) i aplikacje WWW” na stronie 307 .

Dla portów nasłuchiwania systemu WebSphere Application Server :

- Systemy **CONNECTIONNAMELIST** i **CCDTURL** są obsługiwane z ograniczeniami
- **CLIENTRECONNECTOPTIONS** nie jest obsługiwane

## CONNECTIONNAMELIST właściwość

Porty nasłuchiwania korzystają z pul połączeń JMS podczas nawiązywania połączeń z produktem IBM MQ, dlatego mają wpływ na korzystanie z pul połączeń. Więcej informacji na ten temat zawiera sekcja “Specyfikacje aktywowania” na stronie 302.

Jeśli nie ma wolnych połączeń, a maksymalna liczba połączeń nie została jeszcze utworzona na podstawie tej fabryki połączeń, do utworzenia nowego połączenia z produktem IBM MQużywana jest właściwość **CONNECTIONNAMELIST** .

Jeśli wszystkie systemy IBM MQ w **CONNECTIONNAMELIST** nie są dostępne, port nasłuchiwania zostanie zatrzymany.

Następnie port nasłuchiwania czeka przez czas określony przez niestandardową właściwość usługi nasłuchiwania komunikatów **RECOVERY.RETRY.INTERVAL** i próbuje ponownie nawiązać połączenie.

Ta próba ponownego połączenia sprawdza, czy w puli połączeń istnieją wolne połączenia, na wypadek, gdyby między kolejnymi próbami połączenia zostały zwrócone takie połączenia. Jeśli jeden z nich nie jest dostępny, port nasłuchiwania używa pliku **CONNECTIONNAMELIST** w taki sam sposób, jak poprzednio.

Po ponownym połączeniu portu nasłuchiwania z systemem IBM MQ środowisko Java EE wykonuje wszystkie wymagane procedury czyszczące, a następnie wznawia dostarczanie komunikatów do komponentów MDB w celu przetworzenia.

Aby procedura czyszcząca dla transakcji działała poprawnie, środowisko Java EE musi mieć dostęp do dzienników menedżera kolejek, który uległ awarii.

Jeśli porty nasłuchiwania są używane z transakcyjnymi bazami danych, które uczestniczą w transakcjach XA i nawiązują połączenie z **menedżerem kolejek z wieloma instancjami**, właściwość **CONNECTIONNAMELIST** musi zawierać wpis zarówno dla instancji aktywnego, jak i rezerwowego menedżera kolejek.

Oznacza to, że środowisko produktu Java EE może uzyskać dostęp do dzienników menedżera kolejek, jeśli środowisko wymaga wykonania odtwarzania transakcji, niezależnie od tego, z którym menedżerem kolejek środowisko ponownie nawiązuje połączenie po awarii.

Jeśli transakcyjne bazy danych MDB są używane z autonomicznymi menedżerami kolejek, właściwość **CONNECTIONNAMELIST** musi zawierać pojedynczy wpis, aby zapewnić, że specyfikacja aktywowania zawsze będzie ponownie nawiązywać połączenie z tym samym menedżerem kolejek działającym w tym samym systemie po awarii.

## CCDTURL właściwość

Podczas uruchamiania port nasłuchiwania próbuje nawiązać połączenie z menedżerem kolejek określonym we właściwości **QMANAGER** przy użyciu pierwszej pozycji w tabeli definicji kanału klienta.

Jeśli port nasłuchiwania nie może nawiązać połączenia z menedżerem kolejek przy użyciu pierwszej pozycji w tabeli, port nasłuchiwania jest przenoszony do drugiej pozycji itd. do momentu nawiązania połączenia z menedżerem kolejek lub osiągnięcia końca tabeli.

Jeśli port nasłuchiwania nie może nawiązać połączenia z określonym menedżerem kolejek przy użyciu żadnej z pozycji w tabeli definicji kanału klienta, port nasłuchiwania zostanie zatrzymany.

Następnie port nasłuchiwania czeka przez czas określony przez niestandardową właściwość usługi nasłuchiwania komunikatów **RECOVERY.RETRY.INTERVAL** i próbuje ponownie nawiązać połączenie.

Ta próba ponownego połączenia odbywa się przez wszystkie pozycje w tabeli definicji kanału klienta (CCDT), jak wcześniej.

Po uruchomieniu port nasłuchiwania pobiera komunikaty z systemu IBM MQ i dostarcza je do komponentu MDB w celu przetworzenia.

Jeśli działanie menedżera kolejek zakończy się niepowodzeniem podczas przetwarzania komunikatu, środowisko Java EE wykryje niepowodzenie i podejmie próbę ponownego nawiązania połączenia z portem nasłuchiwania. Port nasłuchiwania korzysta z informacji zawartych w tabeli definicji kanału klienta podczas próby ponownego połączenia.

Jeśli port nasłuchiwania próbuje wszystkich wpisów w tabeli definicji kanału klienta i nadal nie może nawiązać połączenia z menedżerem kolejek, port oczekuje przez czas określony we właściwości **RECOVERY.RETRY.INTERVAL** przed ponowną próbą.

Właściwość usługi nasłuchiwania komunikatów **MAX.RECOVERY.RETRIES** definiuje liczbę kolejnych prób ponownego nawiązania połączenia, które są wykonywane przed zatrzymaniem portu nasłuchiwania i wymagają ręcznego zrestartowania.

Po ponownym połączeniu portu nasłuchiwania z systemem IBM MQ środowisko Java EE wykonuje wszystkie wymagane procedury czyszczące, a następnie wznawia dostarczanie komunikatów do komponentów MDB w celu przetworzenia.

Aby procedura czyszcząca dla transakcji działała poprawnie, środowisko Java EE musi mieć dostęp do dzienników menedżera kolejek, który uległ awarii.

Jeśli porty nasłuchiwania są używane z transakcyjnymi komponentami MDB uczestniczącymi w transakcjach XA i nawiązują połączenie z menedżerem kolejek z wieloma instancjami, tabela CCDT musi zawierać pozycję zarówno dla aktywnej, jak i rezerwowej instancji menedżera kolejek.

Oznacza to, że środowisko produktu Java EE może uzyskać dostęp do dzienników menedżera kolejek, jeśli środowisko wymaga wykonania odtwarzania transakcji, niezależnie od tego, z którym menedżerem kolejek środowisko ponownie nawiązuje połączenie po awarii.

Jeśli transakcyjne komponenty MDB są używane z autonomicznymi menedżerami kolejek, tabela CCDT musi zawierać pojedynczą pozycję, aby zapewnić, że port nasłuchiwania będzie zawsze ponownie nawiązywał połączenie z tym samym menedżerem kolejek działającym w tym samym systemie po awarii.

Upewnij się, że dla właściwości **AFFINITY** w tabeli CDT używanej z portami nasłuchiwania ustawiono wartość domyślną *PREFERRED* , tak aby połączenia były nawiązywane z tym samym aktywnym menedżerem kolejek.

## CLIENTRECONNECTOPTIONS właściwość

Porty nasłuchiwania udostępniają własną funkcję ponownego połączenia. Udostępniona funkcja umożliwia portom nasłuchiwania automatyczne ponowne nawiązanie połączenia z systemem IBM MQ , jeśli menedżer kolejek, z którym nawiązano połączenie, zakończy się niepowodzeniem.

Z tego powodu funkcja automatycznego ponownego połączenia klienta udostępniana przez IBM MQ classes for JMS nie jest obsługiwana.

Dla wszystkich portów nasłuchiwania używanych w produkcie Java EEnależy ustawić właściwość **CLIENTRECONNECTOPTIONS** na wartość *DISABLED* .

*Komponenty EJB (Enterprise JavaBeans ) i aplikacje WWW*
Aplikacje EJB (Enterprise JavaBean ) i aplikacje działające w kontenerze WWW, takie jak Serwlety, używają fabryki połączeń JMS do utworzenia połączenia z menedżerem kolejek produktu IBM MQ .

W przypadku komponentów EJB i aplikacji z interfejsem WWW obowiązują następujące ograniczenia:

- Systemy **CONNECTIONNAMELIST** i **CCDTURL** są obsługiwane z ograniczeniami
- **CLIENTRECONNECTOPTIONS** nie jest obsługiwane

## CONNECTIONNAMELIST właściwość

Jeśli środowisko Java EE udostępnia pulę połączeń dla połączeń JMS, należy zapoznać się z sekcją “Używanie listy nazw połączeń (CONNECTIONNAMELIST) lub tabeli definicji kanału klienta (CCDT) w puli połączeń” na stronie 308 , aby uzyskać informacje na temat wpływu na zachowanie właściwości **CONNECTIONNAMELIST** .

Jeśli środowisko Java EE nie udostępnia puli połączeń JMS . Aplikacja używa właściwości **CONNECTIONNAMELIST** w taki sam sposób, jak aplikacje Java SE .

Jeśli aplikacje są używane z transakcyjnymi komponentami MDB, które uczestniczą w transakcjach XA i nawiązują połączenie z menedżerem kolejek z wieloma instancjami, parametr **CONNECTIONNAMELIST** musi zawierać wpis zarówno dla instancji aktywnego, jak i rezerwowego menedżera kolejek.

Oznacza to, że środowisko produktu Java EE może uzyskać dostęp do dzienników menedżera kolejek, jeśli środowisko wymaga wykonania odtwarzania transakcji, niezależnie od tego, z którym menedżerem kolejek środowisko ponownie nawiązuje połączenie po awarii.

Jeśli aplikacje są używane z autonomicznymi menedżerami kolejek, właściwość **CONNECTIONNAMELIST** musi zawierać pojedynczą pozycję, aby zapewnić, że po awarii aplikacja będzie zawsze ponownie nawiązywać połączenie z tym samym menedżerem kolejek, działającym w tym samym systemie.

## CCDTURL właściwość

Jeśli środowisko Java EE udostępnia pulę połączeń dla połączeń JMS , należy zapoznać się z sekcją “Używanie listy nazw połączeń (CONNECTIONNAMELIST) lub tabeli definicji kanału klienta (CCDT) w puli połączeń” na stronie 308 , aby uzyskać informacje na temat wpływu na zachowanie właściwości **CCDTURL** .

Jeśli środowisko Java EE nie udostępnia puli połączeń JMS . Aplikacja używa właściwości **CCDTURL** w taki sam sposób, jak aplikacje Java SE .

Jeśli aplikacje są używane z transakcyjnymi komponentami MDB uczestniczącymi w transakcjach XA i nawiązują połączenie z menedżerem kolejek z wieloma instancjami, tabela CCDT musi zawierać pozycję zarówno dla aktywnej, jak i rezerwowej instancji menedżera kolejek.

Oznacza to, że środowisko produktu Java EE może uzyskać dostęp do dzienników menedżera kolejek, jeśli środowisko wymaga wykonania odtwarzania transakcji, niezależnie od tego, z którym menedżerem kolejek środowisko ponownie nawiązuje połączenie po awarii.

Jeśli aplikacje są używane z autonomicznymi menedżerami kolejek, tabela CCDT musi zawierać pojedynczą pozycję, aby zapewnić, że specyfikacja aktywowania będzie zawsze ponownie nawiązywać połączenie z tym samym menedżerem kolejek działającym w tym samym systemie po awarii.

## CLIENTRECONNECTOPTIONS właściwość

Dla wszystkich fabryk połączeń JMS używanych przez komponenty EJB lub aplikacje działające w kontenerze WWW należy ustawić właściwość **CLIENTRECONNECTOPTIONS** na wartość *DISABLED* .

Aplikacje, które wymagają automatycznego ponownego nawiązania połączenia z nowym menedżerem kolejek, jeśli używany przez nie menedżer kolejek zakończy się niepowodzeniem, muszą

zaimplementować własną logikę ponownego połączenia. Więcej informacji można znaleźć w sekcji “Implementowanie logiki ponownego połączenia w aplikacji Java EE” na stronie 309.

Scenariusze: WebSphere Application Server z IBM MQ

Scenariusze: WebSphere Application Server profil Liberty z serwerem IBM MQ

*Aplikacje działające wewnątrz kontenerów klienckich*
Niektóre środowiska Java EE , takie jak WebSphere Application Server, udostępniają kontener klienta, który może być używany do uruchamiania aplikacji Java SE .

Aplikacje działające w tych środowiskach używają fabryki połączeń JMS do nawiązania połączenia z menedżerem kolejek produktu IBM MQ .

W przypadku aplikacji działających wewnątrz kontenerów klienta:

- **CONNECTIONNAMELIST** i **CCDTURL** są w pełni obsługiwane
- Produkt **CLIENTRECONNECTOPTIONS** jest w pełni obsługiwany

## CONNECTIONNAMELIST właściwość

Jeśli środowisko Java EE udostępnia pulę połączeń dla połączeń JMS, należy zapoznać się z sekcją “Używanie listy nazw połączeń (CONNECTIONNAMELIST) lub tabeli definicji kanału klienta (CCDT) w puli połączeń” na stronie 308 , aby uzyskać informacje na temat wpływu na zachowanie właściwości **CONNECTIONNAMELIST** .

Jeśli środowisko Java EE nie udostępnia puli połączeń JMS . Aplikacja używa właściwości **CONNECTIONNAMELIST** w taki sam sposób, jak aplikacje Java SE .

## CCDTURL właściwość

Jeśli środowisko Java EE udostępnia pulę połączeń dla połączeń JMS , należy zapoznać się z sekcją “Używanie listy nazw połączeń (CONNECTIONNAMELIST) lub tabeli definicji kanału klienta (CCDT) w puli połączeń” na stronie 308 , aby uzyskać informacje na temat wpływu na zachowanie właściwości **CCDTURL** .

Jeśli środowisko Java EE nie udostępnia puli połączeń JMS . Aplikacja używa właściwości **CCDTURL** w taki sam sposób, jak aplikacje Java SE .

*Używanie listy nazw połączeń (CONNECTIONNAMELIST) lub tabeli definicji kanału klienta (CCDT) w puli połączeń*
Niektóre środowiska Java EE , na przykład WebSphere Application Server, udostępniają pulę połączeń z systemem JMS . Kontener, który może być używany do uruchamiania aplikacji Java SE .

Aplikacje, które tworzą połączenie przy użyciu fabryki połączeń zdefiniowanej w środowisku Java EE , uzyskują istniejące wolne połączenie z puli połączeń dla tej fabryki połączeń lub nowe połączenie, jeśli nie istnieje odpowiednie połączenie w puli połączeń.

Może to mieć wpływ, jeśli fabryka połączeń została skonfigurowana ze zdefiniowaną właściwością **CONNECTIONNAMELIST** lub **CCDTURL** .

Przy pierwszym użyciu fabryki połączeń w celu utworzenia połączenia środowisko Java EE używa **CONNECTIONNAMELIST**. lub **CCDTURL** , aby utworzyć nowe połączenie z systemem IBM MQ . Jeśli to połączenie nie jest już potrzebne, jest ono zwracane do puli połączeń, w której połączenie staje się dostępne do ponownego wykorzystania.

Jeśli coś innego tworzy połączenie z fabryki połączeń, środowisko Java EE zwraca połączenie z puli połączeń zamiast używać właściwości **CONNECTIONNAMELIST** lub **CCDTURL** do utworzenia nowego połączenia.

Jeśli połączenie jest używane w przypadku niepowodzenia instancji menedżera kolejek, połączenie jest odrzucane. Jednak zawartość puli połączeń może nie być, co oznacza, że pula może nadal zawierać połączenia z menedżerem kolejek, który nie jest już uruchomiony.

W takiej sytuacji przy następnym żądaniu utworzenia połączenia z fabryki połączeń zwracane jest połączenie z menedżerem kolejek, którego przetwarzanie nie powiodło się. Wszelkie próby użycia tego połączenia nie powiodły się, ponieważ menedżer kolejek nie jest już uruchomiony, co spowodowało odrzucenie połączenia.

Tylko wtedy, gdy pula połączeń jest pusta, środowisko Java EE użyje właściwości **CONNECTIONNAMELIST** lub **CCDTURL** w celu utworzenia nowego połączenia z serwerem IBM MQ.

Ze względu na sposób, w jaki **CONNECTIONNAMELIST** i CCDT są używane do tworzenia połączeń JMS , możliwe jest również utworzenie puli połączeń, która zawiera połączenia z różnymi systemami IBM MQ .

Załóżmy na przykład, że fabryka połączeń została skonfigurowana z właściwością **CONNECTIONNAMELIST** ustawioną na następującą wartość:

```
CONNECTIONNAMELIST = hostname1(port1), hostname2(port2)
```

Załóżmy, że przy pierwszej próbie utworzenia przez aplikację połączenia z autonomicznym menedżerem kolejek z tej fabryki połączeń menedżer kolejek działający w systemie `hostname1(port1)` nie jest dostępny. Oznacza to, że aplikacja nawiąże połączenie z menedżerem kolejek działającym w systemie `hostname2(port2)`.

Inna aplikacja tworzy teraz połączenie JMS z tej samej fabryki połączeń. Menedżer kolejek w systemie `hostname1(port1)` jest teraz dostępny, dlatego zostało utworzone nowe połączenie JMS z tym systemem IBM MQ i jest ono zwracane do aplikacji.

Po zakończeniu działania obie aplikacje zamykają swoje JMS Connections, co powoduje, że połączenia są zwracane do puli połączeń.

W rezultacie pula połączeń dla fabryki połączeń zawiera teraz dwa połączenia JMS :

- Jedno połączenie z menedżerem kolejek działającym w systemie `hostname1(port1)`
- Jedno połączenie z menedżerem kolejek działającym w systemie `hostname2(port2)`

Może to prowadzić do problemów związanych z odtwarzaniem transakcji. Jeśli system Java EE musi wycofać transakcję, musi mieć możliwość nawiązania połączenia z menedżerem kolejek, który ma dostęp do dzienników transakcji.

*Implementowanie logiki ponownego połączenia w aplikacji Java EE*
Komponenty EJB (Enterprise JavaBeans ) i aplikacje z interfejsem WWW, które mają automatycznie nawiązywać ponowne połączenie w przypadku niepowodzenia menedżera kolejek, muszą implementować własną logikę ponownego połączenia.

Poniższe opcje zawierają więcej informacji na temat sposobu, w jaki można to osiągnąć.

## Zezwalaj aplikacji na niepowodzenie

To podejście nie wymaga żadnych zmian w aplikacji, ale wymaga administracyjnej rekonfiguracji definicji fabryki połączeń w celu uwzględnienia właściwości **CONNECTIONNAMELIST** . Jednak takie podejście wymaga, aby użytkownik wywołujący był w stanie odpowiednio obsłużyć awarię. Należy zauważyć, że jest to również wymagane w przypadku awarii, takich jak MQRC_Q_FULL, które nie są związane z awarią połączenia.

Przykładowy kod dla tego procesu:

```
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 (JMSException je) {
 // process exception
   }
 }
}
```

W powyższym kodzie założono, że fabryka połączeń używana przez ten serwlet ma zdefiniowaną właściwość **CONNECTIONNAMELIST** .

Podczas pierwszego przetwarzania serwletu jest tworzone nowe połączenie przy użyciu właściwości **CONNECTIONNAMELIST** , przy założeniu, że żadne zestawione połączenia nie są dostępne z innych aplikacji nawiązujących połączenie z tym samym menedżerem kolejek.

Po zwolnieniu połączenia po wywołaniu metody `close()` połączenie to jest zwracane do puli i ponownie wykorzystywane przy następnym uruchomieniu serwletu-bez odwoływania się do metody **CONNECTIONNAMELIST** -do momentu wystąpienia awarii połączenia, w którym to momencie zostanie wygenerowane zdarzenie CONNECTION_ERROR_OCCURRED. To zdarzenie prosi pulę o zniszczenie połączenia zakończonego niepowodzeniem.

Podczas następnego uruchomienia aplikacji nie jest dostępne żadne połączenie z puli i do nawiązania połączenia z pierwszym dostępnym menedżerem kolejek używany jest program **CONNECTIONNAMELIST** . Jeśli wystąpiło przełączenie awaryjne menedżera kolejek (na przykład niepowodzenie nie było przejściowym awarią sieci), serwlet łączy się z instancją zapasową, gdy tylko jest ona dostępna.

Jeśli w aplikacji są zaangażowane inne zasoby, takie jak bazy danych, może być wskazane wskazanie, że serwer aplikacji powinien wycofać transakcję.

## Obsługa ponownego połączenia w aplikacji

Jeśli użytkownik wywołujący nie może przetworzyć niepowodzenia z serwletu, należy obsłużyć ponowne połączenie w aplikacji. Jak pokazano w poniższym przykładzie, do obsługi ponownego połączenia w aplikacji wymagane jest, aby aplikacja zażądała nowego połączenia, aby mogła buforować fabrykę połączeń wyszukaną w produkcie JNDI i obsłużyć wywołanie `JMSException` , takie jak JMSCMQ0001: `WebSphere MQ nie powiodło się z kodem compcode '2' ('MQCC_FAILED '), przyczyna' 2009 '('MQRC_CONNECTION_BROKEN')`.

```
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();
    }
  }
```

W poniższym przykładzie program `setupResources()` tworzy obiekty JMS i dołącza pętlę uśpienia i ponowienia w celu obsługi ponownego połączenia innego niż natychmiastowe. W praktyce ta metoda zapobiega wielu próbom ponownego nawiązania połączenia. Należy zauważyć, że warunki wyjścia zostały pominięte w przykładzie dla zachowania przejrzystości.

```
 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) {}
      }
    }
```

Jeśli aplikacja zarządza ponownym połączeniem, ważne jest, aby zwolnić wszystkie połączenia utrzymywane do innych zasobów, niezależnie od tego, czy są to inne menedżery kolejek produktu IBM MQ , czy inne usługi zaplecza, takie jak bazy danych. Po zakończeniu ponownego nawiązywania połączenia z nową instancją menedżera kolejek produktu IBM MQ należy ponownie nawiązać te połączenia. Jeśli połączenia nie zostaną ponownie nawiązane, zasoby serwera aplikacji zostaną niepotrzebnie wstrzymane podczas próby ponownego połączenia i może nastąpić przekroczenie limitu czasu przed ponownym wykorzystaniem tych zasobów.

## Korzystanie z menedżera WorkManager

W przypadku długotrwałych aplikacji (na przykład przetwarzania wsadowego), w których czas przetwarzania jest dłuższy niż kilka dziesiątek sekund, można użyć menedżera WebSphere Application Server WorkManager . Poniżej przedstawiono przykładowy fragment kodu dla elementu WebSphere Application Server :

```
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());
    }
}
```

gdzie `web.xml` zawiera:

```
<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>
```

a zadanie wsadowe jest teraz implementowane za pośrednictwem interfejsu roboczego:

```
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 (JMSException je) {
          // drive reconnection
          setupResources();
      }
  }

  public boolean isRunning() {return !sendComplete;}

  public void release() {sendComplete = true;}
```

Jeśli przetwarzanie wsadowe zajmuje dużo czasu, na przykład duże komunikaty, wolna sieć lub rozległy dostęp do bazy danych (szczególnie w połączeniu z powolnym przełączaniem awaryjnym), serwer rozpoczyna wyświetlanie ostrzeżeń zawieszonego wątku, podobnie jak w poniższym przykładzie:

WSVR0605W: Wątek "`WorkManager.DefaultWorkManager` : 0" (00000035) był aktywny przez 694061 milisekund i może być zawieszony. Na serwerze znajduje się łącznie 1 wątek (y), który może być zawieszony.

Te ostrzeżenia można zminimalizować, zmniejszając wielkość zadania wsadowego lub zwiększając limit czasu zawieszonego wątku. Zaleca się jednak zaimplementowanie tego przetwarzania w komponencie EJB (w przypadku wysyłania wsadowego) lub w komponencie bean sterowanym komunikatami (w przypadku konsumowania lub konsumowania i odpowiadania).

Należy zauważyć, że ponowne połączenie zarządzane przez aplikację nie stanowi ogólnego rozwiązania do obsługi błędów w czasie wykonywania, a aplikacja nadal musi obsługiwać błędy, które nie są powiązane z awarią połączenia.

Na przykład próba umieszczenia komunikatu w pełnej kolejce (2053 MQRC_Q_FULL) lub próba nawiązania połączenia z menedżerem kolejek przy użyciu niepoprawnych referencji zabezpieczeń (2035 MQRC_NOT_AUTHORIZED).

Aplikacja musi również obsłużyć błędy 2059 MQRC_Q_MGR_NOT_AVAILABLE, gdy żadne instancje nie są dostępne natychmiast po przełączeniu awaryjnym. Można to osiągnąć, zgłaszając wyjątki JMS w momencie ich wystąpienia, zamiast podejmowania cichej próby ponownego nawiązania połączenia.

*IBM MQ classes for JMS zestawianie obiektów*
Korzystanie z zestawiania połączeń poza programem Java EE pomaga zmniejszyć ogólne obciążenie wynikające na przykład z niektórych autonomicznych aplikacji korzystających ze środowisk lub wdrażanych w środowiskach chmurowych, a także z większej liczby połączeń klienckich w programie `QueueManagers` , co prowadzi do zwiększenia konsolidacji aplikacji i menedżerów kolejek na serwerze.

W modelu programistycznym Java EE istnieje dobrze zdefiniowany cykl życia różnych używanych obiektów. Komponenty bean sterowane komunikatami (MDB) są najbardziej ograniczone, podczas gdy serwlety zapewniają większą swobodę. Dlatego opcje zestawiania połączeń dostępne na serwerach Java EE są zgodne z różnymi używanymi modelami programistycznymi.

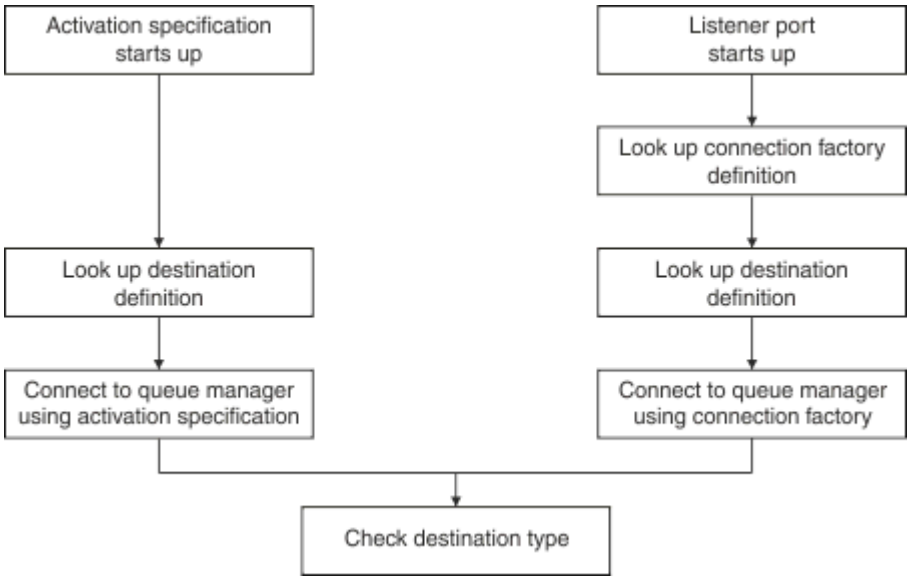
W produkcie Java SE (lub innym środowisku, takim jak Spring) modele programistyczne są niezwykle elastyczne. Dlatego też pojedyncza strategia łączenia w pule nie odpowiada wszystkim. Należy rozważyć, czy istnieją ramy, które mogłyby stworzyć jakąkolwiek formę łączenia, na przykład wiosna.

Strategia łączenia w pule, która ma być używana, zależy od środowiska, w którym działa aplikacja.

*Zestawianie obiektów w środowisku Java EE*
Java EE application servers provide connection pooling functionality that can be used by message-driven bean applications, Enterprise Java Beans and Servlets.

WebSphere Application Server utrzymuje pulę połączeń z dostawcą JMS w celu zwiększenia wydajności. Gdy aplikacja tworzy połączenie JMS , serwer aplikacji określa, czy połączenie już istnieje w puli wolnych połączeń. Jeśli tak, połączenie jest zwracane do aplikacji; w przeciwnym razie tworzone jest nowe połączenie.

Rysunek 41 na stronie 313 pokazuje, w jaki sposób zarówno specyfikacje aktywowania, jak i porty nasłuchiwania nawiązują połączenie JMS i używają tego połączenia do monitorowania miejsca docelowego pod kątem komunikatów w trybie normalnym.




*Rysunek 41. Tryb standardowy*

Jeśli używany jest dostawca przesyłania komunikatów produktu IBM MQ , aplikacje, które wykonują przesyłanie komunikatów wychodzących (takie jak komponenty EJB (Enterprise Java Beans) i serwlety), oraz komponent portu nasłuchiwania komponentu bean sterowanego komunikatami mogą korzystać z tych pul połączeń.

Specyfikacje aktywowania dostawcy przesyłania komunikatów produktu IBM MQ korzystają z funkcji zestawiania połączeń udostępnianych przez adapter zasobów IBM MQ . Więcej informacji na ten temat zawiera sekcja Konfigurowanie właściwości adaptera zasobów produktu WebSphere MQ .

W sekcji “Przykłady użycia puli połączeń” na stronie 317 wyjaśniono, w jaki sposób aplikacje, które wykonują przesyłanie komunikatów wychodzących i porty nasłuchiwania, korzystają z wolnej puli podczas tworzenia połączeń JMS .

“Wolne wątki konserwacji puli połączeń” na stronie 320 wyjaśnia, co dzieje się z tymi połączeniami, gdy aplikacja lub port nasłuchiwania zakończy połączenia.

“Przykłady wątków konserwacji puli” na stronie 321 wyjaśnia, w jaki sposób wolna pula połączeń jest czyszczona, aby zapobiec stawaniu się nieaktualnych połączeń JMS .

Produkt WebSphere Application Server ma limit liczby połączeń, które można utworzyć z fabryki, określony przez właściwość *Maksymalna liczba połączeń* fabryki połączeń. Wartością domyślną tej właściwości jest 10, co oznacza, że w dowolnym momencie może zostać utworzonych maksymalnie 10 połączeń z fabryki.

Z każdą fabryką jest powiązana wolna pula połączeń. Po uruchomieniu serwera aplikacji pule wolnych połączeń są puste. Maksymalna liczba połączeń, które mogą istnieć w wolnej puli dla fabryki, jest również określona przez właściwość Maksymalna liczba połączeń.

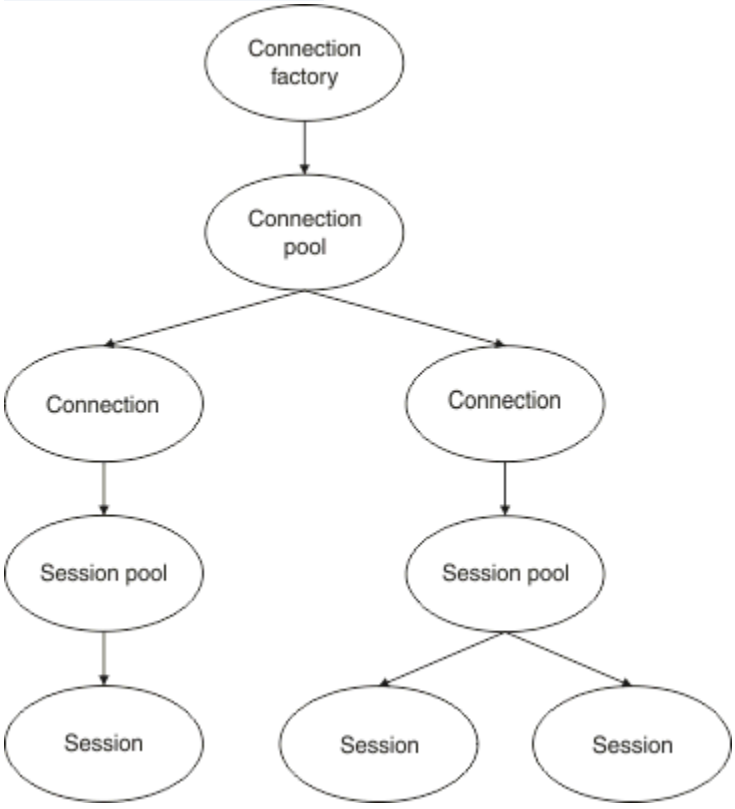
**Wskazówka:** W produkcie JMS 2.0fabryka połączeń może być używana do tworzenia zarówno połączeń, jak i kontekstów. W związku z tym możliwe jest powiązanie puli połączeń z fabryką połączeń, która zawiera zarówno połączenia, jak i konteksty. Zaleca się, aby fabryka połączeń była używana tylko do tworzenia połączeń lub kontekstów. Dzięki temu pula połączeń dla tej fabryki połączeń będzie zawierać tylko obiekty pojedynczego typu, co spowoduje, że pula będzie bardziej wydajna.

Informacje na temat sposobu działania zestawiania połączeń w produkcie WebSphere Application Serverzawiera sekcja Konfigurowanie zestawiania połączeń dla połączeń JMS. W przypadku innych serwerów aplikacji należy zapoznać się z odpowiednią dokumentacją serwera aplikacji.

## Sposób użycia puli połączeń

Z każdą fabryką połączeń JMS jest powiązana pula połączeń, a pula połączeń zawiera zero lub więcej połączeń JMS . Każde połączenie JMS ma powiązaną pulę sesji JMS , a każda pula sesji JMS zawiera zero lub więcej sesji JMS .

Rysunek 42 na stronie 314 przedstawia relacje między tymi obiektami.




*Rysunek 42. Pule połączeń i pule sesji*

Podczas uruchamiania portu nasłuchiwania lub gdy aplikacja, która ma wykonywać przesyłanie komunikatów wychodzących, używa fabryki do utworzenia połączenia, port lub aplikacja wywołuje jedną z następujących metod:

- **`connectionFactory.createConnection()`**
- **`ConnectionFactory.createConnection(String, String)`**

- **QueueConnectionFactory.createQueueConnection()**
- **QueueConnectionFactory.createQueueConnection(String, String)**
- **TopicConnectionFactory.createTopicConnection()**
- **TopicConnectionFactory.createTopicConnection(String, String)**

Menedżer połączeń WebSphere Application Server próbuje uzyskać połączenie z wolnej puli dla tej fabryki i zwrócić je do aplikacji.

Jeśli w puli nie ma wolnych połączeń, a liczba połączeń utworzonych z tej fabryki nie osiągnęła limitu określonego we właściwości *maksymalna liczba połączeń* tej fabryki, menedżer połączeń utworzy nowe połączenie, które będzie używane przez aplikację.

Jeśli jednak aplikacja próbuje utworzyć połączenie, ale liczba połączeń utworzonych z tej fabryki jest już równa właściwości fabryki *maksymalna liczba połączeń* , aplikacja oczekuje na udostępnienie połączenia (do umieszczenia w puli wolnych połączeń).

Czas oczekiwania aplikacji jest określony we właściwości *limit czasu połączenia* puli połączeń, która ma wartość domyślną 180 sekund. Jeśli połączenie zostanie ponownie umieszczone w puli wolnych połączeń w ciągu tego 180 sekund, menedżer połączeń natychmiast ponownie pobiera je z puli i przekazuje je do aplikacji. Jeśli jednak upłynie limit czasu, zgłaszany jest wyjątek *ConnectionWaitTimeoutException* .

Gdy aplikacja zakończy połączenie i zamknie je, wywołując:

- **Connection.close()**
- **QueueConnection.close()**
- **TopicConnection.close()**

Połączenie jest w rzeczywistości utrzymywane jako otwarte i zwracane do wolnej puli, dzięki czemu może być ponownie wykorzystywane przez inną aplikację. Dlatego połączenia między produktem WebSphere Application Server i dostawcą JMS mogą być otwarte, nawet jeśli na serwerze aplikacji nie są uruchomione żadne aplikacje JMS .

*Zaawansowane właściwości puli połączeń*
Istnieje wiele właściwości zaawansowanych, których można użyć do sterowania zachowaniem pul połączeń JMS .

## Ochrona przeciwprzepięciowa

"W jaki sposób aplikacje wykonujące przesyłanie komunikatów wychodzących korzystają z puli połączeń" na stronie 319 opisuje sposób użycia metody `sendMessage()`, która zawiera `connectionFactory.createConnection()`.

Należy rozważyć sytuację, w której 50 komponentów EJB tworzy połączenia JMS z tej samej fabryki połączeń w ramach swojej metody `ejbCreate()` .

Jeśli wszystkie te komponenty bean są tworzone w tym samym czasie, a w puli wolnych połączeń fabryki nie ma żadnych połączeń, serwer aplikacji próbuje jednocześnie utworzyć 50 połączeń JMS z tym samym dostawcą JMS . Wynikiem jest znaczne obciążenie zarówno produktu WebSphere Application Server , jak i dostawcy JMS .

Właściwości ochrony przed przeciążeniem mogą zapobiec takiej sytuacji, ograniczając liczbę połączeń JMS , które można utworzyć z fabryki połączeń w dowolnym momencie, i zmniejszając w ten sposób tworzenie dodatkowych połączeń.

Ograniczenie liczby połączeń JMS w dowolnym momencie jest osiągane za pomocą dwóch właściwości:

- Próg przeciążenia
- Interwał tworzenia przeciążenia.

Gdy aplikacje EJB próbują utworzyć połączenie JMS z fabryki połączeń, menedżer połączeń sprawdza, ile połączeń jest tworzonych. Jeśli ta liczba jest mniejsza lub równa wartości właściwości `surge threshold` , menedżer połączeń kontynuuje otwieranie nowych połączeń.

Jeśli jednak liczba tworzonych połączeń przekracza wartość właściwości `surge threshold` , menedżer połączeń czeka przez czas określony we właściwości `surge creation interval` przed utworzeniem i otwarciem nowego połączenia.

## Zablokowane połączenia

JMS Połączenie jest uważane za `stuck`, jeśli aplikacja JMS używa tego połączenia do wysłania żądania do dostawcy JMS , a dostawca nie odpowie w określonym czasie.

WebSphere Application Server umożliwia wykrywanie połączeń produktu `stuck` JMS . Aby użyć tej funkcji, należy ustawić trzy właściwości:

- Licznik czasu zablokowanych połączeń
- Czas blokady
- Próg zablokowania

"Przykłady wątków konserwacji puli" na stronie 321 wyjaśnia, w jaki sposób wątek konserwacji puli jest uruchamiany okresowo i sprawdza zawartość wolnej puli fabryki połączeń, szukając połączeń, które nie były używane przez określony czas lub istniały przez zbyt długi czas.

Aby wykryć zablokowane połączenia, serwer aplikacji zarządza również wątkiem zablokowanych połączeń, który sprawdza stan wszystkich aktywnych połączeń utworzonych z fabryki połączeń w celu sprawdzenia, czy którekolwiek z nich oczekują na odpowiedź od dostawcy JMS .

Gdy wątek zablokowanych połączeń jest uruchamiany, jest określany przez właściwość `Stuck time timer` . Wartością domyślną tej właściwości jest zero, co oznacza, że wykrywanie zablokowanych połączeń nigdy nie jest uruchamiane.

Jeśli wątek znajdzie wątek oczekujący na odpowiedź, określa czas oczekiwania i porównuje ten czas z wartością właściwości `Stuck time` .

Jeśli czas odpowiedzi dostawcy JMS przekracza czas określony we właściwości `Stuck time` , serwer aplikacji oznacza połączenie JMS jako zablokowane.

Załóżmy na przykład, że fabryka połączeń `jms/CF1` ma właściwość `Stuck time timer` ustawioną na 10, a właściwość `Stuck time` na 15.

Wątek zablokowanych połączeń staje się aktywny co 10 sekund i sprawdza, czy jakiekolwiek połączenie utworzone w produkcie `jms/CF1` oczekiwało dłużej niż 15 sekund na odpowiedź z produktu IBM MQ.

Załóżmy, że komponent EJB tworzy połączenie JMS z bazą danych IBM MQ przy użyciu metody `jms/CF1`, a następnie próbuje utworzyć sesję JMS przy użyciu tego połączenia, wywołując metodę `Connection.createSession()`.

Jednak pewne rzeczy uniemożliwiają dostawcy JMS udzielenie odpowiedzi na żądanie. Być może komputer został zamrożony lub proces uruchomiony w dostawcy JMS jest zakleszczony, co uniemożliwia przetworzenie nowej pracy:

Dziesięć sekund po wywołaniu komponentu EJB o nazwie `Connection.createSession()` licznik czasu zablokowanych połączeń staje się aktywny i sprawdza aktywne połączenia utworzone z poziomu programu `jms/CF1`.

Załóżmy, że istnieje tylko jedno aktywne połączenie, na przykład `c1`. Pierwszy komponent EJB oczekiwał przez 10 sekund na odpowiedź na żądanie wysłane do `c1`, która jest mniejsza niż wartość `Stuck time`, dlatego licznik czasu zablokowanych połączeń ignoruje to połączenie i staje się nieaktywny.

10 sekund później wątek zablokowanych połączeń ponownie staje się aktywny i sprawdza aktywne połączenia dla `jms/CF1`. Przyjmijmy, że istnieje tylko jedno połączenie, `c1`.

Od pierwszego komponentu EJB o nazwie `createSession()` minęło 20 sekund, a komponent EJB nadal oczekuje na odpowiedź. 20 sekund jest dłuższych niż czas określony we właściwości `Stuck time` , dlatego wątek zablokowanych połączeń oznacza `c1` jako zablokowany.

Jeśli po upływie pięciu sekund produkt IBM MQ na końcu odpowie i zezwoli pierwszemu EJB na utworzenie sesji JMS , połączenie jest ponownie używane.

Serwer aplikacji zlicza liczbę zablokowanych połączeń JMS utworzonych z fabryki połączeń. Jeśli aplikacja używa tej fabryki połączeń do utworzenia nowego połączenia JMS i w puli wolnych połączeń tej fabryki nie ma wolnych połączeń, menedżer połączeń porównuje liczbę zablokowanych połączeń z wartością właściwości `Stuck threshold` .

Jeśli liczba zablokowanych połączeń jest mniejsza niż wartość ustawiona dla właściwości `Stuck threshold`, menedżer połączeń tworzy nowe połączenie i przekazuje je do aplikacji.

Jeśli jednak liczba zablokowanych połączeń jest równa wartości właściwości `Stuck threshold` , aplikacja otrzymuje wyjątek zasobu.

## Partycje puli

WebSphere Application Server udostępnia dwie właściwości, które umożliwiają partycjonowanie wolnej puli połączeń dla fabryki połączeń:

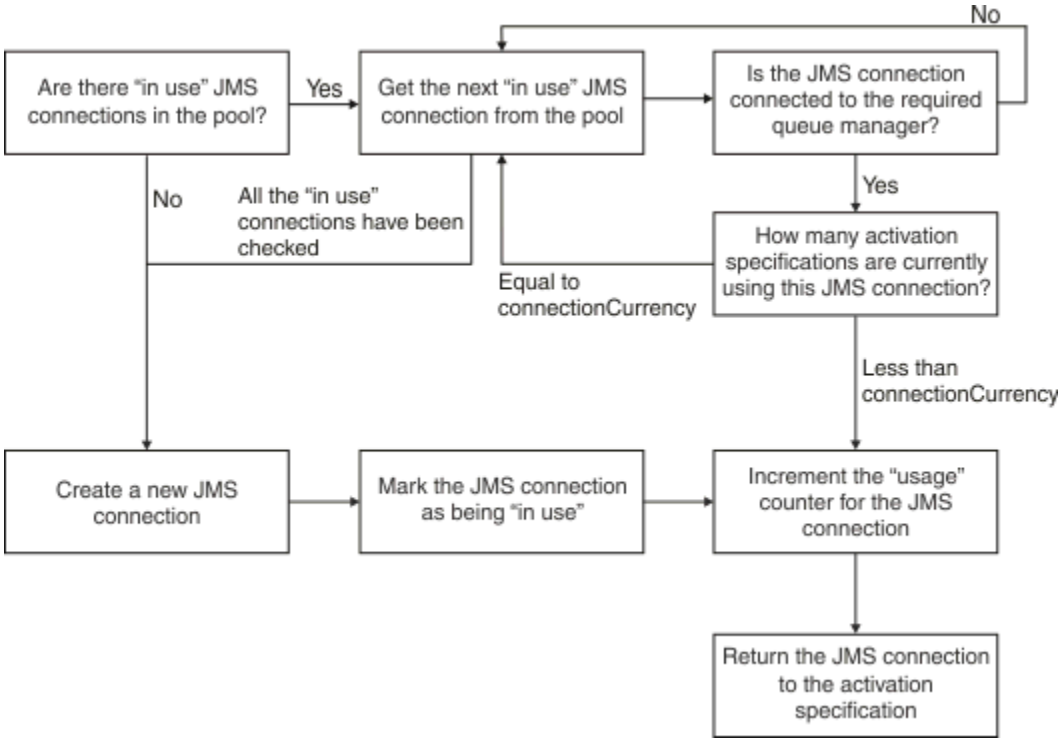
- Parametr `Number of free pool partitions` informuje serwer aplikacji o liczbie partycji, na które ma zostać podzielona wolna pula połączeń.
- `Free pool distribution table size` określa sposób indeksowania partycji.

Należy pozostawić domyślne wartości tych właściwości równe zero, chyba że użytkownik zostanie poproszony o ich zmianę przez Centrum wsparcia IBM .

Należy zauważyć, że WebSphere Application Server ma jedną dodatkową zaawansowaną właściwość puli połączeń o nazwie `Number of shared partitions`. Ta właściwość określa liczbę partycji używanych do przechowywania połączeń współużytkowanych. Ponieważ jednak połączenia JMS są zawsze niewspółużytkowane, ta właściwość nie ma zastosowania.

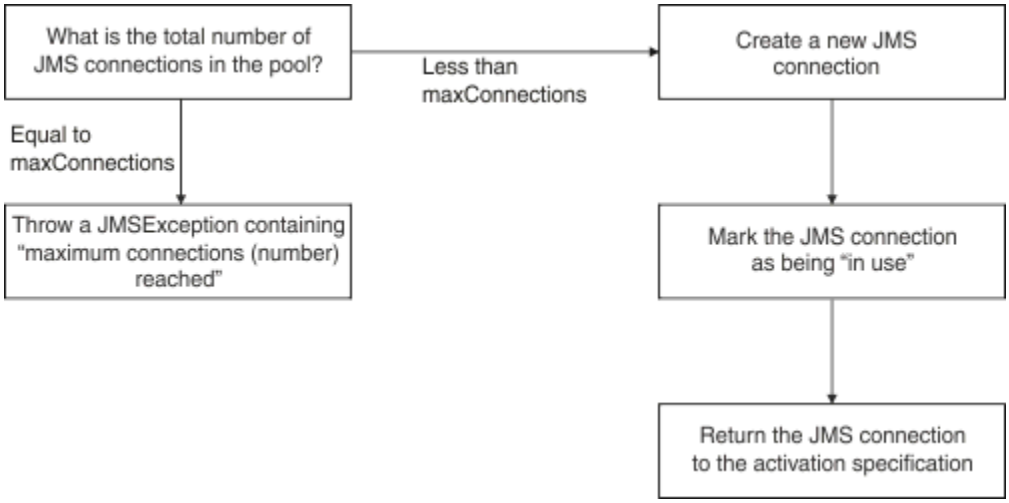
*Przykłady użycia puli połączeń*
Komponent portu nasłuchiwania komponentu bean sterowanego komunikatami oraz aplikacje, które wykonują przesyłanie komunikatów wychodzących, używają puli połączeń JMS .

Rysunek 43 na stronie 317 przedstawia sposób działania puli połączeń dla produktu WebSphere Application Server V7.5 i produktu V8.0.




*Rysunek 43. WebSphere Application Server V7.5 i V8.0 -sposób działania puli połączeń*

Rysunek 44 na stronie 318 przedstawia sposób działania puli połączeń dla produktu WebSphere Application Server V8.5.




*Rysunek 44. WebSphere Application Server V8.5 -sposób działania puli połączeń*

## W jaki sposób porty nasłuchiwania MDB używają puli połączeń

Załóżmy, że w systemie WebSphere Application Server Network Deployment wdrożono komponent MDB, który używa IBM MQ jako dostawcy JMS . Komponent MDB jest wdrażany dla portu nasłuchiwania używającego fabryki połączeń o nazwie na przykład `jms/CF1`, która ma właściwość *maksymalna liczba połączeń* ustawioną na 2, co oznacza, że w danym momencie mogą być tworzone tylko dwa połączenia z tej fabryki.

Po uruchomieniu portu nasłuchiwania port próbuje utworzyć połączenie z bazą danych IBM MQprzy użyciu fabryki połączeń`jms/CF1` .

W tym celu port żąda połączenia od menedżera połączeń. Ponieważ fabryka połączeń produktu `jms/CF1` jest używana po raz pierwszy, w puli wolnych połączeń produktu `jms/CF1` nie ma żadnych połączeń, dlatego menedżer połączeń tworzy nowe połączenie o nazwie, na przykład `c1`. Należy zauważyć, że to połączenie istnieje przez cały czas życia portu nasłuchiwania.

Teraz należy rozważyć sytuację, w której port nasłuchiwania jest zatrzymany przy użyciu Konsoli administracyjnej serwera WebSphere Application Server . W takim przypadku menedżer połączeń pobiera połączenie i umieszcza je z powrotem w wolnej puli. Jednak połączenie z bazą danych IBM MQ pozostaje otwarte.

Jeśli port nasłuchiwania zostanie zrestartowany, menedżer połączeń zostanie ponownie poproszony o nawiązanie połączenia z menedżerem kolejek. Ponieważ w puli wolnych połączeń znajduje się teraz połączenie (`c1`), menedżer połączeń pobiera to połączenie z puli i udostępnia je dla portu nasłuchiwania.

Załóżmy, że na serwerze aplikacji wdrożono drugi komponent MDB, który używa innego portu nasłuchiwania.

Załóżmy, że użytkownik próbuje uruchomić trzeci port nasłuchiwania, który jest również skonfigurowany do używania fabryki połączeń `jms/CF1` . Trzeci port nasłuchiwania żąda połączenia od menedżera połączeń, który wyszukuje w puli wolnych połączeń wartość `jms/CF1` i stwierdza, że jest ona pusta. Następnie sprawdza, ile połączeń zostało już utworzonych z fabryki `jms/CF1` .

Ponieważ właściwość maksymalnej liczby połączeń dla `jms/CF1` jest ustawiona na 2, a użytkownik utworzył już dwa połączenia z tej fabryki, menedżer połączeń czeka 180 sekund (wartość domyślna właściwości limitu czasu połączenia) na udostępnienie połączenia.

Jeśli jednak zostanie zatrzymany pierwszy port nasłuchiwania, jego połączenie `c1` zostanie umieszczone w puli wolnych połączeń dla `jms/CF1`. Menedżer połączeń pobiera to połączenie i przekazuje je do trzeciego programu nasłuchującego.

Jeśli teraz zostanie podjęta próba zrestartowania pierwszego programu nasłuchującego, musi on oczekiwać na zatrzymanie jednego z pozostałych portów programu nasłuchującego, zanim będzie można zrestartować pierwszy program nasłuchujący. Jeśli żaden z działających portów nasłuchiwania nie zostanie zatrzymany w ciągu 180 sekund, pierwszy proces nasłuchiwania otrzyma błąd `ConnectionWaitTimeoutException` i zostanie zatrzymany.

## W jaki sposób aplikacje wykonujące przesyłanie komunikatów wychodzących korzystają z puli połączeń

W przypadku tej opcji załóżmy, że istnieje pojedynczy komponent EJB o nazwie, na przykład `EJB1`, zainstalowany na serwerze aplikacji. Komponent bean implementuje metodę o nazwie `sendMessage()` przez:

- Tworzenie JMS połączenia z IBM MQ z fabryki `jms/CF1`za pomocą `connectionFactory.createConnection()`.
- Tworzenie sesji JMS na podstawie połączenia.
- Tworzenie producenta komunikatów z sesji.
- Wysyłanie komunikatu.
- Zamykanie producenta.
- Zamykanie sesji.
- Zamykanie połączenia przez wywołanie funkcji `connection.close()`.

Załóżmy, że wolna pula dla fabryki `jms/CF1` jest pusta. Gdy komponent EJB jest wywoływany po raz pierwszy, próbuje on utworzyć połączenie z bazą danych IBM MQ z fabryki `jms/CF1`. Ponieważ wolna pula dla fabryki jest pusta, menedżer połączeń tworzy nowe połączenie i przekazuje je do komponentu `EJB1`.

Tuż przed zakończeniem działania metody wywoływana jest metoda `connection.close()`. Zamiast zamykać program `c1`, menedżer połączeń pobiera połączenie i umieszcza je w puli wolnych połączeń dla programu `jms/CF1`.

Przy następnym wywołaniu metody `sendMessage()` metoda `connectionFactory.createConnection()` zwraca wartość `c1` do aplikacji.

Załóżmy, że istnieje druga instancja komponentu EJB działająca w tym samym czasie co pierwsza instancja. Gdy obie instancje wywołują funkcję `sendMessage()`, tworzone są dwa połączenia z fabryki połączeń produktu `jms/CF1` .

Załóżmy teraz, że utworzono trzecią instancję komponentu bean. Gdy trzeci komponent bean wywołuje metodę `sendMessage()`, metoda wywołuje metodę `connectionFactory.createConnection()` w celu utworzenia połączenia z pliku `jms/CF1`.

Istnieją jednak obecnie dwa połączenia utworzone z produktu `jms/CF1`, które są równe wartości maksymalnej liczby połączeń dla tej fabryki. Dlatego metoda `createConnection()` oczekuje przez 180 sekund (wartość domyślna właściwości limitu czasu połączenia) na udostępnienie połączenia.

Jeśli jednak metoda `sendMessage()` dla pierwszego wywołania komponentu EJB `connection.close()` i wyjścia, używane połączenie ( `c1`) zostanie ponownie umieszczone w puli wolnych połączeń. Menedżer połączeń pobiera połączenie z powrotem z wolnej puli i przekazuje je do trzeciego komponentu EJB. Wywołanie z tego komponentu bean do metody `connectionFactory.createConnection()` powoduje następnie powrót, co umożliwia zakończenie działania metody `sendMessage()`.

## Porty nasłuchiwania MDB i komponenty EJB korzystające z tej samej puli połączeń

Dwa poprzednie przykłady pokazują, w jaki sposób porty nasłuchiwania i komponenty EJB mogą korzystać z puli połączeń w izolacji. Można jednak uruchomić zarówno port nasłuchiwania, jak i komponent EJB na tym samym serwerze aplikacji, a także utworzyć połączenia JMS przy użyciu tej samej fabryki połączeń.

Należy wziąć pod uwagę konsekwencje tej sytuacji

Należy pamiętać, że fabryka połączeń jest współużytkowana przez port nasłuchiwania i komponent EJB.

Załóżmy na przykład, że w tym samym czasie działa obiekt nasłuchiwania i komponent EJB. Obie fabryki połączeń korzystają z fabryki połączeń `jms/CF1` , co oznacza, że osiągnięto limit połączeń określony przez właściwość maksymalnej liczby połączeń dla tej fabryki.

Jeśli zostanie podjęta próba uruchomienia innego portu nasłuchiwania lub innej instancji komponentu EJB, należy poczekać, aż połączenie zostanie zwrócone do puli wolnych połączeń dla serwera `jms/CF1`.

*Wolne wątki konserwacji puli połączeń*
Powiązana z każdą wolną pulą połączeń jest wątek konserwacji puli, który monitoruje wolną pulę, aby upewnić się, że połączenia w niej zawarte są nadal poprawne.

Jeśli wątek konserwacji puli zdecyduje, że konieczne jest usunięcie połączenia z wolnej puli, wątek fizycznie zamknie połączenie JMS z serwerem IBM MQ.

## Sposób działania wątku konserwacji puli

Zachowanie wątku konserwacji puli jest określane przez wartość czterech właściwości puli połączeń:

**Limit czasu starości**
Czas, przez jaki połączenie pozostaje otwarte.

**Minimalna liczba połączeń**
Minimalna liczba połączeń utrzymywanych przez menedżera połączeń w wolnej puli fabryki połączeń.

**Czas między sprawdzeniami**
Częstotliwość uruchamiania wątku konserwacji puli.

**Limit czasu nieużywanego połączenia**
Czas, przez jaki połączenie pozostaje w wolnej puli, zanim zostanie zamknięte.

Domyślnie wątek utrzymywany w puli jest uruchamiany co 180 sekund, ale tę wartość można zmienić, ustawiając właściwość puli połączeń **Reap time** .

Wątek konserwacji sprawdza każde połączenie w puli, sprawdza, jak długo znajdowało się w puli i ile czasu upłynęło od momentu jego utworzenia i ostatniego użycia.

Jeśli połączenie nie było używane przez okres dłuższy niż wartość właściwości **Unused timeout** dla puli połączeń, wątek konserwacji sprawdza liczbę połączeń znajdujących się obecnie w puli wolnych połączeń. Jeżeli liczba ta wynosi:

- Wartość większa niż **Minimum connections**oznacza, że menedżer połączeń zamyka połączenie.
- Równa się wartości parametru **Minimum connections**, połączenie nie jest zamykane i pozostaje w wolnej puli.

Wartością domyślną właściwości **Minimum connections** jest *1*, co oznacza, że ze względu na wydajność menedżer połączeń zawsze próbuje zachować co najmniej jedno połączenie w puli wolnych połączeń.

Wartością domyślną właściwości **Unused timeout** jest 1800 sekund. Domyślnie, jeśli połączenie zostanie ponownie umieszczone w puli wolnych połączeń i nie będzie ponownie używane przez co najmniej 1800 sekund, połączenie to zostanie zamknięte, pod warunkiem, że zostanie zamknięte, i pozostawi co najmniej jedno połączenie w puli wolnych połączeń.

Ta procedura zapobiega staniu się nieużywanych połączeń. Aby wyłączyć tę funkcję, należy ustawić właściwość **Unused timeout** na wartość zero.

Jeśli połączenie znajduje się w puli wolnych połączeń, a czas, który upłynął od jego utworzenia, jest większy niż wartość właściwości **Aged timeout** dla puli połączeń, jest ono zamykane niezależnie od czasu, który upłynął od ostatniego użycia.

Domyślnie właściwość **Aged timeout** jest ustawiona na zero, co oznacza, że wątek konserwacji nigdy nie wykonuje tego sprawdzenia. Połączenia, które były używane dłużej niż wynosi wartość właściwości **Aged timeout** , są odrzucane bez względu na to, ile połączeń pozostanie w wolnej puli. Należy zauważyć, że właściwość **Minimum connections** nie ma wpływu na tę sytuację.

## Wyłączanie wątku konserwacji puli

Z poprzedniego opisu wynika, że wątek konserwacji puli wykonuje wiele pracy, gdy jest aktywny, szczególnie jeśli w wolnej puli fabryki połączeń znajduje się duża liczba połączeń.

Na przykład można założyć, że istnieją trzy fabryki połączeń JMS z właściwością **Maximum connections** ustawioną na wartość 10 dla każdej fabryki. Co 180 sekund trzy wątki konserwacji puli stają się aktywne i skanują wolne pule odpowiednio dla każdej fabryki połączeń. Jeśli wolne pule mają wiele połączeń, wątki konserwacji mają wiele do zrobienia, co może znacząco wpłynąć na wydajność.

Wątek konserwacji puli można wyłączyć dla pojedynczej puli wolnych połączeń, ustawiając jej właściwość **Reap time** na wartość zero.

Wyłączenie wątku konserwacji oznacza, że połączenia nigdy nie są zamykane, nawet jeśli upłynął czas trwania **Unused timeout** . Jednak połączenia mogą być nadal zamykane, jeśli parametr **Aged timeout** został przekazany.

Po zakończeniu połączenia przez aplikację menedżer połączeń sprawdza, jak długo istnieje połączenie, a jeśli ten okres jest dłuższy niż wartość właściwości **Aged timeout** , menedżer połączeń zamyka połączenie, zamiast zwracać je do puli wolnych połączeń.

## Transakcyjne konsekwencje przekroczenia limitu czasu wieku

Zgodnie z opisem w poprzedniej sekcji właściwość **Aged timeout** określa, jak długo połączenie z dostawcą JMS pozostaje otwarte, zanim menedżer połączeń je zamknie.

Wartością domyślną właściwości **Aged timeout** jest zero, co oznacza, że połączenie nigdy nie zostanie zamknięte, ponieważ jest zbyt stare. Należy pozostawić tę wartość właściwości **Aged timeout** , ponieważ włączenie opcji **Aged timeout** może mieć wpływ na transakcje podczas używania serwera JMS wewnątrz komponentów EJB.

W systemie JMSjednostką transakcji jest JMS *sesja*, który jest tworzony na podstawie JMS *Połączenie*. Jest to *sesja* JMS , która jest rejestrowanych w transakcjach, a nie w *połączeniu* JMS .

Ze względu na projekt serwera aplikacji połączenia JMS mogą zostać zamknięte, ponieważ upłynął czas **Aged timeout** , nawet jeśli sesje JMS utworzone na podstawie tego połączenia są zaangażowane w transakcję.

Zamknięcie połączenia JMS powoduje wycofanie wszelkich zaległych operacji transakcyjnych w sesjach JMS, zgodnie ze specyfikacją JMS . Jednak serwer aplikacji nie wie, że sesje JMS utworzone na podstawie połączenia nie są już poprawne. Gdy serwer próbuje użyć sesji do zatwierdzenia lub wycofania transakcji, występuje błąd `IllegalStateException` .

**Ważne:** Aby używać serwera **Aged timeout** z połączeniami JMS z poziomu komponentów EJB, należy upewnić się, że wszystkie operacje JMS zostały jawnie zatwierdzone w sesji JMS przed wyjściem z metody komponentu EJB wykonującej operacje JMS .

*Przykłady wątków konserwacji puli*
Użycie przykładu EJB (Enterprise JavaBean ) do zrozumienia działania wątku konserwacji puli. Należy zauważyć, że można również używać komponentów bean sterowanych komunikatami (Message Driven Beans-MDB) i portów nasłuchiwania, ponieważ potrzebny jest tylko sposób na uzyskanie połączeń w wolnej puli.

Więcej informacji na temat metody `sendMessage()` zawiera sekcja "W jaki sposób aplikacje wykonujące przesyłanie komunikatów wychodzących korzystają z puli połączeń" na stronie 319 .

Fabryka połączeń została skonfigurowana z następującymi wartościami:

- **Reap time** z wartością domyślną 180 sekund
- **Aged timeout** przy domyślnej wartości zero sekund
- **Unused timeout** ustawione na 300 sekund

Po uruchomieniu serwera aplikacji wywoływana jest metoda `sendMessage()` .

Metoda tworzy połączenie wywoływane, na przykład c1, przy użyciu fabryki jms/CF1, używa tej fabryki do wysłania komunikatu, a następnie wywołuje funkcję connection.close(), co powoduje umieszczenie parametru c1 w puli wolnych bloków.

Po upływie 180 sekund zostanie uruchomiony wątek konserwacji puli i zostanie przeszuona wolna pula połączeń jms/CF1 . W puli znajduje się wolne połączenie c1 , więc wątek konserwacji sprawdza, kiedy połączenie zostało ponownie umieszczone, i porównuje je z bieżącym czasem.

Upłynęło 180 sekund od umieszczenia połączenia w puli wolnych połączeń, która jest mniejsza niż wartość właściwości **Unused timeout** dla jms/CF1. Dlatego wątek konserwacji pozostawia połączenie w spokoju.

180 sekund później wątek konserwacji puli zostanie ponownie uruchomiony. Wątek konserwacji znajduje połączenie c1i określa, że połączenie znajdowało się w puli przez 360 sekund, która jest dłuższa niż ustawiona wartość **Unused timeout** , więc menedżer połączeń zamyka połączenie.

Jeśli teraz ponownie zostanie uruchomiona metoda sendMessage() , gdy aplikacja wywoła metodę connectionFactory.createConnection() , menedżer połączeń utworzy nowe połączenie z bazą danych IBM MQ , ponieważ wolna pula połączeń dla fabryki połączeń jest pusta.

W poprzednim przykładzie pokazano, w jaki sposób wątek konserwacji używa właściwości **Reap time** i **Unused timeout** , aby zapobiec nieaktualnym połączeniom, gdy właściwość **Aged timeout** ma wartość zero.

Jak działa właściwość **Aged timeout** ?

W poniższym przykładzie założono, że ustawiono:

- Właściwość **Aged timeout** na 300 sekund
- Wartość właściwości **Unused timeout** wynosi zero.

Wywoływana jest metoda sendMessage() , a ta metoda próbuje utworzyć połączenie z fabryki połączeń jms/CF1 .

Ponieważ wolna pula dla tej fabryki jest pusta, menedżer połączeń tworzy nowe połączenie ( c1) i zwraca je do aplikacji. Gdy sendMessage() wywołuje connection.close(), c1 jest umieszczany z powrotem w puli wolnych połączeń.

180 sekund później zostanie uruchomiony wątek konserwacji puli. Wątek znajduje plik c1 w puli wolnych połączeń i sprawdza, jak dawno temu został utworzony. Połączenie istniało przez 180 sekund, czyli mniej niż **Aged timeout**, więc wątek konserwacji puli pozostawia je w stanie uśpienia.

60 sekund później ponownie wywoływana jest funkcja sendMessage() . Tym razem, gdy metoda wywoła metodę connectionFactory.createConnection(), menedżer połączeń wykryje, że w puli wolnych połączeń dla systemu jms/CF1jest dostępne połączenie c1. Menedżer połączeń pobiera produkt c1 z wolnej puli i przekazuje to połączenie do aplikacji.

Połączenie jest zwracane do wolnej puli po wyjściu z programu sendMessage() . 120 sekund później wątek konserwacji puli ponownie się budzi, skanuje zawartość wolnej puli pod kątem jms/CF1 i wykrywa c1.

Mimo że połączenie było używane tylko 120 sekund temu, wątek konserwacji puli zamyka połączenie, ponieważ połączenie istnieje przez łącznie 360 sekund, co przekracza wartość 300 sekund ustawioną dla właściwości **Aged timeout** .

## Wpływ właściwości Minimalna liczba połączeń na wątek konserwacji puli

Korzystając z przykładu “W jaki sposób porty nasłuchiwania MDB używają puli połączeń” na stronie 318 , załóżmy, że na serwerze aplikacji są wdrożone dwie bazy danych MDB, z których każda używa innego portu nasłuchiwania.

Każdy port nasłuchiwania jest skonfigurowany pod kątem używania fabryki połączeń jms/CF1 , która została skonfigurowana z następującymi portami:

- Właściwość **Unused timeout** ustawiona na 120 sekund

- Właściwość **Reap time** ustawiona na 180 sekund
- Właściwość **Minimum connections** ustawiona na 1

Załóżmy, że pierwszy program nasłuchujący jest zatrzymany, a jego połączenie c1 jest umieszczane w puli wolnych bloków. 180 sekund później wątek konserwacji puli aktywuje się, skanuje zawartość wolnej puli pod kątem produktu jms/CF1i wykrywa, że serwer c1 przebywał w wolnej puli przez czas dłuższy niż wartość właściwości **Unused timeout** dla fabryki połączeń.

Jednak przed zamknięciem programu c1wątek konserwacji puli sprawdza, ile połączeń pozostanie w puli, jeśli połączenie zostanie odrzucone. Ponieważ c1 jest jedynym połączeniem w puli wolnych połączeń, menedżer połączeń nie zamyka go, ponieważ spowodowałoby to, że liczba połączeń pozostających w puli wolnych połączeń byłaby mniejsza niż wartość ustawiona dla **Minimum connections**.

Załóżmy, że drugi proces nasłuchujący został zatrzymany. Pula wolnych połączeń zawiera teraz dwa wolne połączenia- c1 i c2.

180 sekund później wątek konserwacji puli zostanie ponownie uruchomiony. W tym czasie program c1 był w puli wolnych połączeń przez 360 sekund, a program c2 przez 180 sekund.

Wątek konserwacji puli sprawdza właściwość c1 i wykrywa, że znajdowała się ona w puli przez czas dłuższy niż wartość właściwości **Unused timeout** .

Następnie wątek sprawdza, ile połączeń znajduje się w wolnej puli, i porównuje to z wartością właściwości **Minimum connections** . Ponieważ pula zawiera dwa połączenia, a parametr **Minimum connections** ma wartość 1, menedżer połączeń zamyka plik c1.

Wątek konserwacji sprawdza teraz plik c2. Wartość ta znajdowała się również w puli wolnych połączeń przez czas dłuższy niż wartość właściwości **Unused timeout** . Jednak ze względu na to, że zamknięcie programu c2 spowodowałoby pozostawienie wolnej puli połączeń z liczbą połączeń mniejszą niż ustawiona minimalna liczba połączeń, menedżer połączeń pozostawi w spokoju parametr c2 .

*JMS i IBM MQ*
Informacje na temat używania produktu IBM MQ jako dostawcy JMS .

## Korzystanie z transportu powiązań

Jeśli fabryka połączeń została skonfigurowana do używania transportu powiązań, każde połączenie produktu JMS nawiązuje konwersację (zwaną również **hconn**) z produktem IBM MQ. Konwersacja używa komunikacji międzyprocesowej (lub pamięci współużytkowanej) do komunikacji z menedżerem kolejek.

## Korzystanie z transportu klienta

Jeśli fabryka połączeń dostawcy przesyłania komunikatów produktu IBM MQ została skonfigurowana pod kątem używania transportu klienta, każde połączenie utworzone z tej fabryki nawiąże nową konwersację (zwaną również**hconn**) z produktem IBM MQ.

W przypadku fabryk połączeń, które łączą się z menedżerem kolejek w trybie normalnym dostawcy usługi przesyłania komunikatów produktu IBM MQ , istnieje możliwość współużytkowania połączenia TCP/IP z produktem IBM MQprzez wiele połączeń produktu JMS utworzonych na podstawie fabryki połączeń. Więcej informacji na ten temat zawiera sekcja “Współużytkowanie połączenia TCP/IP w programie IBM MQ classes for JMS” na stronie 326.

Aby określić maksymalną liczbę kanałów klienta używanych przez połączenia JMS jednocześnie, należy dodać wartość właściwości *Maksymalna liczba połączeń* dla wszystkich fabryk połączeń, które wskazują na ten sam menedżer kolejek.

Załóżmy na przykład, że istnieją dwie fabryki połączeń jms/CF1 i jms/CF2 , które zostały skonfigurowane do nawiązywania połączenia z tym samym menedżerem kolejek produktu IBM MQ przy użyciu tego samego kanału produktu IBM MQ .

Te fabryki używają domyślnych właściwości puli połączeń, co oznacza, że wartość *Maksymalna liczba połączeń* jest ustawiona na 10. Jeśli wszystkie połączenia są używane jednocześnie z obu jms/CF1 i jms/CF2 , będzie 20 konwersacji między serwerem aplikacji i produktem IBM MQ.

Jeśli fabryka połączeń nawiązuje połączenie z menedżerem kolejek przy użyciu trybu normalnego dostawcy usługi przesyłania komunikatów produktu IBM MQ , to maksymalna liczba połączeń TCP/IP między serwerem aplikacji a menedżerem kolejek dla tych fabryk połączeń wynosi:

```
20/the value of SHARECNV for the IBM MQ channel
```

Jeśli fabryka połączeń jest skonfigurowana do nawiązywania połączenia przy użyciu trybu migracji dostawcy przesyłania komunikatów produktu IBM MQ , maksymalna liczba połączeń TCP/IP między serwerem aplikacji a serwerem IBM MQ dla tych fabryk połączeń będzie wynosić 20 (po jednym dla każdego połączenia JMS w pulach połączeń dla dwóch fabryk).

**Pojęcia pokrewne**

“Korzystanie z IBM MQ classes for JMS/Jakarta Messaging” na stronie 84
IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging są dostawcami przesyłania komunikatów produktu Java dostarczanymi wraz z produktem IBM MQ. Oprócz implementowania interfejsów zdefiniowanych w specyfikacjach JMS i Jakarta Messaging dostawcy przesyłania komunikatów dodają dwa zestawy rozszerzeń do interfejsu API przesyłania komunikatów produktu Java .

*Zestawianie obiektów w środowisku Java SE*
W produkcie Java SE (lub innym środowisku, takim jak Spring) modele programistyczne są niezwykle elastyczne. Dlatego też pojedyncza strategia łączenia w pule nie odpowiada wszystkim. Należy rozważyć, czy istnieją ramy, które mogłyby stworzyć jakąkolwiek formę łączenia, na przykład wiosna.

W przeciwnym razie logika aplikacji może to zająć. Zadaj sobie pytanie, jak skomplikowana jest sama aplikacja? Najlepiej zrozumieć, czego wymaga aplikacja i czego wymaga połączenie z systemem przesyłania komunikatów. Aplikacje są często pisane również we własnym kodzie opakowania wokół podstawowego interfejsu API języka JMS .

Chociaż może to być bardzo rozsądne podejście i może ukrywać złożoność, warto pamiętać, że może ono wprowadzać problemy. Na przykład często wywoływana ogólna metoda `getMessage()` nie powinna tylko otwierać i zamykać konsumentów.

Punkty, które należy wziąć pod uwagę:

- Jak długo aplikacja będzie potrzebowała dostępu do IBM MQ? Przez cały czas, lub po prostu okazjonalnie.
- Jak często będą wysyłane wiadomości? Im rzadziej, tym bardziej pojedyncze połączenie z serwerem IBM MQ może być współużytkowane.
- Wyjątek zerwania połączenia jest zwykle oznaką konieczności ponownego utworzenia połączenia w puli. Co z:
  - Wyjątki zabezpieczeń lub host niedostępny
  - Wyjątki zapełnienia kolejki
- Jeśli wystąpi wyjątek zerwania połączenia, co powinno się stać z innymi wolnymipołączeniami w puli? Czy należy je zamknąć i odtworzyć?
- Jeśli na przykład używany jest protokół TLS, jak długo ma być otwarte pojedyncze połączenie?
- W jaki sposób połączenie z puli będzie identyfikować się w taki sposób, że administrator menedżera kolejek będzie mógł rozpoznać połączenie i śledzić je z powrotem.

Wszystkie obiekty JMS powinny być umieszczane w puli i umieszczane w puli, gdy tylko jest to możliwe. Do obiektów należą:

- Połączenia serwera JMS
- Sesja
- Konteksty
- Producenci i konsumenci wszystkich rodzajów

Jeśli używany jest transport klienta, połączenia JMS , sesje i konteksty będą używać gniazd podczas komunikacji z menedżerem kolejek produktu IBM MQ . Dzięki umieszczeniu tych obiektów w puli można

zaoszczędzić na liczbie przychodzących połączeń IBM MQ (hConns) do menedżera kolejek i zmniejszyć liczbę instancji kanału.

Użycie transportu powiązań do menedżera kolejek powoduje całkowite usunięcie warstwy sieciowej. Jednak wiele aplikacji używa transportu klienta, aby zapewnić większą dostępność i równoważenie obciążenia.

Producenci i konsumenci produktu JMS otwierają miejsca docelowe w menedżerze kolejek. Jeśli zostanie otwarta mniejsza liczba kolejek lub tematów, a wiele części aplikacji używa tych obiektów, może to być przydatne podczas łączenia ich w pule.

Z perspektywy programu IBM MQ ten proces zapisuje sekwencję operacji MQOPEN i MQCLOSE.

## Połączenia, sesje i konteksty

Wszystkie te obiekty hermetyzują uchwyty połączeń IBM MQ z menedżerem kolejek i są generowane na podstawie ConnectionFactory. Do aplikacji można dodać logikę ograniczającą liczbę połączeń i inne obiekty utworzone z pojedynczej fabryki połączeń do określonej liczby.

W aplikacji można użyć prostej struktury danych, która będzie zawierać utworzone połączenia. Kod aplikacji, który wymaga użycia jednej z tych struktur danych, może *pobrać* obiekt do użycia.

Należy wziąć pod uwagę następujące czynniki:

- Kiedy należy usunąć połączenia z puli? Ogólnie rzecz biorąc, w połączeniu należy utworzyć obiekt nasłuchiwania wyjątków. Po wywołaniu programu nasłuchującego w celu przetworzenia wyjątku należy ponownie utworzyć połączenie i wszystkie sesje utworzone z tego połączenia.
- Jeśli tabela CCDT jest używana na potrzeby równoważenia obciążenia, połączenia mogą być nawiązywane z różnymi menedżerami kolejek. Może to mieć zastosowanie w przypadku wymagań dotyczących łączenia w pule.

Należy pamiętać, że specyfikacja JMS określa, że błąd programistyczny w przypadku wielu wątków dostępu do sesji lub kontekstu w tym samym czasie. Kod IBM MQ JMS próbuje być rygorystyczny w obsłudze wątków. Należy jednak dodać logikę do aplikacji, aby zapewnić, że sesja lub obiekt kontekstu będzie używany tylko przez jeden wątek naraz.

## Producenci i konsumenci

Każdy utworzony producent i konsument otwiera miejsce docelowe w menedżerze kolejek. Jeśli to samo miejsce docelowe ma być używane do wykonywania różnych zadań, sensowne jest, aby obiekty konsumenta lub producenta były otwarte. Obiekt należy zamknąć tylko wtedy, gdy cała praca zostanie wykonana.

Chociaż otwieranie i zamykanie miejsca docelowego to krótkie operacje, jeśli są wykonywane często, czas może się sumować.

Zasięg tych obiektów znajduje się w obrębie sesji lub kontekstu, z którego zostały utworzone, dlatego muszą być przechowywane w tym zasięgu. Ogólnie rzecz biorąc, aplikacje są napisane w taki sposób, że jest to całkiem proste.

## Monitorowanie

W jaki sposób aplikacje będą monitorować swoje pule obiektów? Odpowiedź na to pytanie zależy w dużej mierze od złożoności wdrożonego rozwiązania w zakresie łączenia w pule.

Jeśli użytkownik rozważa implementację łączenia w pule JavaEE, istnieje duża liczba opcji, w tym:

- Bieżąca wielkość pul
- Czas, jaki obiekty w nich spędziły
- Czyszczenie basenów
- Odświeżanie połączeń

Należy również rozważyć sposób, w jaki w menedżerze kolejek wyświetlana jest jedna ponownie używana sesja. Istnieją właściwości fabryki połączeń umożliwiające identyfikację aplikacji (na przykład appName), które mogą być przydatne.

“Korzystanie z IBM MQ classes for JMS/Jakarta Messaging” na stronie 84
IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging są dostawcami przesyłania komunikatów produktu Java dostarczanymi wraz z produktem IBM MQ. Oprócz implementowania interfejsów zdefiniowanych w specyfikacjach JMS i Jakarta Messaging dostawcy przesyłania komunikatów dodają dwa zestawy rozszerzeń do interfejsu API przesyłania komunikatów produktu Java .

*Współużytkowanie połączenia TCP/IP w programie IBM MQ classes for JMS*
Aby współużytkować pojedyncze połączenie TCP/IP, można utworzyć wiele instancji kanału MQI.

Aplikacje działające w tym samym środowisku wykonawczym Java , które używają adaptera zasobów IBM MQ classes for JMS lub IBM MQ do nawiązywania połączenia z menedżerem kolejek przy użyciu transportu CLIENT, mogą współużytkować instancję kanału.

Jeśli kanał jest zdefiniowany z parametrem **SHARECNV** ustawionym na wartość większą niż 1, to ta liczba konwersacji może współużytkować instancję kanału. Aby umożliwić użycie tej funkcji przez fabrykę połączeń lub specyfikację aktywowania, należy ustawić właściwość **SHARECONVALLOWED** na wartość YES.

Każde połączenie JMS i każda sesja JMS utworzone przez aplikację JMS tworzy własną konwersację z menedżerem kolejek.

Po uruchomieniu specyfikacji aktywowania adapter zasobów IBM MQ rozpoczyna konwersację z menedżerem kolejek, która ma być używana przez specyfikację aktywowania. Każda sesja serwera w puli sesji serwera, która jest powiązana ze specyfikacją aktywowania, również rozpoczyna konwersację z menedżerem kolejek.

Atrybut **SHARECNV** to metoda optymalnie dołożona do współużytkowania połączeń. Dlatego, jeśli dla parametru IBM MQ classes for JMSzostanie użyta wartość **SHARECNV** większa niż 0, nie ma gwarancji, że nowe żądanie połączenia będzie zawsze współużytkowane z już nawiązanym połączeniem.

## Sposób współużytkowania połączeń TCP/IP

Dostępne są dwie strategie współużytkowania połączeń TCP/IP:

**Strategia GLOBAL**
Jest to domyślna strategia współużytkowania połączeń TCP/IP. Każde połączenie lub sesja JMS może korzystać z konwersacji na dowolnym odpowiednim połączeniu TCP/IP. Przydatność jest określana na podstawie takich czynników, jak adres hosta, numer portu, ID użytkownika i hasło oraz parametry TLS/SSL.

Takie podejście do współużytkowania połączeń TCP/IP minimalizuje liczbę używanych instancji kanałów, ale kosztem rywalizacji o dostęp do globalnej puli połączeń TCP/IP.

**Strategia CONNECTION**
W przypadku tej strategii instancje kanału są współużytkowane tylko między powiązanymi obiektami JMS . W szczególności po utworzeniu połączenia JMS tworzona jest dla niego instancja kanału, a dodatkowe konwersacje w tej instancji kanału są dostępne tylko dla sesji JMS , które są tworzone przez to połączenie JMS .

Jeśli zostanie utworzonych więcej konwersacji niż określono w atrybucie SHARECNV, tworzona jest nowa instancja kanału, która może być używana tylko przez sesje JMS utworzone przez oryginalne połączenie JMS .

Takie podejście do współużytkowania instancji kanału zmniejsza rywalizację o konwersacje, ponieważ potencjalnie może wymagać znacznie większej liczby instancji kanału.

## Jawne określanie strategii współużytkowania instancji kanału

V 9.3.2

Domyślnie strategia GLOBAL jest używana, jeśli nie można ponownie nawiązać połączenia z aplikacjami. Aplikacje, które można ponownie nawiązać połączenie, zawsze używają strategii CONNECTION.

W przypadku aplikacji korzystających z systemu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messagingstrategię CONNECTION można włączyć dla aplikacji bez możliwości ponownego połączenia w całej aplikacji. Strategię CONNECTION można włączyć, ustawiając właściwość systemową `com.ibm.mq.jms.channel.sharing` na wartość `CONNECTION`. W tej wartości nie jest rozróżniana wielkość liter, a każda wartość inna niż `CONNECTION` jest ignorowana.

Właściwość systemową `com.ibm.mq.jms.channel.sharing` można ustawić na jeden z następujących sposobów:

- Ustaw tę właściwość w ramach inicjowania maszyny JVM przy użyciu opcji wiersza komend -D:

```
-Dcom.ibm.mq.jms.channel.sharing=CONNECTION
```

- Tę właściwość należy ustawić przed użyciem właściwości IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging za pomocą komendy `System.setProperty()` .

## Obliczanie liczby instancji kanału dla strategii współużytkowania globalnego

Aby określić maksymalną liczbę instancji kanału, które są tworzone przez aplikację, należy użyć następujących wzorów:

**Specyfikacje aktywowania**

Liczba instancji kanału = (*maxPoolDepth_value* + 1)/ *SHARECNV_value*

Gdzie *maxPoolDepth_value* jest wartością właściwości **maxPoolDepth** , a *SHARECNV_value* jest wartością właściwości **SHARECNV** w kanale używanym przez specyfikację aktywowania.

**Inne aplikacje JMS**

Liczba instancji kanału = (*jms_connections* + *jms_sessions*)/ *SHARECNV_value*

Gdzie *jms_connections* to liczba połączeń utworzonych przez aplikację, *jms_sessions* to liczba sesji JMS utworzonych przez aplikację, a *SHARECNV_value* to wartość właściwości **SHARECNV** w kanale używanym przez specyfikację aktywowania.

## Obliczanie liczby instancji kanału dla strategii współużytkowania CONNECTION

Liczba instancji kanału zależy od rozkładu sesji JMS między połączenia JMS w aplikacji.

Zezwól na jedną konwersację dla połączenia JMS i jedną konwersację dla każdej sesji JMS w ramach tego połączenia JMS , a następnie podziel ją przez wartość **SHARECNV** , zaokrąglając w górę. To obliczenie przedstawia instancje kanału, które są wymagane przez to połączenie JMS .

Tę samą zasadę można zastosować do specyfikacji aktywowania. Specyfikację aktywowania należy traktować jako połączenie JMS , a właściwość **maxPoolDepth** jako liczbę sesji JMS .

## Przykłady

W poniższych przykładach przedstawiono sposób użycia formuł do obliczenia liczby instancji kanału, które są tworzone w menedżerze kolejek przez aplikacje przy użyciu adaptera zasobów produktu IBM MQ classes for JMS lub IBM MQ .

**JMS przykład zastosowania**
Połączenie aplikacji JMS łączy się z menedżerem kolejek przy użyciu transportu CLIENT i tworzy JMS połączenie i trzy JMS sesje. Kanał używany przez aplikację do nawiązywania połączenia z menedżerem kolejek ma właściwość **SHARECNV** ustawioną na wartość 10. Gdy aplikacja jest uruchomiona, istnieją cztery konwersacje między aplikacją i menedżerem kolejek oraz jedna instancja kanału. Wszystkie cztery konwersacje współużytkują instancję kanału.

**Przykład specyfikacji aktywowania**
Specyfikacja aktywowania nawiązuje połączenie z menedżerem kolejek przy użyciu transportu CLIENT. Specyfikacja aktywowania jest skonfigurowana z właściwością **maxPoolDepth** ustawioną na wartość 10. Kanał, który jest używany przez specyfikację aktywowania, ma właściwość **SHARECNV**

ustawioną na wartość 10. Jeśli specyfikacja aktywowania jest uruchomiona i jednocześnie przetwarza 10 komunikatów, liczba konwersacji między specyfikacją aktywowania a menedżerem kolejek wynosi 11 (10 konwersacji dla sesji serwera i 1 dla specyfikacji aktywowania). Liczba instancji kanału używanych przez specyfikację aktywowania wynosi 2.

**Przykład specyfikacji aktywowania**

Specyfikacja aktywowania nawiązuje połączenie z menedżerem kolejek przy użyciu transportu CLIENT. Specyfikacja aktywowania jest skonfigurowana z właściwością **maxPoolDepth** ustawioną na wartość 5. Kanał, który ma być używany przez specyfikację aktywowania, ma właściwość **SHARECNV** ustawioną na wartość 0. Jeśli specyfikacja aktywowania jest uruchomiona i jednocześnie przetwarza 5 komunikatów, liczba konwersacji między specyfikacją aktywowania a menedżerem kolejek wynosi 6 (pięć konwersacji dla sesji serwera i jedna dla specyfikacji aktywowania). Liczba instancji kanału używanych przez specyfikację aktywowania wynosi 6, ponieważ właściwość **SHARECNV** w kanale ma wartość 0. Każda konwersacja używa własnej instancji kanału.

**Zadania pokrewne**

“Określanie liczby połączeń TCP/IP tworzonych z zakresu od WebSphere Application Server do IBM MQ” na stronie 520

Korzystając z funkcji współużytkowania konwersacji, wiele konwersacji może współużytkować instancje kanału MQI, co jest również nazywane połączeniem TCP/IP.

*Określanie zakresu portów dla połączeń klienckich w produkcie IBM MQ classes for JMS*

Użyj właściwości LOCALADDRESS, aby określić zakres portów, z którymi aplikacja może być powiązana.

Gdy aplikacja IBM MQ classes for JMS próbuje połączyć się z menedżerem kolejek IBM MQ w trybie klienta, firewall może zezwalać tylko na połączenia pochodzące z określonych portów lub z określonego zakresu portów. W takiej sytuacji można użyć właściwości LOCALADDRESS obiektu fabryki ConnectionFactory, obiektu fabryki QueueConnectionlub obiektu fabryki TopicConnection, aby określić port lub zakres portów, z którymi może zostać powiązana aplikacja.

Właściwość LOCALADDRESS można ustawić za pomocą narzędzia administracyjnego IBM MQ JMS lub wywołując metodę setLocalAddress () w aplikacji JMS . Poniżej przedstawiono przykład ustawiania właściwości z poziomu aplikacji:

```
mqConnectionFactory.setLocalAddress("192.0.2.0(2000,3000)");
```

Gdy aplikacja łączy się później z menedżerem kolejek, zostaje powiązana z lokalnym adresem IP i numerem portu z zakresu od 192.0.2.0(2000) do 192.0.2.0(3000).

W systemie z więcej niż jednym interfejsem sieciowym można również użyć właściwości LOCALADDRESS, aby określić, który interfejs sieciowy ma być używany dla połączenia.

W przypadku połączenia w czasie rzeczywistym z brokerem właściwość LOCALADDRESS ma zastosowanie tylko wtedy, gdy używane jest rozsyłanie grupowe. W takim przypadku można użyć tej właściwości, aby określić, który interfejs sieci lokalnej musi być używany dla połączenia, ale wartość właściwości nie może zawierać numeru portu ani zakresu numerów portów.

Jeśli zakres portów zostanie ograniczony, mogą wystąpić błędy połączenia. Jeśli wystąpi błąd, zgłaszany jest wyjątek JMSException z osadzonym wyjątkiem MQException, który zawiera kod przyczyny IBM MQ MQRC_Q_MGR_NOT_AVAILABLE i następujący komunikat:

```
Odmowa połączenia z gniazdem z powodu ograniczeń LOCAL_ADDRESS_PROPERTY
```

Błąd może wystąpić, jeśli wszystkie porty z podanego zakresu są używane lub jeśli podany adres IP, nazwa hosta lub numer portu nie są poprawne (na przykład ujemny numer portu).

Ponieważ produkt IBM MQ classes for JMS może tworzyć połączenia inne niż wymagane przez aplikację, należy zawsze rozważyć określenie zakresu portów. Na ogół każda sesja utworzona przez aplikację wymaga jednego portu, a program IBM MQ classes for JMS może wymagać trzech lub czterech dodatkowych portów. Jeśli wystąpi błąd połączenia, zwiększ zakres portów.

Zestawianie połączeń, które jest domyślnie używane w produkcie IBM MQ classes for JMS, może mieć wpływ na szybkość, z jaką porty mogą być ponownie wykorzystywane. W wyniku tego podczas zwalniania portów może wystąpić błąd połączenia.

*Kompresja kanału w programie IBM MQ classes for JMS*
Aplikacja IBM MQ classes for JMS może używać narzędzi IBM MQ do kompresowania nagłówka lub danych komunikatu.

Kompresja danych przepływających przez kanał IBM MQ może zwiększyć wydajność kanału i zmniejszyć ruch w sieci. Funkcja dostarczana z produktem IBM MQumożliwia kompresowanie danych przepływających przez kanały komunikatów i kanały MQI. W każdym typie kanału można kompresować dane nagłówka i komunikatu niezależnie od siebie. Domyślnie w kanale nie są kompresowane żadne dane.

Aplikacja IBM MQ classes for JMS określa techniki, które mogą być używane do kompresowania danych nagłówka lub komunikatu w połączeniu przez utworzenie obiektu java.util.Collection . Każda technika kompresji jest obiektem typu Integer w kolekcji, a kolejność, w jakiej aplikacja dodaje techniki kompresji do kolekcji, określa kolejność, w jakiej techniki kompresji są negocjowane z menedżerem kolejek podczas tworzenia połączenia przez aplikację. Aplikacja może następnie przekazać kolekcję do obiektu ConnectionFactory , wywołując metodę setHdrCompList() dla danych nagłówka lub metodę setMsgCompList() dla danych komunikatu. Gdy aplikacja jest gotowa, może utworzyć połączenie.

Poniższe fragmenty kodu ilustrują opisane podejście. Pierwszy fragment kodu przedstawia sposób implementowania kompresji danych nagłówka:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(WMQConstants.WMQ_COMPHDR_SYSTEM));
.
.
.
((MQConnectionFactory) cf).setHdrCompList(headerComp);
.
.
.
connection = cf.createConnection();
```

Drugi fragment kodu przedstawia sposób implementowania kompresji danych komunikatu:

```
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();
```

W drugim przykładzie techniki kompresji są negocjowane w kolejności RLE, a następnie ZLIBHIGH, gdy połączenie jest tworzone. Wybrana technika kompresji nie może być zmieniona w czasie życia obiektu połączenia. Aby użyć kompresji w połączeniu, przed utworzeniem obiektu połączenia należy wywołać metody setHdrCompList() i setMsgCompList().

*Asynchroniczne umieszczanie komunikatów w produkcie IBM MQ classes for JMS*
Zwykle, gdy aplikacja wysyła komunikaty do miejsca docelowego, musi czekać, aż menedżer kolejek potwierdzi przetworzenie żądania. W pewnych okolicznościach można zwiększyć wydajność przesyłania komunikatów, wybierając opcję asynchronicznego umieszczania komunikatów. Jeśli aplikacja umieszcza komunikat asynchronicznie, menedżer kolejek nie zwraca informacji o powodzeniu lub niepowodzeniu każdego wywołania, ale można sprawdzać okresowo występowanie błędów.

To, czy miejsce docelowe zwraca sterowanie do aplikacji bez określania, czy menedżer kolejek otrzymał komunikat bezpiecznie, zależy od następujących właściwości:

**Właściwość docelowa JMS PUTASYNCALLOWED (nazwa skrócona-PAALD).**
Parametr PUTASYNCALLOWED określa, czy aplikacje JMS mogą umieszczać komunikaty asynchronicznie, jeśli ta opcja jest dozwolona przez bazową kolejkę lub temat reprezentowany przez miejsce docelowe JMS .

**Właściwość DEFPRESP kolejki lub tematu produktu IBM MQ (domyślny typ odpowiedzi umieszczania).**
DEFPRESP określa, czy aplikacje, które umieszczają komunikaty w kolejce lub publikują komunikaty w temacie, mogą korzystać z funkcji asynchronicznego umieszczania.

W poniższej tabeli przedstawiono możliwe wartości właściwości PUTASYNCALLOWED i DEFPRESP oraz kombinacje wartości używane do włączenia funkcji asynchronicznego umieszczania:

*Tabela 46. Sposób łączenia właściwości PUTASYNCALLOWED i DEFPRESP w celu określenia, czy komunikaty są umieszczane w miejscu docelowym asynchronicznie.*

| Właściwość kolejki IBM MQ | PUTASYNCALLOWED = NIE | PUTASYNCALLOWED = TAK | Włączona funkcja asynchronicznego umieszczania |
|---|---|---|---|
| DEFPRESP=SYNC | Funkcja asynchronicznego umieszczania nie jest włączona | Włączona funkcja asynchronicznego umieszczania | PUTASYNCALLOWED = AS_DEST lub AS_Q_DEF lub AS_T_DEF |
| DEFPRESP=ASYNC | Funkcja asynchronicznego umieszczania nie jest włączona | Włączona funkcja asynchronicznego umieszczania | PUTASYNCALLOWED = AS_DEST lub AS_Q_DEF lub AS_T_DEF |

Zachowanie to można zmienić, określając właściwość IBM MQ-JMS Destination na wartość "NO" lub "YES", jak pokazano w tabeli, ale można ją również przesłonić dla całej wirtualnej maszyny języka Java przy użyciu maszyny JVM **SystemProperty** i wartości:

```
com.ibm.mq.cfg.Channels.Put1DefaultAlwaysSync=Y
```

W przypadku komunikatów wysłanych w sesji z transakcją aplikacja ostatecznie określa, czy menedżer kolejek otrzymał komunikaty bezpiecznie podczas wywoływania funkcji `commit()`.

Jeśli aplikacja wysyła komunikaty trwałe w ramach sesji transakcyjnej, a co najmniej jeden komunikat nie zostanie bezpiecznie odebrany, transakcja nie zostanie zatwierdzona i zostanie zgłoszony wyjątek. Jeśli jednak aplikacja wysyła komunikaty nietrwałe w ramach sesji transakcyjnej, a co najmniej jeden komunikat nie zostanie bezpiecznie odebrany, transakcja zostanie pomyślnie zatwierdzona. Aplikacja nie otrzymuje żadnych informacji zwrotnych o tym, że komunikaty nietrwałe nie dotarły bezpiecznie.

W przypadku nietrwałych komunikatów wysyłanych w sesji, która nie jest transakcją, właściwość SENDCHECKCOUNT obiektu *ConnectionFactory* określa liczbę komunikatów, które mają zostać wysłane, zanim program IBM MQ classes for JMS sprawdzi, czy menedżer kolejek otrzymał komunikaty w sposób bezpieczny.

Jeśli podczas sprawdzania zostanie wykryte, że co najmniej jeden komunikat nie został bezpiecznie odebrany, a aplikacja zarejestrowała w połączeniu obiekt nasłuchiwania wyjątków, metoda IBM MQ classes for JMS wywołuje metodę `onException()` obiektu nasłuchiwania wyjątków w celu przekazania do aplikacji wyjątku JMS .

Wyjątek JMS ma kod błędu JMSWMQ0028 i ten kod wyświetla następujący komunikat:

```
At least one asynchronous put message failed or gave a warning.
```

Wyjątek JMS zawiera również powiązany wyjątek, który udostępnia więcej szczegółów. Wartością domyślną właściwości SENDCHECKCOUNT jest zero, co oznacza, że takie sprawdzenia nie są wykonywane.

Ta optymalizacja jest najbardziej korzystna dla aplikacji, która nawiązuje połączenie z menedżerem kolejek w trybie klienta i musi wysłać sekwencję komunikatów w krótkim odstępie czasu, ale nie wymaga

natychmiastowej reakcji od menedżera kolejek dla każdego wysłanego komunikatu. Jednak aplikacja może nadal korzystać z tej optymalizacji, nawet jeśli nawiązuje połączenie z menedżerem kolejek w trybie powiązań, ale oczekiwana wydajność nie jest tak duża.

**Uwaga:** Jeśli do wysłania komunikatu w ramach transakcji używany jest niezidentyfikowany element **MessageProducer** , to domyślnie komunikaty są umieszczane w kolejce przy użyciu mechanizmu asynchronicznego umieszczania.

Może to być spowodowane tym, że interfejs API JMS umożliwia utworzenie pliku **MessageProducer** bez określania miejsca docelowego przy użyciu następującej składni:

```
javax.jms.MessageProducer messageProducer =  javax.jms.Session.createProducer(null);
  messageProducer.send(Destination destination, Message message, int deliveryMode, int priority, long
timeToLive);
```

W tym scenariuszu miejsce docelowe JMS jest udostępniane wtedy, gdy komunikat jest wysyłany, a nie przed czasem, gdy tworzony jest **MessageProducer** . W przypadku funkcji API języka IBM MQ powoduje to wysłanie komunikatu MQPUT1 w celu umieszczenia go w kolejce.

Jeśli zostanie to wykonane w punkcie synchronizacji systemu IBM MQ , co oznacza (w terminologii JMS ), że komunikat jest umieszczany w transakcji przy użyciu sesji JMS z transakcją lub przy użyciu interfejsu API języka IBM MQ classes for JMS ( XASession ), nastąpi przełączenie na asynchroniczne umieszczanie.

*Korzystanie z odczytu z wyprzedzeniem w produkcie IBM MQ classes for JMS*
Funkcja odczytu z wyprzedzeniem, która jest udostępniana przez produkt IBM MQ , umożliwia wysyłanie nietrwałych komunikatów, które są odbierane poza transakcją, do IBM MQ classes for JMS przed zażądaniem ich przez aplikację. Program IBM MQ classes for JMS zapisuje komunikaty w wewnętrznym buforze i przekazuje je do aplikacji, gdy żąda ich aplikacja.

Aplikacje IBM MQ classes for JMS , które używają produktu MessageConsumers lub MessageListeners do odbierania komunikatów z miejsca docelowego poza transakcją, mogą korzystać z funkcji odczytu z wyprzedzeniem. Korzystanie z odczytu z wyprzedzeniem pozwala aplikacjom, które używają tych obiektów, na zwiększenie wydajności podczas odbierania komunikatów.

To, czy aplikacja używająca języka MessageConsumers lub MessageListeners może korzystać z odczytu z wyprzedzeniem, zależy od następujących właściwości:

**Właściwość docelowa JMS READAHEADALLOWED (krótka nazwa-RAALD).**
Parametr READAHEADALLOWED określa, czy aplikacje JMS mogą używać odczytu z wyprzedzeniem podczas pobierania lub przeglądania nietrwałych komunikatów poza transakcją, jeśli ta opcja jest dozwolona w bazowej kolejce lub temacie reprezentowanych przez miejsce docelowe JMS .

**Kolejka IBM MQ lub właściwość tematu DEFREADA (domyślny odczyt z wyprzedzeniem).**
DEFREADA określa, czy aplikacje, które odbierają lub przeglądają nietrwałe komunikaty poza transakcją, mogą korzystać z odczytu z wyprzedzeniem.

W poniższej tabeli przedstawiono możliwe wartości właściwości READAHEADALLOWED i DEFREADA oraz kombinacje wartości używane do włączenia funkcji odczytu z wyprzedzeniem:

*Tabela 47. Sposób łączenia właściwości READAHEADALLOWED i DEFREADA w celu określenia, czy odczyt z wyprzedzeniem jest używany podczas odbierania lub przeglądania nietrwałych komunikatów poza transakcją.*

| Właściwość kolejki IBM MQ | READAHEADALLOWED = TAK | READAHEADALLOWED = NIE | AS_DEST lub AS_Q_DEF lub AS_T_DEF |
|---|---|---|---|
| DEFREADA = NIE | Włączona funkcja odczytu z wyprzedzeniem | Funkcja odczytu z wyprzedzeniem nie jest włączona | Funkcja odczytu z wyprzedzeniem nie jest włączona |
| DEFREADA = TAK | Włączona funkcja odczytu z wyprzedzeniem | Funkcja odczytu z wyprzedzeniem nie jest włączona | Włączona funkcja odczytu z wyprzedzeniem |

*Tabela 47. Sposób łączenia właściwości READAHEADALLOWED i DEFREADA w celu określenia, czy odczyt z wyprzedzeniem jest używany podczas odbierania lub przeglądania nietrwałych komunikatów poza transakcją. (kontynuacja)*

| **Właściwość kolejki IBM MQ** | **READAHEADALLOWED = TAK** | **READAHEADALLOWED = NIE** | **AS_DEST lub AS_Q_DEF lub AS_T_DEF** |
|---|---|---|---|
| DEFREADA = WYŁĄCZONE | Funkcja odczytu z wyprzedzeniem nie jest włączona | Funkcja odczytu z wyprzedzeniem nie jest włączona | Funkcja odczytu z wyprzedzeniem nie jest włączona |

Jeśli funkcja odczytu z wyprzedzeniem jest włączona, gdy aplikacja tworzy `MessageConsumer` lub `MessageListener` , IBM MQ classes for JMS tworzy wewnętrzny bufor dla miejsca docelowego monitorowanego przez `MessageConsumer` lub `MessageListener` . Dla każdego systemu `MessageConsumer` lub `MessageListener`istnieje jeden bufor wewnętrzny. Menedżer kolejek rozpoczyna wysyłanie nietrwałych komunikatów do konsoli IBM MQ classes for JMS , gdy aplikacja wywołuje jedną z następujących metod:

- `MessageConsumer.receive()`
- `MessageConsumer.receive(long timeout)`
- `MessageConsumer.receiveNoWait()`
- `Session.setMessageListener(MessageListener listener)`

Produkt IBM MQ classes for JMS automatycznie zwraca pierwszy komunikat z powrotem do aplikacji przy użyciu wywołania metody, które zostało wykonane przez aplikację. Inne nietrwałe komunikaty są zapisywane przez IBM MQ classes for JMS w wewnętrznym buforze, który został utworzony dla miejsca docelowego. Gdy aplikacja zażąda przetworzenia następnego komunikatu, IBM MQ classes for JMS zwróci następny komunikat w buforze wewnętrznym.

Program IBM MQ classes for JMS żąda od menedżera kolejek większej liczby nietrwałych komunikatów, gdy bufor wewnętrzny jest pusty.

Bufor wewnętrzny używany przez IBM MQ classes for JMS jest usuwany, gdy aplikacja zamyka `MessageConsumer`lub sesję JMS , z którą powiązany jest `MessageListener` .

W przypadku systemu `MessageConsumers`wszystkie nieprzetworzone komunikaty w buforze wewnętrznym są tracone.

Jeśli używana jest wartość `MessageListeners`, to, co dzieje się z komunikatami w buforze wewnętrznym, zależy od właściwości docelowej JMS READAHEADCLOSEPOLICY (krótka nazwa-RACP). Wartością domyślną tej właściwości jest DELIVER_ALL, co oznacza, że sesja produktu JMS , która została użyta do utworzenia pliku `MessageListener` , nie zostanie zamknięta, dopóki wszystkie komunikaty w buforze wewnętrznym nie zostaną dostarczone do aplikacji. Jeśli ta właściwość ma wartość DELIVER_CURRENT, sesja JMS zostanie zamknięta po przetworzeniu bieżącego komunikatu przez aplikację i wszystkie pozostałe komunikaty w buforze wewnętrznym zostaną usunięte.

*Zachowane publikacje w produkcie IBM MQ classes for JMS*
Klient IBM MQ classes for JMS można skonfigurować w taki sposób, aby używał zachowanych publikacji.

Publikator może określić, że kopia publikacji musi zostać zachowana, aby mogła zostać wysłana do przyszłych subskrybentów, którzy zarejestrują zainteresowanie tematem. W tym celu w programie IBM MQ classes for JMS należy ustawić wartość 1 dla właściwości JMS_IBM_RETAIN będącej liczbą całkowitą. Stałe zostały zdefiniowane dla tych wartości w interfejsie com.ibm.msg.client.jms.JmsConstants . Jeśli na przykład utworzono komunikat *msg*, aby ustawić go jako zachowaną publikację, należy użyć następującego kodu:

```
// set as a retained publication
msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION);
```

Teraz można wysłać wiadomość w normalny sposób. Zapytanie dotyczące JMS_IBM_RETAIN można również wysłać w odebranym komunikacie. Dlatego można sprawdzić, czy odebrany komunikat jest zachowaną publikacją.

### *Obsługa interfejsu XA w produkcie IBM MQ classes for JMS*

Produkt JMS obsługuje transakcje zgodne z interfejsem XA w powiązaniach i trybach klienta z obsługiwanym menedżerem transakcji w kontenerze JEE .

Jeśli w środowisku serwera aplikacji wymagana jest funkcjonalność interfejsu XA, należy odpowiednio skonfigurować aplikację. Informacje na temat konfigurowania aplikacji do korzystania z transakcji rozproszonych można znaleźć w dokumentacji serwera aplikacji.

Menedżer kolejek systemu IBM MQ nie może działać jako menedżer transakcji dla systemu JMS.

### *Opóźnienie dostarczania komunikatów JMS*

W przypadku systemu JMS 2.0 lub nowszego można określić opóźnienie dostarczania podczas wysyłania komunikatu. Menedżer kolejek nie dostarcza komunikatu, dopóki nie upłynie określone opóźnienie dostarczenia.

Aplikacja może określić opóźnienie dostawy (w milisekundach) podczas wysyłania komunikatu za pomocą opcji `MessageProducer.setDeliveryDelay(long deliveryDelay)` lub `JMSProducer.setDeliveryDelay(long deliveryDelay)`. Ta wartość jest dodawana do czasu wysłania komunikatu i podaje najwcześniejszy czas, w którym każda inna aplikacja może uzyskać ten komunikat.

Opóźnienie dostarczenia jest implementowane przy użyciu pojedynczej wewnętrznej kolejki przemieszczania. Komunikaty z niezerowym opóźnieniem dostarczenia są umieszczane w tej kolejce z nagłówkiem, który wskazuje opóźnienie dostarczenia i informacje o kolejce docelowej. Komponent menedżera kolejek nazywany procesorem opóźnienia dostarczania monitoruje komunikaty w kolejce przemieszczania. Po zakończeniu opóźnienia dostarczenia komunikatu komunikat jest pobierany z kolejki przemieszczania i umieszczany w kolejce docelowej.

## Klienty przesyłania komunikatów

Implementacja opóźnienia dostarczania w systemie IBM MQ jest dostępna do użycia tylko wtedy, gdy używany jest klient JMS . W przypadku korzystania z opóźnienia dostarczania z produktem IBM MQobowiązują następujące ograniczenia. Te ograniczenia dotyczą w równym stopniu produktów `MessageProducers` i `JMSProducers`, ale wyjątek `JMSRuntimeExceptions` jest zgłaszany w przypadku produktu `JMSProducers`.

- Każda próba wywołania funkcji `MessageProducer.setDeliveryDelay` z wartością niezerową po nawiązaniu połączenia z menedżerem kolejek w wersji wcześniejszej niż IBM MQ 8.0powoduje wywołanie funkcji `JMSException` z komunikatem MQRC_FUNCTION_NOT_SUPPORTED.
- Opóźnienie dostarczenia nie jest obsługiwane w przypadku klastrowych miejsc docelowych, które mają wartość **DEFBIND** inną niż MQBND_BIND_NOT_FIXED. Jeśli parametr `MessageProducer` ma ustawione niezerowe opóźnienie dostarczenia i podejmowana jest próba wysłania do miejsca docelowego, które nie spełnia tego wymagania, wywołanie zwraca błąd `JMSException` z komunikatem MQRC_OPTIONS_ERROR.
- Każda próba ustawienia wartości czasu życia, która jest mniejsza niż poprzednio określone niezerowe opóźnienie dostarczania lub odwrotnie, powoduje wygenerowanie komunikatu `JMSException` z komunikatem MQRC_EXPIRY_ERROR. To sprawdzanie jest wykonywane przy wywoływaniu metod `setTimeToLive` , `setDeliveryDelay` lub `send` , w zależności od wybranego zestawu operacji.
- Użycie zachowanych publikacji i opóźnienia dostarczania nie jest obsługiwane. Próba opublikowania komunikatu z opóźnieniem dostarczenia, jeśli ten komunikat został oznaczony jako zachowany przy użyciu parametru `msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION)` , spowoduje wyświetlenie komunikatu `JMSException` z komunikatem MQRC_OPTIONS_ERROR.

- Opóźnienie dostarczenia i grupowanie komunikatów nie są obsługiwane, a każda próba użycia tej kombinacji powoduje wyświetlenie komunikatu `JMSException` z komunikatem MQRC_OPTIONS_ERROR.

Każde niepowodzenie wysłania komunikatu z opóźnieniem dostarczenia powoduje, że klient zgłasza wyjątek `JMSException` z odpowiednim komunikatem o błędzie, na przykład pełną kolejką. W niektórych sytuacjach komunikat o błędzie może dotyczyć miejsca docelowego, kolejki pomostowej lub obu tych elementów.

**Uwaga:** Produkt IBM MQ umożliwia aplikacjom, które umieściły komunikat w jednostce pracy, ponowne pobranie tego samego komunikatu, nawet jeśli jednostka pracy nie została zatwierdzona. Ta technika nie działa z opóźnieniem dostarczenia, ponieważ komunikat nie jest umieszczany w kolejce przemieszczania do momentu zatwierdzenia jednostki pracy i w rezultacie nie zostanie wysłany do miejsca docelowego.

## Autoryzacja

Produkt IBM MQ przeprowadza sprawdzanie autoryzacji w oryginalnym miejscu docelowym, gdy aplikacja wysyła komunikat z niezerowym opóźnieniem dostarczenia. Jeśli aplikacja nie jest autoryzowana, wysyłanie nie powiedzie się. Gdy menedżer kolejek wykryje, że opóźnienie dostarczenia komunikatu zostało zakończone, otwiera kolejkę docelową. W tym momencie nie są przeprowadzane żadne kontrole autoryzacji.

## SYSTEM.DDELAY.LOCAL.QUEUE

Kolejka systemowa SYSTEM.DDELAY.LOCAL.QUEUE: służy do implementowania opóźnienia dostarczania.

- Multi W systemie Wiele platform: SYSTEM.DDELAY.LOCAL.QUEUE domyślnie istnieje. Należy zmienić kolejkę systemową, tak aby jej atrybuty MAXMSGL i MAXDEPTH były wystarczające dla oczekiwanego obciążenia.
- z/OS W systemie IBM MQ for z/OS: SYSTEM.DDELAY.LOCAL.QUEUE jest używana jako kolejka pomostowa dla komunikatów wysyłanych z opóźnieniem dostarczania zarówno do kolejek lokalnych, jak i współużytkowanych. W systemie z/OSnależy utworzyć i zdefiniować kolejkę, aby jej atrybuty MAXMSGL i MAXDEPTH były wystarczające dla oczekiwanego obciążenia.

Po utworzeniu tej kolejki musi być ona zabezpieczona tak, aby dostęp do niej miał jak najmniej użytkowników. Dostęp do kolejki musi być wyłącznie na potrzeby konserwacji i monitorowania.

Jeśli komunikat jest wysyłany przez aplikację JMS z niezerowym opóźnieniem dostarczenia, jest on umieszczany w tej kolejce z nowym identyfikatorem komunikatu. Oryginalny identyfikator komunikatu jest umieszczany w identyfikatorze korelacji komunikatu. Ten identyfikator korelacji umożliwia aplikacji pobranie komunikatu z kolejki pomostowej, gdy jest to wymagane, na przykład jeśli przez pomyłkę użyto dużego opóźnienia dostawy.

## Uwagi dotyczące produktu z/OS

z/OS

Jeśli system działa w systemie z/OS, należy wziąć pod uwagę dodatkowe zagadnienia, które należy wziąć pod uwagę, jeśli planowane jest użycie opóźnienia dostawy.

Jeśli ma być używane opóźnienie dostarczania, kolejka systemowa SYSTEM.DDELAY.LOCAL.QUEUE musi być zdefiniowana. Musi być ona zdefiniowana z klasą pamięci masowej, która jest wystarczająca dla jej oczekiwanego ładowania oraz z określonymi parametrami INDXTYPE (NONE) i MSGDLVSQ (FIFO). Przykładowa definicja kolejki systemowej jest przekształcona w komentarz w zadaniu JCL CSQ4INSG .

## Kolejki współużytkowane

Opóźnienie dostarczania jest obsługiwane w przypadku wysyłania komunikatów do kolejek współużytkowanych. Jednak istnieje tylko jedna prywatna kolejka pomostowa, która jest używana niezależnie od tego, czy kolejka docelowa jest współużytkowana, czy nie. Menedżer kolejek, do którego

należy ta kolejka prywatna, musi być uruchomiony, aby po zakończeniu opóźnienia możliwe było wysłanie opóźnionego komunikatu do docelowej kolejki współużytkowanej.

**Uwaga:** Jeśli komunikat nietrwały zostanie umieszczony w kolejce współużytkowanej z opóźnieniem dostarczania, a menedżer kolejek, do którego należy kolejka pomostowa, zostanie zamknięty, oryginalny komunikat zostanie utracony. W wyniku tego nietrwałe komunikaty wysłane z opóźnieniem dostarczenia do kolejki współużytkowanej mogą zostać utracone bardziej niż nietrwałe komunikaty wysłane bez opóźnienia dostarczenia do kolejki współużytkowanej.

## Docelowe rozstrzygnięcie

Jeśli komunikat jest wysyłany do kolejki, rozstrzygnięcie jest sterowane dwa razy: raz przez aplikację JMS i raz przez menedżer kolejek, gdy komunikat jest zdejmiany z kolejki pomostowej i wysyłany do kolejki docelowej.

Subskrypcje docelowe dla publikacji są uzgadniane, gdy aplikacja JMS wywołuje metodę send.

Jeśli komunikat jest wysyłany z trwałością lub priorytetem zgodnie z definicją kolejki, wartość jest ustawiana w pierwszym rozstrzygnięcie, a nie w drugim.

## Interwał utraty ważności

Opóźnienie dostarczania zachowuje zachowanie właściwości utraty ważności **MQMD.Expiry**. Jeśli na przykład komunikat został wysłany z aplikacji JMS z okresem ważności wynoszącym 20 000 ms i opóźnieniem dostarczania wynoszącym 5 000 ms, a następnie został odebrany po upływie 10 000 ms, wartość w polu MQMD.expiry może wynosić około 50 dziesiątych sekundy. Ta wartość wskazuje, że upłynęło 15 sekund od momentu umieszczenia komunikatu do momentu jego uzyskania.

Jeśli komunikat utraci ważność w kolejce przemieszczania i zostanie ustawiona jedna z opcji MQRO_EXPIRATION_ *, wygenerowany raport będzie dotyczyć oryginalnego komunikatu wysłanego przez aplikację, nagłówek zawierający informacje o opóźnieniu dostarczania zostanie usunięty.

## Zatrzymywanie i uruchamianie procesora opóźnienia dostarczania

z/OS W systemie z/OSprocesor opóźnienia dostarczania jest zintegrowany z przestrzenią adresową MSTR menedżera kolejek. Po uruchomieniu menedżera kolejek uruchamiany jest również procesor opóźnienia dostarczania. Jeśli kolejka pomostowa jest dostępna, otwiera ją i oczekuje na nadejście komunikatów do przetworzenia. Jeśli kolejka pomostowa nie została zdefiniowana lub jest wyłączona dla operacji pobierania albo wystąpi inny błąd, procesor opóźnienia dostarczania zostanie wyłączony. Jeśli kolejka pomostowa zostanie później zdefiniowana lub zmodyfikowana w celu włączenia pobierania, procesor opóźnienia dostarczania zostanie zrestartowany. Jeśli procesor opóźnienia dostarczania zostanie wyłączony z jakiegokolwiek innego powodu, można go zrestartować, zmieniając wartość atrybutu **PUT** kolejki pomostowej z ENABLED na DISABLED i ponownie na ENABLED . Jeśli z dowolnej przyczyny konieczne jest zatrzymanie procesora opóźnienia dostarczania, należy ustawić atrybut PUT kolejki pomostowej na wartość DISABLED.

Multi W systemie Wiele platformprocesor opóźnienia jest uruchamiany wraz z menedżerem kolejek i jest automatycznie restartowany w przypadku awarii naprawialnej.

## Niepowodzenie umieszczenia w kolejce docelowej

Jeśli opóźniony komunikat nie może zostać umieszczony w kolejce docelowej po zakończeniu opóźnienia, komunikat jest traktowany w sposób wskazany w jego opcjach raportu: jest usuwany lub wysyłany do kolejki niedostarczonych komunikatów. Jeśli to działanie nie powiedzie się, zostanie podjęta próba umieszczenia komunikatu w późniejszym czasie. Jeśli działanie zakończy się pomyślnie, zostanie wygenerowany raport o wyjątku i wysłany do określonej kolejki, jeśli raport zostanie zażądany. Jeśli nie można wysłać komunikatu raportu, zostanie on wysłany do kolejki niedostarczonych komunikatów. Jeśli wysyłanie raportu do kolejki niedostarczonych komunikatów nie powiedzie się, a komunikat jest trwały, wszystkie zmiany są usuwane, a oryginalny komunikat jest wycofywany i ponownie dostarczany później.

Jeśli komunikat jest nietrwały, komunikat raportu jest odrzucany, ale inne zmiany są zatwierdzane. Jeśli nie można dostarczyć opóźnionej publikacji, ponieważ subskrybent anulował subskrypcję lub w przypadku nietrwałego subskrybenta, ponieważ jest on rozłączony, komunikat jest odrzucany w trybie cichym. Komunikaty raportu są nadal generowane zgodnie z wcześniejszym opisem.

Jeśli opóźniona publikacja nie może zostać dostarczona do subskrybenta i zostanie umieszczona w kolejce niedostarczonych komunikatów, a umieszczenie w kolejce niedostarczonych komunikatów nie powiedzie się, komunikat zostanie odrzucony.

Aby zmniejszyć prawdopodobieństwo niepowodzenia operacji umieszczania w kolejce docelowej po zakończeniu opóźnienia dostarczania, menedżer kolejek przeprowadza pewne podstawowe sprawdzenia, gdy klient JMS wysyła komunikat z niezerowym opóźnieniem dostarczania. Te sprawdzenia obejmują, czy kolejka jest wyłączona, czy komunikat jest dłuższy niż maksymalna dozwolona długość komunikatu i czy kolejka jest pełna.

## Publikowanie / subskrypcja

Dopasowanie publikacji do dostępnych subskrypcji ma miejsce, gdy aplikacja JMS wysyła komunikat z niezerowym opóźnieniem dostarczenia. Komunikat dla każdego zgodnego subskrybenta jest umieszczany w systemie SYSTEM.DDELAY.LOCAL.QUEUE , w której jest przechowywana do czasu zakończenia opóźnienia dostarczania. Jeśli jeden z tych subskrybentów jest subskrypcją proxy dla innego menedżera kolejek, po zakończeniu opóźnienia dostawy następuje zwielokrotnienie w tym menedżerze kolejek. Może to spowodować, że subskrybenci w innym menedżerze kolejek będą otrzymywać publikacje, które zostały pierwotnie opublikowane przed zasubskrybowaniem. Jest to odchylenie od specyfikacji JMS 2.0 lub nowszej.

Opóźnienie dostarczania z publikowaniem/subskrybowaniem jest obsługiwane tylko wtedy, gdy temat docelowy jest skonfigurowany z parametrem (N) PMSGDLV = ALLAVAIL. Próba użycia innych wartości powoduje wystąpienie błędu MQRC_PUBLICATION_FAILURE. Jeśli działanie procesora opóźnienia dostarczenia zakończy się niepowodzeniem podczas umieszczania komunikatu w kolejce docelowej, wynik jest opisany w sekcji "Niepowodzenie umieszczenia w kolejce docelowej".

## Komunikaty raportu

Wszystkie opcje raportu są obsługiwane i obsługiwane przez procesor dostarczania, inne niż następujące opcje, które są ignorowane, ale przekazywane w komunikacie, gdy jest on wysyłany do kolejki docelowej:

- MQRO_KOA*
- MQRO_KOD *
- Panel MQRO/MQRO_NAN
- DZIAŁANIE MQRO_ACTIVITY

### *Subskrypcje sklonowane i współużytkowane*

W produkcie IBM MQ 8.0 lub nowszym istnieją dwie metody nadawania wielu konsumentom dostępu do tej samej subskrypcji. Te dwie metody są realizowane za pomocą sklonowanych subskrypcji lub za pomocą subskrypcji współużytkowanych.

## Sklonowane subskrypcje

Sklonowana subskrypcja jest rozszerzeniem IBM MQ . Sklonowane subskrypcje umożliwiają wielu konsumentom w różnych wirtualnych maszynach języka Java (JVM) współbieżny dostęp do subskrypcji. Tego zachowania można użyć, ustawiając właściwość **CLONESUPP** na wartość `Enabled` w obiekcie ConnectionFactory . Domyślnie **CLONESUPP** ma wartość `Wyłączone`. Sklonowane subskrypcje można włączyć tylko w przypadku subskrypcji trwałych. Jeśli opcja **CLONESUPP** jest włączona, każde kolejne połączenie nawiązywane przy użyciu tej ConnectionFactory jest klonowane.

Subskrypcja trwała może zostać uznana za sklonowaną, jeśli jeden lub więcej konsumentów zostanie utworzonych w celu odbierania komunikatów z tej subskrypcji, czyli jeśli zostali oni utworzeni przy użyciu tej samej nazwy subskrypcji. Można to zrobić tylko wtedy, gdy połączenie, w którym utworzono

konsumentów, ma wartość **CLONESUPP** ustawioną na `Włączone` w obiekcie MQConnectionFactory. Po opublikowaniu komunikatu w temacie subskrypcji kopia tego komunikatu jest wysyłana do subskrypcji. Komunikat jest dostępny dla wszystkich konsumentów, ale tylko jeden z nich go odbiera.

**Uwaga:** Włączenie sklonowanych subskrypcji rozszerza specyfikację JMS .

## Subskrypcje współużytkowane

Subskrypcje współużytkowane, które zostały wprowadzone przez specyfikację JMS 2.0 , umożliwiają współużytkowanie komunikatów z subskrypcji tematu przez wielu konsumentów. Każdy komunikat z subskrypcji jest dostarczany tylko do jednego z konsumentów tej subskrypcji. Subskrypcje współużytkowane są włączane przez odpowiednie wywołanie interfejsu API w wersji JMS 2.0 lub nowszej.

Interfejsy API można wywoływać w jeden z następujących sposobów:

- Z aplikacji Java SE (lub kontenera klienta Java EE ).
- Z serwletu lub implementacji komponentu MDB.

Specyfikacja JMS 2.0 lub nowsza nie definiuje żadnego standardowego sposobu kierowania komponentem MDB z subskrypcji współużytkowanej, dlatego produkt IBM MQ 8.0 lub nowszy udostępnia w tym celu właściwość specyfikacji aktywowania **`sharedSubscription`** . Więcej informacji na temat tej właściwości zawierają “Konfigurowanie adaptera zasobów na potrzeby komunikacji przychodzącej” na stronie 467 i “Przykłady definiowania właściwości sharedSubscription” na stronie 486.

Jeśli subskrypcja współużytkowana jest włączona, nie można anulować jej współużytkowania.

Subskrypcje współużytkowane mogą być tworzone jako subskrypcje trwałe lub nietrwałe. Nie ma konieczności oddzielnego tworzenia obiektów po stronie menedżera kolejek poza normalną konfiguracją produktu JMS . Wszystkie wymagane obiekty są tworzone dynamicznie.

## Podejmowanie decyzji między subskrypcjami współużytkowanymi lub sklonowanymi

Podejmując decyzję, czy mają być używane subskrypcje współużytkowane, czy sklonowane, należy wziąć pod uwagę korzyści wynikające z obu tych metod. Tam, gdzie to możliwe, należy używać subskrypcji współużytkowanych, ponieważ jest to zachowanie zdefiniowane w specyfikacji, a nie rozszerzenie specyficzne dla produktu IBM MQ .

Poniższa tabela zawiera niektóre punkty, które należy wziąć pod uwagę podczas podejmowania decyzji między subskrypcjami współużytkowanymi i sklonowanymi:

*Tabela 48. Porównanie uwag dotyczących subskrypcji współużytkowanych i sklonowanych*

| **Subskrypcje współużytkowane** | **Sklonowane subskrypcje** |
|---|---|
| Subskrypcje współużytkowane są standardową częścią specyfikacji produktu JMS 2.0 lub nowszego. | Sklonowane subskrypcje są rozszerzeniem specyficznym dla produktu IBM MQ . |
| Subskrypcje współużytkowane są tworzone przy użyciu jawnych wywołań metod interfejsu API. | Klonowane subskrypcje są kontrolowane administracyjnie na poziomie ConnectionFactory . |
| Subskrypcje współużytkowane mogą być trwałe lub nietrwałe. | Sklonowane subskrypcje mogą być tylko trwałe. |
| Subskrypcje współużytkowane są tworzone jawnie dla poszczególnych subskrypcji. | Sklonowane subskrypcje są używane dla każdej trwałej subskrypcji w ramach połączenia, dla którego funkcja jest włączona. |

| *Tabela 48. Porównanie uwag dotyczących subskrypcji współużytkowanych i sklonowanych (kontynuacja)* | |
|---|---|
| **Subskrypcje współużytkowane** | **Sklonowane subskrypcje** |
| Jeśli subskrypcja została utworzona jako współużytkowana, nie można jej później zmienić na niewspółużytkowaną (lub odwrotnie). | Subskrypcja może zostać zmieniona z klonowanej na niesklonowaną za każdym razem, gdy zostanie ponownie otwarta, a właściwość **CLONESUPP** połączenia będącego właścicielem zostanie zmieniona. |

**Pojęcia pokrewne**

Subskrybenci i subskrypcje

Trwałość subskrypcji

**Zadania pokrewne**

Korzystanie ze współużytkowanych subskrypcji programu JMS 2.0

**Odsyłacze pokrewne**

“Przykłady definiowania właściwości sharedSubscription” na stronie 486

Właściwość sharedSubscription specyfikacji aktywowania można zdefiniować w pliku WebSphere Liberty `server.xml` . Inną możliwością jest zdefiniowanie właściwości w komponencie bean sterowanym komunikatami (MDB) przy użyciu adnotacji.

CLONESUPP

### *Właściwość SupportMQExtensions*

W specyfikacji JMS 2.0 wprowadzono zmiany w sposobie działania niektórych zachowań. IBM MQ 8.0 i nowsze zawierają właściwość **`com.ibm.mq.jms.SupportMQExtensions`**, którą można ustawić na wartość TRUE , aby przywrócić poprzednie implementacje tych zmienionych zachowań.

V 9.3.0 JM 3.0 V 9.3.0 Właściwość **`com.ibm.mq.jakarta.jms.SupportMQExtensions`** (Jakarta Messaging 3.0) jest obsługiwana przez IBM MQ classes for Jakarta Messaging, które są dostępne w pliku `com.ibm.mq.jakarta.client.jar`.

JMS 2.0 Właściwość **`com.ibm.mq.jms.SupportMQExtensions`** (JMS 2.0) jest obsługiwana przez IBM MQ classes for JMS, które są dostępne w systemach `com.ibm.mq.allclient.jar` i `com.ibm.mqjms.jar`.

Trzy obszary funkcjonalności są przywracane przez ustawienie parametru **`SupportMQExtensions`** na wartość `True`:

**Priorytet komunikatu**
: Wiadomościom można przypisać priorytet z zakresu od 0 do 9. Przed JMS 2.0komunikaty mogą również używać wartości -1, co oznacza, że używany jest domyślny priorytet kolejki. JMS 2.0 i nowsze wersje nie zezwalają na ustawienie priorytetu komunikatu -1. Włączenie opcji **`SupportMQExtensions`** umożliwia użycie wartości -1.

**Identyfikator klienta**
: Specyfikacja JMS 2.0 lub nowsza wymaga, aby identyfikatory klientów o wartości innej niż NULL były sprawdzane pod kątem unikalności podczas nawiązywania połączenia. Włączenie opcji **`SupportMQExtensions`** oznacza, że to wymaganie nie jest brane pod uwagę i że identyfikator klienta może być ponownie wykorzystywany.

**NoLocal**
: Specyfikacja JMS 2.0 lub nowsza wymaga, aby po włączeniu tej stałej konsument nie mógł odbierać komunikatów publikowanych przez ten sam identyfikator klienta. Przed produktem JMS 2.0ten atrybut został ustawiony na subskrybencie, aby uniemożliwić mu odbieranie komunikatów publikowanych przez jego własne połączenie. Włączenie opcji **`SupportMQExtensions`** przywraca to zachowanie do poprzedniej implementacji.

Tę właściwość można ustawić w następujący sposób:

```
java -Dcom.ibm.mq.jms.SupportMQExtensions=true
```

Tę właściwość można ustawić jako standardową właściwość systemową maszyny JVM w komendzie **java** lub w pliku konfiguracyjnym IBM MQ classes for JMS .

**Pojęcia pokrewne**

“Plik konfiguracyjny IBM MQ classes for JMS/Jakarta Messaging” na stronie 102
Pliki konfiguracyjne produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging określają właściwości, które są używane do konfigurowania produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging.

**Odsyłacze pokrewne**

“Właściwości używane do konfigurowania zachowania klienta JMS” na stronie 109
Te właściwości służą do konfigurowania zachowania klienta JMS .

## *Korzystanie z subskrypcji współużytkowanych w aplikacjach JMS*

W przypadku subskrypcji współużytkowanych pojedyncza subskrypcja jest współużytkowana przez wielu konsumentów, a tylko jeden z nich otrzymuje publikację w danym momencie.

Subskrypcje współużytkowane są dostępne od wersji JMS 2.0 . W związku z tym podczas tworzenia aplikacji produktu JMS dla systemu IBM MQ 8.0 lub nowszego może być konieczne wzięcie pod uwagę wpływu tej funkcji na menedżer kolejek.

Ideą subskrypcji współużytkowanych jest współużytkowanie obciążenia przez wielu konsumentów. Trwała subskrypcja może być również współużytkowana przez wielu konsumentów.

Załóżmy na przykład, że:

- Subskrypcja SUB, subskrybowanie tematu FIFA2014/UPDATES , aby otrzymywać aktualizacje dotyczące meczu piłki nożnej, współużytkowane przez trzech konsumentów C1, C2i C3
- Publikowanie producenta P1 w temacie FIFA2014/UPDATES

Po opublikowaniu w serwisie FIFA2014/UPDATESzostanie ono odebrane tylko przez jednego z trzech konsumentów (C1, C2lub C3), ale nie przez wszystkich.

W poniższym przykładzie przedstawiono użycie subskrypcji współużytkowanych, a także użycie dodatkowego interfejsu API w produkcie JMS 2.0( `Message.receiveBody()`) w celu pobrania tylko treści komunikatu.

Przykład tworzy trzy wątki subskrybenta, które tworzą współużytkowaną subskrypcję tematu FIFA2014/UPDATES i jeden wątek publikatora.

V 9.3.0 JM 3.0 V 9.3.0

```
package mqv91Samples;

import jakarta.jms.JMSException;

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(JMSException 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 stadiumAttendence = 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 Attendence " + stadiumAttendence;
                        stadiumAttendence += 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(JMSException 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();
    }
}
```

**Pojęcia pokrewne**

Interfejsy języka IBM MQ Java

## V 9.3.2 Konfigurowanie aplikacji modułowej pod kątem używania produktu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging

V 9.3.2 Modułów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging można używać w sposób modułowy, wymagając odpowiedniego modułu w aplikacji i dołączając odpowiedni katalog do ścieżki modułu.

### Opakowanie modułowe

Zunifikowane pliki JAR dla produktów IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging udostępniają automatyczne nazwy modułów, które zastępują domyślne nazwy pochodzące z nazw plików JAR.

- Moduł IBM MQ classes for JMS (`com.ibm.mq.allclient.jar`) jest dostarczany z nazwą modułu `com.ibm.mq.javax`.
- Moduł IBM MQ classes for Jakarta Messaging (`com.ibm.mq.jakarta.client.jar`) jest dostarczany z nazwą modułu `com.ibm.mq.jakarta`.

Domyślny katalog `MQ_HOME/java/lib` nie jest odpowiedni do modułowego użycia, ponieważ moduły nie mogą zawierać tego samego pakietu, a katalog domyślny zawiera te same pakiety w wielu plikach JAR. Dlatego dostępne są nowe katalogi, które zawierają tylko potrzebne pliki JAR, bez duplikowania pakietów między plikami JAR. Te katalogi są odpowiednie do włączenia do `module-path`.

**Uwaga:** Jeśli istnieją aplikacje, które używają dostępnych plików JAR w kontekście modułowym, polegając na domyślnych nazwach modułów, należy zaktualizować aplikacje tak, aby wymagały nowych nazw modułów. Domyślne nazwy modułów pochodzą z nazw plików JAR.

### Konfigurowanie aplikacji modułowej pod kątem używania produktu IBM MQ classes for JMS

Aplikację modułową można skonfigurować w taki sposób, aby korzystała z produktu IBM MQ classes for JMS (`com.ibm.mq.allclient.jar`), wykonując następujące kroki:

- Skonfiguruj aplikację tak, aby wymagała modułu `com.ibm.mq.javax` .
- Skonfiguruj aplikację, aby dołączyć katalog `MQ_HOME/java/lib/modules/javax` do ścieżki modułu.

### Konfigurowanie aplikacji modułowej pod kątem używania produktu IBM MQ classes for Jakarta Messaging

Aplikację modułową można skonfigurować w taki sposób, aby korzystała z produktu IBM MQ classes for Jakarta Messaging (`com.ibm.mq.jakarta.client.jar`), wykonując następujące kroki:

- Skonfiguruj aplikację tak, aby wymagała modułu `com.ibm.mq.jakarta` .
- Skonfiguruj aplikację, aby dołączyć katalog `MQ_HOME/java/lib/modules/jakarta` do ścieżki modułu.

### Konfigurowanie aplikacji modułowej pod kątem używania produktu IBM MQ classes for Java

Aby użyć produktu IBM MQ classes for Java z aplikacji modułowej, można użyć konfiguracji dla produktu IBM MQ classes for JMS lub konfiguracji dla produktu IBM MQ classes for Jakarta Messaging, ponieważ

oba pliki JAR klienta obsługują plik IBM MQ classes for Java. Jednak aplikacja musi używać tylko jednej z tych konfiguracji, a nie obu.

## Narzędzia serwera IBM MQ classes for JMS Application Server

W tym temacie opisano, w jaki sposób produkt IBM MQ classes for JMS implementuje klasę ConnectionConsumer i zaawansowaną funkcjonalność w klasie Session. Zawiera również podsumowanie funkcji puli sesji serwera.

**Ważne:** Te informacje są przeznaczone wyłącznie do celów informacyjnych. Aplikacja nie może być napisana w taki sposób, aby używała tego interfejsu: jest on używany w adapterze zasobów IBM MQ do nawiązywania połączeń z serwerami Java EE. Aby uzyskać praktyczne informacje o połączeniu, patrz “Korzystanie z adaptera zasobów IBM MQ” na stronie 450.

IBM MQ classes for JMS obsługuje narzędzia ASF (Application Server Facilities) określone w specyfikacji *Java Message Service Specification* (patrz Oracle Technology Network for Java Developers). Ta specyfikacja identyfikuje trzy role w tym modelu programistycznym:

- **Dostawca JMS** udostępnia ConnectionConsumer i zaawansowane funkcje sesji.
- **Serwer aplikacji** udostępnia funkcje ServerSessionPool i ServerSession .
- **Aplikacja kliencka** korzysta z funkcji dostawcy JMS i serwera aplikacji.

Informacje zawarte w tym temacie nie mają zastosowania, jeśli aplikacja używa połączenia z brokerem w czasie rzeczywistym.

### *JMS ConnectionConsumer*

Interfejs ConnectionConsumer udostępnia wysokowydajną metodę współbieżnego dostarczania komunikatów do puli wątków.

Specyfikacja JMS umożliwia ścisłą integrację serwera aplikacji z implementacją serwera JMS przy użyciu interfejsu `ConnectionConsumer` . Ta opcja umożliwia współbieżne przetwarzanie komunikatów. Zwykle serwer aplikacji tworzy pulę wątków, a implementacja JMS udostępnia komunikaty tym wątkom. Serwer aplikacji obsługujący system JMS(na przykład WebSphere Application Server) może używać tej funkcji do udostępniania funkcji przesyłania komunikatów wysokiego poziomu, takich jak komponenty bean sterowane komunikatami.

Zwykłe aplikacje nie korzystają z interfejsu ConnectionConsumer, ale mogą z niego korzystać zaawansowane klienty JMS . Dla takich klientów klasa ConnectionConsumer udostępnia wydajną metodę współbieżnego dostarczania komunikatów do puli wątków. Po nadejściu komunikatu do kolejki lub tematu program JMS wybiera wątek z puli i dostarcza do niego partię komunikatów. W tym celu program JMS uruchamia powiązaną metodę `onMessage()` obiektu MessageListener.

Ten sam efekt można uzyskać, konstruując wiele obiektów Session i MessageConsumer , z których każdy ma zarejestrowany obiekt MessageListener. Jednak klasa ConnectionConsumer zapewnia lepszą wydajność, mniejsze wykorzystanie zasobów i większą elastyczność. W szczególności wymagana jest mniejsza liczba obiektów sesji.

### *Planowanie aplikacji z ASF*

W tej sekcji opisano sposób planowania aplikacji, w tym:

- “Ogólne zasady przesyłania komunikatów w trybie punkt z punktem przy użyciu ASF” na stronie 344
- “Ogólne zasady przesyłania komunikatów w trybie publikowania/subskrypcji przy użyciu ASF” na stronie 345
- “Usuwanie komunikatów z kolejki w ASF” na stronie 346
- Obsługa komunikatów nieprzetwarzalnych w ASF. Patrz “Obsługa komunikatów nieprzetwarzalnych w produkcie IBM MQ classes for JMS” na stronie 240.

*Ogólne zasady przesyłania komunikatów w trybie punkt z punktem przy użyciu ASF*

Ten temat zawiera ogólne informacje dotyczące przesyłania komunikatów w trybie punkt z punktem przy użyciu ASF.

Gdy aplikacja tworzy obiekt ConnectionConsumer z obiektu QueueConnection , określa obiekt kolejki JMS i łańcuch selektora. Następnie konsument ConnectionConsumer rozpoczyna udostępnianie komunikatów dla sesji w powiązanej puli ServerSession. Komunikaty docierają do kolejki i jeśli są zgodne z selektorem, są dostarczane do sesji w powiązanej puli ServerSession.

W terminologii IBM MQ obiekt kolejki odnosi się do QLOCAL lub QALIAS w lokalnym menedżerze kolejek. Jeśli jest to QALIAS, to QALIAS musi odnosić się do QLOCAL. W pełni rozstrzygnięta nazwa IBM MQ QLOCAL jest znana jako *bazowa nazwa QLOCAL*. Konsument ConnectionConsumer jest *aktywny* , jeśli nie jest zamknięty, a jego element nadrzędny QueueConnection jest uruchomiony.

Możliwe jest uruchomienie wielu ConnectionConsumers, z których każdy ma różne selektory, dla tego samego bazowego połączenia QLOCAL. Aby utrzymać wydajność, niepożądane komunikaty nie mogą być gromadzone w kolejce. Niepożądane komunikaty to te, dla których żaden aktywny konsument ConnectionConsumer nie ma zgodnego selektora. Fabrykę QueueConnectionmożna ustawić w taki sposób, aby niepożądane komunikaty były usuwane z kolejki (szczegółowe informacje na ten temat zawiera sekcja “Usuwanie komunikatów z kolejki w ASF” na stronie 346 ). To zachowanie można ustawić na jeden z dwóch sposobów:

- Użyj narzędzia administracyjnego JMS , aby ustawić fabrykę QueueConnectionna wartość MRET (NO).
- W programie należy użyć:

```
MQQueueConnectionFactory.setMessageRetention(WMQConstants.WMQ_MRET_NO)
```

Jeśli to ustawienie nie zostanie zmienione, domyślnie takie niechciane komunikaty będą przechowywane w kolejce.

Podczas konfigurowania menedżera kolejek produktu IBM MQ należy wziąć pod uwagę następujące kwestie:

- Bazowe zadanie QLOCAL musi być włączone dla współużytkowanych danych wejściowych. W tym celu należy użyć następującej komendy MQSC:

```
ALTER QLOCAL( your.qlocal.name ) SHARE GET(ENABLED)
```

- Menedżer kolejek musi mieć włączoną kolejkę niedostarczonych komunikatów. Jeśli podczas umieszczania komunikatu w kolejce niedostarczonych komunikatów wystąpi problem z obiektem ConnectionConsumer , dostarczanie komunikatów z bazowego obiektu QLOCAL zostanie zatrzymane. Aby zdefiniować kolejkę niedostarczonych komunikatów, należy użyć komendy:

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- Użytkownik, który uruchamia interfejs ConnectionConsumer , musi mieć uprawnienia do wykonywania operacji MQOPEN z MQOO_SAVE_ALL_CONTEXT i MQOO_PASS_ALL_CONTEXT. Szczegółowe informacje na ten temat zawiera dokumentacja IBM MQ dla konkretnej platformy.
- Jeśli w kolejce pozostaną niepożądane komunikaty, obniżą one wydajność systemu. Dlatego należy zaplanować selektory komunikatów w taki sposób, aby między nimi klasa ConnectionConsumers (Konsumenty połączeń) usuwała wszystkie komunikaty z kolejki.

Szczegółowe informacje na temat komend MQSC zawiera sekcja Komendy MQSC.

*Ogólne zasady przesyłania komunikatów w trybie publikowania/subskrypcji przy użyciu ASF*
Klasa ConnectionConsumers odbiera komunikaty dla określonego tematu. Klasa ConnectionConsumer może być trwała lub nietrwała. Należy określić, która kolejka lub które kolejki są używane przez ConnectionConsumer .

Gdy aplikacja tworzy obiekt ConnectionConsumer na podstawie obiektu TopicConnection , określa on obiekt Topic i łańcuch selektora. Następnie konsument ConnectionConsumer rozpoczyna odbieranie komunikatów zgodnych z selektorem danego tematu, w tym wszelkich zachowanych publikacji dla subskrybowanego tematu.

Alternatywnie aplikacja może utworzyć trwały obiekt ConnectionConsumer , który jest powiązany z konkretną nazwą. Ten konsument ConnectionConsumer odbiera komunikaty opublikowane w temacie od czasu ostatniej aktywności trwałego konsumenta ConnectionConsumer . Odbiera on wszystkie takie komunikaty, które są zgodne z selektorem w temacie. Jeśli jednak konsument ConnectionConsumer używa odczytu z wyprzedzeniem, podczas zamykania może utracić nietrwałe komunikaty, które znajdują się w buforze klienta.

Jeśli produkt IBM MQ classes for JMS jest w trybie migracji dostawcy usługi przesyłania komunikatów produktu IBM MQ , dla nietrwałych subskrypcji ConnectionConsumer używana jest oddzielna kolejka.Konfigurowalna opcja CCSUB fabryki TopicConnectionokreśla kolejkę, która ma być używana. Zwykle CCSID określa pojedynczą kolejkę do użycia przez wszystkie ConnectionConsumers , które korzystają z tej samej fabryki TopicConnection. Możliwe jest jednak, aby każdy konsument ConnectionConsumer wygenerował kolejkę tymczasową, określając przedrostek nazwy kolejki, po którym następuje gwiazdka (*).

Jeśli produkt IBM MQ classes for JMS jest w trybie migracji dostawcy usługi przesyłania komunikatów produktu IBM MQ , właściwość CCDSUB tematu określa kolejkę, która ma być używana na potrzeby trwałych subskrypcji. Może to być kolejka, która już istnieje, lub przedrostek nazwy kolejki, po którym następuje gwiazdka (*). Jeśli zostanie podana kolejka, która już istnieje, wszystkie trwałe ConnectionConsumers (Konsumenty połączeń), które subskrybują temat, będą używać tej kolejki. Jeśli zostanie podany przedrostek nazwy kolejki, po którym następuje gwiazdka (*), kolejka jest generowana przy pierwszym utworzeniu trwałego obiektu ConnectionConsumer o określonej nazwie. Ta kolejka jest ponownie wykorzystywana później, gdy zostanie utworzony trwały obiekt ConnectionConsumer o takiej samej nazwie.

Podczas konfigurowania menedżera kolejek produktu IBM MQ należy wziąć pod uwagę następujące kwestie:

- Menedżer kolejek musi mieć włączoną kolejkę niedostarczonych komunikatów. Jeśli podczas umieszczania komunikatu w kolejce niedostarczonych komunikatów wystąpi problem z obiektem ConnectionConsumer , dostarczanie komunikatów z bazowego obiektu QLOCAL zostanie zatrzymane. Aby zdefiniować kolejkę niedostarczonych komunikatów, należy użyć komendy:

  ```
  ALTER QMGR DEADQ( your.dead.letter.queue.name )
  ```

- Użytkownik, który uruchamia interfejs ConnectionConsumer , musi mieć uprawnienia do wykonywania operacji MQOPEN z MQOO_SAVE_ALL_CONTEXT i MQOO_PASS_ALL_CONTEXT. Szczegółowe informacje na ten temat zawiera dokumentacja IBM MQ dla używanej platformy.
- Wydajność pojedynczego obiektu ConnectionConsumer można zoptymalizować, tworząc dla niego oddzielną, dedykowaną kolejkę. Jest to koszt dodatkowego wykorzystania zasobów.

*Usuwanie komunikatów z kolejki w ASF*
Jeśli aplikacja używa klasy ConnectionConsumers, w wielu sytuacjach może być konieczne usunięcie komunikatów z kolejki przez program JMS .

Są to następujące sytuacje:

**Niepoprawnie sformatowany komunikat**
Może zostać odebrany komunikat, którego program JMS nie może przeanalizować.

**Wiadomość nieprzetwarzalna**
Komunikat może osiągnąć próg wycofania, ale konsument ConnectionConsumer nie może ponownie umieścić go w kolejce wycofania.

**Brak zainteresowanych ConnectionConsumer**
W przypadku przesyłania komunikatów w trybie punkt z punktem, gdy fabryka QueueConnectionjest ustawiona w taki sposób, że nie zachowuje niepożądanych komunikatów, pojawia się komunikat, który jest niepożądany przez ConnectionConsumers.

W takich sytuacjach ConnectionConsumer próbuje usunąć komunikat z kolejki. Opcje rozporządzania w polu raportu deskryptora MQMD komunikatu ustawiają dokładne zachowanie. Dostępne są następujące opcje:

**MQRO_DEAD_LETTER_Q**
Komunikat jest umieszczany w kolejce niedostarczonych komunikatów menedżera kolejek. Jest to opcja domyślna.

**MQRO_DISCARD_MSG**
Komunikat został odrzucony.

Klasa ConnectionConsumer generuje również komunikat raportu, który jest również zależny od pola raportu deskryptora MQMD komunikatu. Ten komunikat jest wysyłany do kolejki ReplyTow menedżerze kolejek ReplyTo. Jeśli podczas wysyłania komunikatu raportu wystąpi błąd, komunikat zostanie wysłany do kolejki niedostarczonych komunikatów. Opcje raportu wyjątków w polu raportu zestawu MQMD komunikatu dotyczące szczegółów komunikatu raportu. Dostępne są następujące opcje:

**MQRO_EXCEPTION,**
Zostanie wygenerowany komunikat raportu, który zawiera deskryptor MQMD oryginalnego komunikatu. Nie zawiera żadnych danych treści komunikatu.

**MQRO_EXCEPTION_WITH_DATA**
Generowany jest komunikat raportu, który zawiera deskryptor MQMD, wszystkie nagłówki MQ i 100 bajtów danych treści.

**MQRO_EXCEPTION_WITH_FULL_DATA**
Generowany jest komunikat raportu, który zawiera wszystkie dane z oryginalnego komunikatu.

**default**
Nie jest generowany żaden komunikat raportu.

Podczas generowania raportów uwzględniane są następujące opcje:

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID (Identyfikator KORELACJI MQ)

Jeśli komunikat nieprzetwarzalny nie może zostać umieszczony w kolejce, być może z powodu zapełnienia kolejki niewysłanych wiadomości lub błędnego określenia autoryzacji, to, co się stanie, zależy od trwałości komunikatu. Jeśli komunikat jest nietrwały, komunikat jest odrzucany i nie jest generowany żaden komunikat raportu. Jeśli komunikat jest trwały, dostarczanie komunikatów do wszystkich konsumentów połączeń nasłuchujących w tym miejscu docelowym zostanie zatrzymane. Takie konsumenty połączeń muszą zostać zamknięte, a problem rozwiązany przed ponownym utworzeniem i zrestartowaniem dostarczania komunikatów.

Ważne jest, aby zdefiniować kolejkę niedostarczonych komunikatów i regularnie ją sprawdzać w celu upewnienia się, że nie występują żadne problemy. W szczególności upewnij się, że kolejka niedostarczonych komunikatów nie osiągnęła maksymalnego zapełnienia i że maksymalna wielkość komunikatu jest wystarczająca dla wszystkich komunikatów.

Po umieszczeniu komunikatu w kolejce niedostarczonych komunikatów jest on poprzedzony nagłówkiem IBM MQ dead-letter (MQDLH). Szczegółowe informacje na temat formatu MQDLH zawiera sekcja MQDLH-Dead-letter header (Nagłówek niedostarczonego komunikatu). Następujące pola umożliwiają zidentyfikowanie komunikatów umieszczonych przez ConnectionConsumer w kolejce niedostarczonych komunikatów lub zgłoszenie komunikatów wygenerowanych przez ConnectionConsumer :

- PutApplTyp to `MQAT_JAVA (0x1C)`
- PutApplNazwa to "`MQ JMS ConnectionConsumer`

Pola te znajdują się w nagłówku MQDLH komunikatów w kolejce niedostarczonych komunikatów oraz w deskryptorze MQMD komunikatów raportu. Pole informacji zwrotnej deskryptora MQMD i pole przyczyny deskryptora MQDLH zawierają kod opisujący błąd. Szczegółowe informacje na temat tych kodów zawiera sekcja “Kody przyczyny i sprzężenia zwrotnego w ASF” na stronie 349. Inne pola są opisane w sekcji MQDLH-nagłówek niedostarczonego komunikatu.

*Obsługa komunikatów nieprzetwarzalnych w ASF*
W ramach narzędzi serwera aplikacji obsługa komunikatów nieprzetwarzalnych jest obsługiwana nieco inaczej niż w innych miejscach w produkcie IBM MQ classes for JMS.

Informacje na temat obsługi komunikatów nieprzetwarzalnych w produkcie IBM MQ classes for JMSzawiera sekcja "Obsługa komunikatów nieprzetwarzalnych w produkcie IBM MQ classes for JMS" na stronie 240.

W przypadku korzystania z narzędzi ASF (Application Server Facilities) konsument ConnectionConsumer, a nie MessageConsumer, przetwarza komunikaty nieprzetwarzalne. Konsument ConnectionConsumer ponownie umieszcza komunikaty w kolejce zgodnie z właściwościami nazwy QName BackoutThreshold i BackoutRequeuekolejki.

Jeśli aplikacja używa klasy ConnectionConsumers, okoliczności, w których komunikat jest wycofywany, zależą od sesji udostępnianej przez serwer aplikacji:

- Jeśli sesja nie jest transakcją, z wartością AUTO_ACKNOWLEDGE lub DUPS_OK_ACKNOWLEDGE, komunikat jest wycofywany tylko po wystąpieniu błędu systemowego lub w przypadku nieoczekiwanego zakończenia aplikacji.
- Jeśli sesja nie jest transakcją z CLIENT_ACKNOWLEDGE, niepotwierdzone komunikaty mogą zostać wycofane przez serwer aplikacji wywołując metodę Session.recover().

  Zwykle implementacja klienta MessageListener lub serwer aplikacji wywołuje metodę Message.acknowledge(). Message.acknowledge() potwierdza wszystkie komunikaty dostarczone do tej pory w sesji.
- Gdy sesja jest transakcją, niepotwierdzone komunikaty mogą zostać wycofane przez serwer aplikacji wywołujący metodę Session.rollback().
- Jeśli serwer aplikacji udostępnia XASession, komunikaty są zatwierdzane lub wycofywane w zależności od transakcji rozproszonej. Serwer aplikacji przyjmuje odpowiedzialność za zakończenie transakcji.

**Pojęcia pokrewne**
"Obsługa komunikatów nieprzetwarzalnych w produkcie IBM MQ classes for JMS" na stronie 240
Komunikat nieprzetwarzalny to taki, który nie może być przetwarzany przez aplikację odbierającą. Jeśli komunikat nieprzetwarzalny zostanie dostarczony do aplikacji i wycofany określoną liczbę razy, IBM MQ classes for JMS może przenieść go do kolejki wycofania.

## *Obsługa błędów*

W tej sekcji opisano różne aspekty obsługi błędów, w tym "Odzyskiwanie po wystąpieniu błędów w ASF" na stronie 348 i "Kody przyczyny i sprzężenia zwrotnego w ASF" na stronie 349.

*Odzyskiwanie po wystąpieniu błędów w ASF*
Jeśli w obiekcie ConnectionConsumer wystąpi poważny błąd, dostarczanie komunikatów do wszystkich ConnectionConsumers , które interesują się tym samym zatrzymaniem QLOCAL. W takim przypadku powiadamiany jest każdy obiekt ExceptionListener , który jest zarejestrowany w danym połączeniu. Istnieją dwa sposoby, w jaki aplikacja może odzyskać sprawność po wystąpieniu tych warunków błędu.

Zwykle poważny błąd tego rodzaju występuje, jeśli konsument ConnectionConsumer nie może ponownie umieścić komunikatu w kolejce niedostarczonych komunikatów lub wystąpił błąd podczas odczytywania komunikatów z kolejki QLOCAL.

Ponieważ każdy obiekt ExceptionListener , który jest zarejestrowany w tym połączeniu, jest powiadamiany, można go użyć do określenia przyczyny problemu. W niektórych przypadkach administrator systemu musi interweniować w celu rozwiązania problemu.

Użyj jednej z następujących technik, aby naprawić te błędy:

- Wywołaj `close()` dla wszystkich ConnectionConsumers, których to dotyczy. Nowa klasa ConnectionConsumers może zostać utworzona przez aplikację dopiero po zamknięciu wszystkich ConnectionConsumers , których to dotyczy, i rozwiązaniu wszelkich problemów z systemem.

- Wywołaj `stop()` dla wszystkich połączeń, których to dotyczy. Po zatrzymaniu wszystkich połączeń i rozwiązaniu wszystkich problemów z systemem aplikacja może `start()` pomyślnie wykonać swoje połączenia.

*Kody przyczyny i sprzężenia zwrotnego w ASF*
Użyj kodów przyczyny i informacji zwrotnych, aby określić przyczynę błędu. W tym miejscu podano typowe kody przyczyny generowane przez klasę ConnectionConsumer .

Aby określić przyczynę błędu, użyj następujących informacji:

- Kod opinii w dowolnym komunikacie raportu
- Kod przyczyny w MQDLH wszystkich komunikatów w kolejce niedostarczonych komunikatów

ConnectionConsumers (Konsumenci połączeń) generują następujące kody przyczyny.

**MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)**

**Przyczyna**
Komunikat osiągnął próg wycofania zdefiniowany dla kolejki QLOCAL, ale nie zdefiniowano kolejki wycofania.

Na platformach, na których nie można zdefiniować kolejki wycofania, komunikat osiągnął zdefiniowany przez JMSpróg wycofania wynoszący 20.

**Działanie**
Jeśli nie jest to potrzebne, zdefiniuj kolejkę wycofania dla odpowiedniej kolejki QLOCAL. Poszukaj również przyczyny wielu wycofań.

**MQRC_MSG_NOT_MATCHED (0x93B; 2363)**

**Przyczyna**
W przypadku przesyłania komunikatów w trybie punkt z punktem istnieje komunikat, który nie jest zgodny z żadnym z selektorów dla ConnectionConsumers monitorujących kolejkę. Aby utrzymać wydajność, komunikat jest umieszczany w kolejce niedostarczonych komunikatów.

**Działanie**
Aby uniknąć takiej sytuacji, należy upewnić się, że klasa ConnectionConsumers używająca kolejki udostępnia zestaw selektorów, które zajmują się wszystkimi komunikatami, lub ustawić fabrykę QueueConnectionna przechowywanie komunikatów.

Można również sprawdzić źródło komunikatu.

**MQRC_JMS_FORMAT_ERROR (0x93C; 2364)**

**Przyczyna**
Program JMS nie może zinterpretować komunikatu w kolejce.

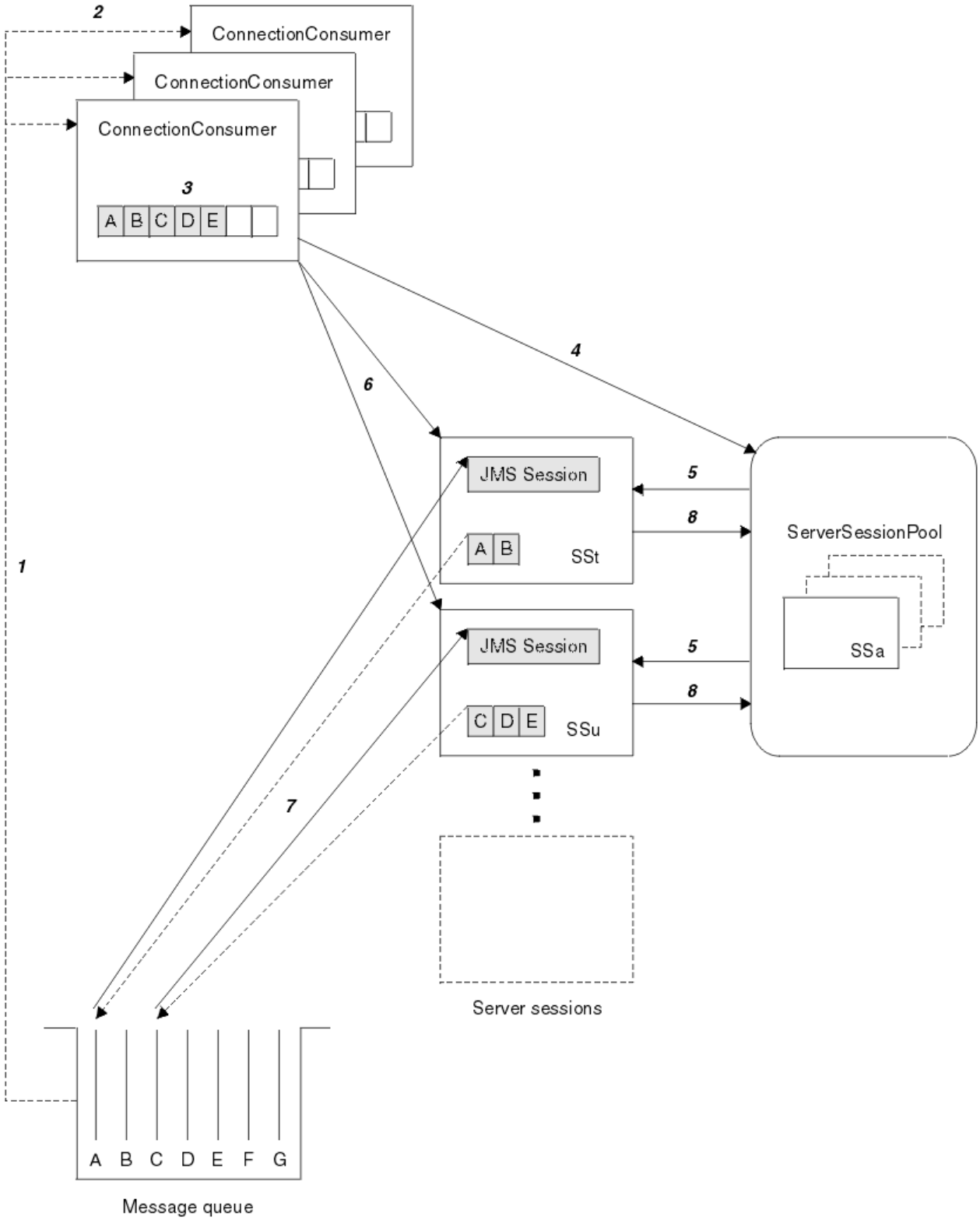
**Działanie**
Sprawdź źródło komunikatu. JMS zwykle dostarcza komunikaty w nieoczekiwanym formacie BytesMessage lub TextMessage. Czasami działanie to kończy się niepowodzeniem, jeśli komunikat jest bardzo źle sformatowany.

Inne kody wyświetlane w tych polach są spowodowane nieudaną próbą ponownego umieszczenia komunikatu w kolejce wycofanych komunikatów. W tej sytuacji kod opisuje przyczynę niepowodzenia ponownego umieszczenia w kolejce. Aby zdiagnozować przyczynę tych błędów, należy zapoznać się z sekcją Kody zakończenia i kody przyczyny interfejsu API.

Jeśli komunikatu raportu nie można umieścić w kolejce ReplyTo, jest on umieszczany w kolejce niedostarczonych komunikatów. W tej sytuacji pole informacji zwrotnej deskryptora MQMD zostało zakończone zgodnie z opisem w tym temacie. Pole przyczyny w MQDLH wyjaśnia, dlaczego nie można umieścić komunikatu raportu w kolejce ReplyTo.

## *Funkcja puli sesji serwera w systemie plików AFS*
W tej sekcji przedstawiono podsumowanie funkcji puli sesji serwera.

Rysunek 45 na stronie 350 zawiera podsumowanie zasad działania funkcji ServerSessionPool i ServerSession .




*Rysunek 45. Funkcjonalność ServerSessionPool i ServerSession*

1. Konsumenty połączeń ConnectionConsumers pobierają odwołania do komunikatów z kolejki.
2. Każdy konsument ConnectionConsumer wybiera konkretne odwołania do komunikatów.
3. W buforze ConnectionConsumer znajdują się wybrane odwołania do komunikatów.

4. Klasa ConnectionConsumer żąda jednego lub kilku ServerSessions z puli ServerSession.
5. ServerSessions są przydzielane z puli ServerSession.
6. Konsument ConnectionConsumer przypisuje odwołania do komunikatów do sesji ServerSessions i uruchamia działające wątki ServerSession .
7. Każda sesja serwera ( ServerSession ) pobiera przywoływane komunikaty z kolejki. Przekazuje je do metody onMessage z obiektu MessageListener , który jest powiązany z sesją JMS .
8. Po zakończeniu przetwarzania sesja serwera ( ServerSession ) jest zwracana do puli.

Serwer aplikacji zwykle udostępnia funkcje ServerSessionPool i ServerSession .

## z/OS Używanie języka IBM MQ classes for JMS na serwerze CICS Liberty JVM

Z poziomu systemu IBM MQ 9.1.0programy Java działające na serwerze CICS Liberty JVM mogą używać pliku IBM MQ classes for JMS w celu uzyskania dostępu do katalogu IBM MQ.

Musi być używana wersja adaptera zasobów produktu IBM MQ ( IBM MQ 9.1.0lub nowsza). Adapter zasobów można uzyskać z serwisu Fix Central (patrz sekcja “Instalowanie adaptera zasobów w systemie Liberty” na stronie 460).

Istnieją dwa rodzaje maszyn JVM profili Liberty dostępnych w wersji CICS 5.3 i nowszych. Typy połączeń, które można nawiązać z produktem IBM MQ , są ograniczone w następujący sposób:

**CICS Liberty Standard**

- Adapter zasobów IBM MQ może łączyć się z dowolną wersją serwera IBM MQ w trybie klienta.
- Adapter zasobów IBM MQ może połączyć się z dowolną wersją serwera IBM MQ for z/OS w trybie BINDINGS, jeśli nie ma połączenia CICS (aktywnego zasobu CICS MQCONN) z tym samym menedżerem kolejek z tego samego regionu CICS .

**CICS Liberty Zintegrowane**

- Adapter zasobów IBM MQ może połączyć się z dowolną wersją serwera IBM MQ w trybie klienta.
- Połączenie w trybie BINDINGS nie jest obsługiwane.

Szczegółowe informacje na temat konfigurowania systemu zawiera sekcja Używanie języka IBM MQ classes for JMS na serwerze JVM Liberty w dokumentacji produktu CICS .

## Używanie IBM MQ classes for JMS/ Jakarta Messaging w IMS

Obsługa przesyłania komunikatów opartego na standardach w środowisku IMS jest udostępniana przy użyciu produktu IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging.

Sprawdź wymagania systemowe dla systemu IMS używanego w przedsiębiorstwie. Więcej informacji na ten temat zawiera publikacja IMS 15.2 .

W tym zestawie tematów opisano sposób konfigurowania IBM MQ classes for JMS w środowisku IMS oraz ograniczenia dotyczące interfejsu API, które mają zastosowanie w przypadku korzystania z klasycznych (JMS 1.1) i uproszczonych (JMS 2.0) interfejsów. Listę informacji specyficznych dla interfejsu API można znaleźć w sekcji “Ograniczenia interfejsu API JMS” na stronie 356 .

**Uwaga:** Podobne ograniczenia mają zastosowanie do wcześniejszych interfejsów specyficznych dla domeny (JMS 1.0.2), ale nie zostały one szczegółowo opisane w tej sekcji.

V 9.3.0 JM 3.0 V 9.3.0 W produkcie IBM MQ 9.3.0produkt Jakarta Messaging 3.0 jest obsługiwany na potrzeby tworzenia nowych aplikacji. Produkt IBM MQ 9.3.0 nadal obsługuje produkt JMS 2.0 dla istniejących aplikacji. Nie jest obsługiwane używanie zarówno interfejsu API Jakarta Messaging 3.0 , jak i interfejsu API JMS 2.0 w tej samej aplikacji. Więcej informacji na ten temat zawiera sekcja Używanie klas IBM MQ dla przesyłania komunikatów JMS/Jakarta.

## Obsługiwane regiony zależne IMS

Obsługiwane są następujące typy regionów zależnych:

- MPR
- BMP
- IFP
- JMP 31 i 64-bitowe maszyny wirtualne Java (JVM)
- JBP 31 i 64-bitowe maszyny JVM

IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging zachowują się tak samo we wszystkich typach regionów, chyba że zostały wymienione w poniższych tematach.

## Obsługiwane maszyny wirtualne Java

IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging wymagają środowiska IBM Runtime Environment, Java Technology Edition 8. Produkt IBM Semeru Runtime Certified Edition dla systemu z/OS, wersja 11, nie jest obsługiwany.

## Inne ograniczenia

Jeśli produkt IBM MQ classes for JMS jest używany w środowisku IMS , obowiązują następujące ograniczenia:

- Połączenia w trybie klienta nie są obsługiwane.
- Połączenia są obsługiwane tylko w przypadku menedżerów kolejek systemu IBM MQ 8.0 korzystających z trybu IBM MQ dostawcy przesyłania komunikatów `Normal`.

  Atrybut **PROVIDERVERSION** w fabryce połączeń musi być nieokreślony lub mieć wartość większą lub równą 7.
- Użycie dowolnej z fabryk połączeń XA, na przykład `com.ibm.mq.jms.MQXAConnectionFactory`, nie jest obsługiwane.

**Zadania pokrewne**

Definiowanie IBM MQ w pliku IMS

# *Konfigurowanie adaptera IMS do użycia z IBM MQ classes for JMS/Jakarta Messaging*

IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging używają tego samego adaptera IBM MQ-IMS , który jest używany przez inne języki programowania. Ten adapter używa narzędzia IMS External Subsystem Attach Facility (ESAF).

## Zanim rozpoczniesz

Przed wykonaniem poniższej procedury należy skonfigurować adapter IMS dla odpowiednich menedżerów kolejek oraz regionów sterujących i zależnych IMS zgodnie z opisem w sekcji Konfigurowanie adaptera IMS.

**Ostrzeżenie:** Nie trzeba wykonywać kroku, który opisuje budowanie dynamicznego kodu pośredniczącego, chyba że jest on potrzebny do innych celów.

Po skonfigurowaniu adaptera IMS należy wykonać poniższą procedurę.

## Procedura

1. Zaktualizuj zmienną LIBPATH w podzbiorze biblioteki IMS PROCLIB, do której odwołuje się parametr ENVIRON w zależnym regionie JCL (na przykład DFSJVMEV), tak aby zawierała biblioteki rodzime IBM MQ classes for JMS .

Jest to katalog zFS , który zawiera `libmqjims.so`. Na przykład, DFSJVMEV może wyglądać następująco, gdzie ostatni wiersz jest katalogiem zawierającym rodzime biblioteki IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging :

```
LIBPATH=>
/java/latest/bin/j9vm:>
/java/latest/bin:>
/ims/latest/dbdc/imsjava/classic/lib:>
/ims/latest/dbdc/imsjava/lib:>
/mqm/latest/java/lib
```

2. Dodaj parametr IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging do ścieżki klasy maszyny JVM używanej przez region zależny IMS , aktualizując opcję `java.class.path` .

   W tym celu należy postępować zgodnie z instrukcjami zawartymi w sekcji Element DFSJVMMS zestawu danych IMS PROCLIB.

   Na przykład można użyć następującego tekstu, w którym pogrubiony wiersz wskazuje aktualizację:

V 9.3.0 JM 3.0 V 9.3.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.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/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.allclient.jar
```

   **Uwaga:** Chociaż w katalogu zawierającym pliki IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messagingdostępnych jest wiele różnych plików jar, potrzebne są tylko pliki `com.ibm.mq.allclient.jar` (JMS 2.0) lub `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0).
3. Zatrzymaj i zrestartuj wszystkie zależne regiony IMS , które będą korzystać z IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging.

## Co dalej

Utwórz i skonfiguruj fabryki połączeń i miejsca docelowe.

Istnieją trzy możliwe metody tworzenia instancji implementacji IBM MQ fabryk połączeń i miejsc docelowych. Szczegółowe informacje na ten temat zawiera sekcja “Tworzenie i konfigurowanie fabryk połączeń i miejsc docelowych” na stronie 211.

Należy zauważyć, że wszystkie te trzy podejścia są poprawne w środowisku IMS .

**Pojęcia pokrewne**

Konfigurowanie adaptera IMS

Definiowanie IBM MQ w pliku IMS

### *Zachowanie transakcyjne*

Komunikaty wysyłane i odbierane przez IBM MQ classes for JMS w środowisku IMS są zawsze powiązane z jednostką pracy (UOW) IMS , która jest aktywna w bieżącym zadaniu.

Tę jednostkę pracy można zakończyć tylko przez wywołanie metod zatwierdzania lub wycofywania zmian w instancji obiektu `com.ibm.ims.dli.tm.Transaction` lub przez normalne zakończenie zadania IMS , w którym to przypadku jednostka pracy jest niejawnie zatwierdzana. Jeśli zadanie IMS zostanie zakończone nieprawidłowo, jednostka pracy zostanie wycofana.

W wyniku tego wartości argumentów **`transacted`** i **`acknowledgeMode`** w razie wywołania dowolnej z metod `Connection.createSession` lub `ConnectionFactory.createContext` zostaną zignorowane. Ponadto nie są obsługiwane poniższe metody. Wywołanie dowolnej z następujących metod w przypadku sesji spowoduje wystąpienie wyjątku `IllegalStateException`:

- `javax.jms.Session.commit()`

- `javax.jms.Session.recover()`
- `javax.jms.Session.rollback()`

i `IllegalStateRuntimeSession` w przypadku kontekstu JMS:

- `javax.jms.JMSContext.commit()`
- `javax.jms.JMSContext.recover()`
- `javax.jms.JMSContext.rollback()`

Istnieje jeden wyjątek od tego zachowania. Jeśli sesja lub kontekst JMS jest tworzony za pomocą jednego z następujących mechanizmów:

- `Connection.createSession(false, Session.AUTO_ACKNOWLEDGE)`
- `Connection.createSession(Session.AUTO_ACKNOWLEDGE)`
- `ConnectionFactory.createContext(JMSContext.AUTO_ACKNOWLEDGE)`

to zachowanie tej sesji lub kontekstu JMS jest następujące:

- Wszystkie wysłane komunikaty są wysyłane poza jednostkę pracy IMS . Oznacza to, że będą one natychmiast dostępne w miejscu docelowym lub po zakończeniu podanego okresu opóźnienia dostawy.
- Wszystkie nietrwałe komunikaty będą odbierane poza jednostką pracy IMS , chyba że właściwość SYNCPOINTALLGETS została określona w fabryce połączeń, która utworzyła sesję lub kontekst JMS .
- Komunikaty trwałe będą zawsze odbierane w jednostce pracy IMS .

Może to być przydatne na przykład wtedy, gdy użytkownik chce zapisać komunikat kontroli w kolejce, nawet jeśli jednostka pracy wycofuje zmiany.

### *Implikacje punktów synchronizacji IMS*

Produkt IBM MQ classes for JMS jest oparty na istniejącej obsłudze adaptera IBM MQ , który korzysta z ESAF. Oznacza to, że udokumentowane zachowanie ma zastosowanie, w tym wszystkie otwarte uchwyty zamykane przez adapter IMS w przypadku wystąpienia punktu synchronizacji.

Więcej informacji zawiera sekcja “Punkty synchronizacji w aplikacjach IMS” na stronie 72.

Aby zilustrować ten punkt, należy rozważyć następujący kod działający w środowisku JMP. Drugie wywołanie metody `mp.send()` powoduje wywołanie metody `JMSException` , ponieważ kod `messageQueue.getUnique(inputMessage)` powoduje zamknięcie wszystkich otwartych połączeń IBM MQ i uchwytów obiektów.

Podobne zachowanie jest obserwowane, jeśli wywołanie `getUnique()` zostało zastąpione przez `Transaction.commit()`, ale nie w przypadku użycia funkcji `Transaction.rollback()` .

```
//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 JMSException containing a
//MQRC_HCONN_ERROR as the connection/handle has been closed.
mp.send(m);
```

Poprawny kod do użycia w tym scenariuszu jest następujący. W tym przypadku połączenie z serwerem IBM MQ jest zamykane przed wywołaniem metody `getUnique()`. Połączenie i sesja są następnie ponownie tworzone w celu wysłania kolejnego komunikatu.

```
//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);
```

### *Uwagi dotyczące używania adaptera IMS*

Należy pamiętać o następujących ograniczeniach. Dla każdego menedżera kolejek może istnieć tylko jeden uchwyt połączenia. Interakcja z produktem IBM MQ ma wpływ zarówno na użycie kodu JMS , jak i kodu rodzimego. Istnieją ograniczenia dotyczące uwierzytelniania i autoryzacji połączenia.

## Jeden uchwyt połączenia dla każdego menedżera kolejek

W danym momencie w regionach zależnych produktu IMS dozwolony jest tylko jeden uchwyt połączenia z konkretnym menedżerem kolejek. Wszystkie kolejne próby nawiązania połączenia z tym samym menedżerem kolejek będą ponownie wykorzystywać istniejący uchwyt.

To zachowanie nie powinno powodować żadnych problemów w aplikacji, która używa tylko produktu IBM MQ classes for JMS, ale może powodować problemy w aplikacjach, które wchodzą w interakcję z produktem IBM MQ, jeśli zarówno produkt IBM MQ classes for JMS , jak i produkt MQI są używane w kodzie rodzimym napisanym w językach, takich jak COBOL lub C.

## Implikacje interakcji z produktem IBM MQ w przypadku używania zarówno kodu JMS , jak i kodu rodzimego

Problemy mogą wystąpić podczas współdziałania kodu Java i kodu rodzimego, które korzystają z funkcji IBM MQ oraz gdy połączenie z serwerem IBM MQ nie zostanie zamknięte przed opuszczeniem kodu rodzimego lub kodu Java.

Na przykład w poniższym pseudokodzie uchwyt połączenia z menedżerem kolejek jest początkowo ustanawiany w kodzie Java za pomocą kodu IBM MQ classes for JMS. Uchwyt połączenia jest ponownie wykorzystywany w kodzie COBOL i unieważniany przez wywołanie MQDISC.

Następnym razem program IBM MQ classes for JMS użyje uchwytu połączenia `JMSException` z kodem przyczyny MQRC_HCONN_ERROR.

```
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
```

Istnieją inne podobne wzorce użycia, które mogą spowodować wystąpienie błędu MQRC_HCONN_ERROR.

Chociaż możliwe jest współużytkowanie uchwytów połączeń IBM MQ przez kod rodzimy i kod Java (na przykład poprzedni przykład zadziałałby, gdyby nie było wywołania MQDISC), sprawdzoną procedurą jest zamknięcie uchwytów połączeń przed przejściem z kodu Java na kod rodzimy lub odwrotnie.

## Uwierzytelnianie i autoryzacja połączenia

Specyfikacja JMS umożliwia podanie nazwy użytkownika i hasła na potrzeby uwierzytelniania i autoryzacji podczas tworzenia połączenia lub obiektu kontekstu JMS .

Nie jest to obsługiwane w środowisku IMS . Próba utworzenia połączenia podczas określania nazwy użytkownika i hasła spowoduje zgłoszenie zgłoszenia `JMS Exception` . Próba utworzenia kontekstu JMS podczas określania nazwy użytkownika i hasła spowoduje zgłoszenie `JMSRuntimeException` .

Zamiast tego należy użyć istniejących mechanizmów uwierzytelniania i autoryzacji podczas nawiązywania połączenia z produktem IBM MQ ze środowiska IMS .

Więcej informacji na ten temat zawiera sekcja Konfigurowanie zabezpieczeń w systemie z/OS.
W szczególności należy zapoznać się z sekcją Identyfikatory użytkowników do sprawdzania zabezpieczeń, w której opisano możliwe do użycia identyfikatory użytkowników.

**Zadania pokrewne**

Konfigurowanie zabezpieczeń w systemie z/OS

## *Ograniczenia interfejsu API JMS*

Z perspektywy specyfikacji JMS serwer IBM MQ classes for JMS traktuj IMS jako serwer aplikacji zgodny z Java EE lub Jakarta EE , który zawsze ma trwającą transakcję JTA.

Na przykład nigdy nie można wywołać metody `javax.jms.Session.commit()` w systemie IMS, ponieważ specyfikacja JMS określa, że nie można jej wywołać w komponencie EJB JEE ani w kontenerze WWW, podczas gdy transakcja JTA jest w toku.

Powoduje to następujące ograniczenia dotyczące interfejsu API JMS , oprócz tych opisanych w sekcji “Zachowanie transakcyjne” na stronie 353.

## Ograniczenia dotyczące klasycznego interfejsu API

- Metoda `javax.jms.Connection.createConnectionConsumer(javax.jms.Destination, String, javax.jms.ServerSessionPool, int)` zawsze zwraca wyjątek `JMSException`.
- Metoda `javax.jms.Connection.createDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` zawsze zwraca wyjątek `JMSException`.
- Wszystkie trzy warianty metody `javax.jms.Connection.createSession` zawsze zwracają wyjątek `JMSException`, jeśli dla połączenia już istnieje aktywna sesja.

- Metoda `javax.jms.Connection.createSharedConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` zawsze zwraca wyjątek `JMSException`.
- Metoda `javax.jms.Connection.createSharedDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` zawsze zwraca wyjątek `JMSException`.
- Metoda `javax.jms.Connection.setClientID()` zawsze zwraca wyjątek `JMSException`.
- Metoda `javax.jms.Connection.setExceptionListener(javax.jms.ExceptionListener)` zawsze zwraca wyjątek `JMSException`.
- Metoda `javax.jms.Connection.stop()` zawsze zwraca wyjątek `JMSException`.
- Metoda `javax.jms.MessageConsumer.setMessageListener(javax.jms.MessageListener)` zawsze zwraca wyjątek `JMSException`.
- Metoda `javax.jms.MessageConsumer.getMessageListener()` zawsze zwraca wyjątek `JMSException`.
- Metoda `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message,javax.jms.CompletionListener)` zawsze zwraca wyjątek `JMSException`.
- Metoda `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message, int, int, long, javax.jms.CompletionListener)` zawsze zwraca wyjątek `JMSException`.
- Metoda `javax.jms.MessageProducer.send(javax.jms.Message, int, int, long, javax.jms.CompletionListener)` zawsze zwraca wyjątek `JMSException`.
- Metoda `javax.jms.MessageProducer.send(javax.jms.Message,javax.jms.CompletionListener )` zawsze zwraca wyjątek `JMSException`.
- Metoda `javax.jms.Session.run()` zawsze zwraca wyjątek `JMSRuntimeException`.
- Metoda `javax.jms.Session.setMessageListener(javax.jms.MessageListener)` zawsze zwraca wyjątek `JMSException`.
- Metoda `javax.jms.Session.getMessageListener()` zawsze zwraca wyjątek `JMSException`.

### Ograniczenia uproszczonego interfejsu API

- Metoda `javax.jms.JMSContext.createContext(int)` zawsze zwraca wyjątek `JMSRuntimeException`.
- Metoda `javax.jms.JMSContext.setClientID(String)` zawsze zwraca wyjątek `JMSRuntimeException`.
- Metoda `javax.jms.JMSContext.setExceptionListener(javax.jms.ExceptionListener)` zawsze zwraca wyjątek `JMSRuntimeException`.
- Metoda `javax.jms.JMSContext.stop()` zawsze zwraca wyjątek `JMSRuntimeException`.
- Metoda `javax.jms.JMSProducer.setAsync(javax.jms.CompletionListener)` zawsze zwraca wyjątek `JMSRuntimeException`.

## użycieIBM MQ classes for Java

Użyj IBM MQ w środowisku Java . IBM MQ classes for Java Zezwalaj aplikacji Java na nawiązywanie połączenia z produktem IBM MQ jako klientem IBM MQ lub nawiązywanie połączenia bezpośrednio z menedżerem kolejek produktu IBM MQ .

**Uwaga:**

Stabilized Produkt IBM nie zawiera żadnych dodatkowych udoskonaleń w produkcie IBM MQ classes for Java i są one funkcjonalnie ustabilizowane na poziomie dostarczanym wraz z produktem IBM MQ 8.0. Istniejące aplikacje korzystające z produktu IBM MQ classes for Java nadal będą w pełni obsługiwane, ale

nowe funkcje nie zostaną dodane, a żądania rozszerzeń zostaną odrzucone. W pełni obsługiwane oznacza, że defekty zostaną usunięte wraz ze wszystkimi zmianami wymaganymi przez zmiany w wymaganiach systemowych produktu IBM MQ .

IBM MQ classes for Java nie są obsługiwane w systemie IMS.

IBM MQ classes for Java nie są obsługiwane w systemie WebSphere Liberty. Nie można ich używać z funkcją przesyłania komunikatów produktu IBM MQ Liberty ani z ogólną obsługą języka JCA . Więcej informacji na ten temat zawiera sekcja Używanie interfejsów Java produktu WebSphere MQ w środowiskach J2EE/JEE.

IBM MQ classes for Java jest jednym z trzech alternatywnych interfejsów API, które mogą być używane przez aplikacje Java do uzyskiwania dostępu do zasobów IBM MQ . Inne interfejsy API to:

- 

IBM MQ classes for Jakarta Messaging
- 

IBM MQ classes for JMS

Więcej informacji na ten temat zawiera “Uzyskiwanie dostępu do produktu IBM MQ z poziomu programu Java -wybór interfejsu API” na stronie 89.

Od wersji IBM MQ 9.3 IBM MQ classes for Java są budowane z użyciem języka Java 8. Środowisko wykonawcze programów Java 8 obsługuje uruchamianie wcześniejszych wersji plików klas.

Produkt IBM MQ classes for Java hermetyzuje interfejs MQI (Message Queue Interface), rodzimy interfejs API języka IBM MQ i używa modelu obiektowego podobnego do interfejsu C++ i .NET dla produktu IBM MQ.

Opcje programowalne umożliwiają programowi IBM MQ classes for Java nawiązanie połączenia z programem IBM MQ w jeden z następujących sposobów:

- W trybie klienta jako IBM MQ MQI client przy użyciu protokołu Transmission Control Protocol/Internet Protocol (TCP/IP)
- W trybie powiązańbezpośrednie połączenie z produktem IBM MQ przy użyciu rodzimego interfejsu języka Java (JNI)

**Uwaga:** Automatyczne ponowne nawiązywanie połączenia przez klient nie jest obsługiwane przez produkt IBM MQ classes for Java.

## połączenie w trybie klienta

Aplikacja IBM MQ classes for Java może nawiązać połączenie z dowolnym obsługiwanym menedżerem kolejek przy użyciu trybu klienta.

Aby nawiązać połączenie z menedżerem kolejek w trybie klienta, aplikacja programu IBM MQ classes for Java może działać w tym samym systemie, w którym działa menedżer kolejek, lub w innym systemie. W każdym przypadku program IBM MQ classes for Java łączy się z menedżerem kolejek za pośrednictwem protokołu TCP/IP.

Więcej informacji na temat pisania aplikacji korzystających z połączeń w trybie klienta zawiera sekcja “Tryby połączenia IBM MQ classes for Java” na stronie 384.

## połączenie w trybie powiązań

W przypadku użycia w trybie powiązań produkt IBM MQ classes for Java używa rodzimego interfejsu języka Java (JNI) do wywoływania bezpośrednio w istniejącym interfejsie API menedżera kolejek, zamiast komunikowania się w sieci. W większości środowisk połączenie w trybie powiązań zapewnia lepszą wydajność aplikacji IBM MQ classes for Java niż połączenie w trybie klienta, dzięki uniknięciu kosztów komunikacji TCP/IP.

Aplikacje używające programu IBM MQ classes for Java do nawiązywania połączeń w trybie powiązań muszą działać w tym samym systemie co menedżer kolejek, z którym nawiązywane jest połączenie.

Środowisko Java Runtime Environment, które jest używane do uruchamiania aplikacji IBM MQ classes for Java , musi być skonfigurowane do ładowania bibliotek IBM MQ classes for Java . Więcej informacji na ten temat zawiera sekcja “IBM MQ classes for Java biblioteki” na stronie 369 .

Więcej informacji na temat pisania aplikacji używających połączeń w trybie powiązań zawiera sekcja “Tryby połączenia IBM MQ classes for Java” na stronie 384.

**Pojęcia pokrewne**

Interfejsy języka IBM MQ Java

“Korzystanie z IBM MQ classes for JMS/Jakarta Messaging” na stronie 84
IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging są dostawcami przesyłania komunikatów produktu Java dostarczanymi wraz z produktem IBM MQ. Oprócz implementowania interfejsów zdefiniowanych w specyfikacjach JMS i Jakarta Messaging dostawcy przesyłania komunikatów dodają dwa zestawy rozszerzeń do interfejsu API przesyłania komunikatów produktu Java .

**Zadania pokrewne**

Śledzenie aplikacji IBM MQ classes for Java

Rozwiązywanie problemów z systemem Java i JMS

## Dlaczego należy używać IBM MQ classes for Java?

Aplikacja Java może korzystać z IBM MQ classes for Java lub IBM MQ classes for JMS w celu uzyskania dostępu do zasobów IBM MQ .

**Uwaga:** V 9.3.0 V 9.3.0 Mimo że istniejące aplikacje korzystające z produktu IBM MQ classes for Java nadal są w pełni obsługiwane, nowe aplikacje powinny używać produktu IBM MQ classes for Jakarta Messaging. Funkcje, które zostały ostatnio dodane do produktu IBM MQ, takie jak wykorzystanie asynchroniczne i automatyczne ponowne połączenie, nie są dostępne w produkcie IBM MQ classes for Java, ale są dostępne w produkcie IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging. Więcej informacji na ten temat zawierają sekcje “Dlaczego należy używać IBM MQ classes for JMS?” na stronie 87 i “Dlaczego należy używać IBM MQ classes for Jakarta Messaging?” na stronie 86.

**Uwaga:**

Stabilized Produkt IBM nie zawiera żadnych dodatkowych udoskonaleń w produkcie IBM MQ classes for Java i są one funkcjonalnie ustabilizowane na poziomie dostarczanym wraz z produktem IBM MQ 8.0. Istniejące aplikacje korzystające z produktu IBM MQ classes for Java nadal będą w pełni obsługiwane, ale nowe funkcje nie zostaną dodane, a żądania rozszerzeń zostaną odrzucone. W pełni obsługiwane oznacza, że defekty zostaną usunięte wraz ze wszystkimi zmianami wymaganymi przez zmiany w wymaganiach systemowych produktu IBM MQ .

IBM MQ classes for Java nie są obsługiwane w systemie IMS.

IBM MQ classes for Java nie są obsługiwane w systemie WebSphere Liberty. Nie można ich używać z funkcją przesyłania komunikatów produktu IBM MQ Liberty ani z ogólną obsługą języka JCA . Więcej informacji na ten temat zawiera sekcja Używanie interfejsów Java produktu WebSphere MQ w środowiskach J2EE/JEE.

**Pojęcia pokrewne**

“Uzyskiwanie dostępu do produktu IBM MQ z poziomu programu Java -wybór interfejsu API” na stronie 89
IBM MQ udostępnia trzy interfejsy języka Java .

## Wymagania wstępne dla produktu IBM MQ classes for Java

Do korzystania z produktu IBM MQ classes for Javapotrzebne są inne produkty oprogramowania.

Informacje na temat wymagań wstępnych dla systemu IBM MQ classes for Javamożna znaleźć na stronie WWW Wymagania systemowe produktu IBM MQ .

Do tworzenia aplikacji IBM MQ classes for Java potrzebny jest pakiet Java Development Kit (JDK). Szczegółowe informacje na temat pakietów JDK obsługiwanych w systemie operacyjnym można znaleźć w sekcji Wymagania systemowe produktu IBM MQ .

Do uruchamiania aplikacji IBM MQ classes for Java potrzebne są następujące komponenty oprogramowania:

- Menedżer kolejek systemu IBM MQ dla aplikacji, które łączą się z menedżerem kolejek
- Środowisko Java Runtime Environment (JRE) dla każdego systemu, w którym uruchamiane są aplikacje. Odpowiednie środowisko JRE jest dostarczane z produktem IBM MQ.
- IBM i Dla systemu IBM i, QShell, który jest opcją 30 systemu operacyjnego
- z/OS W systemie z/OS: z/OS UNIX System Services (z/OS UNIX)

Jeśli do korzystania z modułów szyfrujących, które mają certyfikat FIPS 140-2, wymagane są połączenia TLS, potrzebny jest dostawca IBM Java JSSE FIPS (IBMJSSEFIPS). Każdy pakiet IBM JDK i środowisko JRE w wersji 1.4.2 lub nowszej zawiera IBMJSSEFIPS.

W aplikacjach IBM MQ classes for Java można używać adresów protokołu Internet Protocol w wersji 6 (IPv6) , jeśli IPv6 jest obsługiwane przez wirtualną maszynę języka Java (JVM) i implementację protokołu TCP/IP w systemie operacyjnym.

## Uruchamianie aplikacji IBM MQ classes for Java w produkcie Java EE

Istnieją pewne ograniczenia i uwagi dotyczące projektu, które należy wziąć pod uwagę przed użyciem produktu IBM MQ classes for Java w produkcie Java EE.

Produkt IBM MQ classes for Java ma ograniczenia, jeśli jest używany w środowisku Java Platform, Enterprise Edition (Java EE). Istnieją również dodatkowe uwagi, które należy wziąć pod uwagę podczas projektowania, implementowania i zarządzania aplikacją IBM MQ classes for Java działającą w środowisku Java EE . Te ograniczenia i uwagi zostały opisane w poniższych sekcjach.

### Ograniczenia transakcji JTA

Jedynym obsługiwanym menedżerem transakcji dla aplikacji korzystających z produktu IBM MQ classes for Java jest sam produkt IBM MQ . Chociaż aplikacja pod kontrolą JTA może korzystać z języka IBM MQ classes for Java, żadna praca wykonywana za pomocą tych klas nie jest kontrolowana przez jednostki pracy JTA. Zamiast tego tworzą one lokalne jednostki pracy oddzielone od jednostek zarządzanych przez serwer aplikacji za pośrednictwem interfejsów JTA. W szczególności wycofanie transakcji JTA nie powoduje wycofania żadnych wysłanych lub odebranych komunikatów. To ograniczenie dotyczy transakcji zarządzanych przez aplikację lub komponent bean, transakcji zarządzanych przez kontener i wszystkich kontenerów Java EE . Aby przesyłanie komunikatów działało bezpośrednio z produktem IBM MQ wewnątrz transakcji koordynowanych przez serwer aplikacji, należy zamiast niego użyć produktu IBM MQ classes for JMS .

### Tworzenie wątku

IBM MQ classes for Java tworzy wątki wewnętrznie dla różnych operacji. Na przykład w przypadku uruchamiania w trybie BINDINGS w celu wywołania bezpośrednio w lokalnym menedżerze kolejek wywołania są wykonywane w wątku roboczym utworzonym wewnętrznie przez program IBM MQ classes for Java. Inne wątki mogą być tworzone wewnętrznie, na przykład w celu usunięcia nieużywanych połączeń z puli połączeń lub usunięcia subskrypcji dla zakończonych aplikacji publikowania/subskrybowania.

Niektóre aplikacje Java EE (na przykład działające w kontenerach EJB i WWW) nie mogą tworzyć nowych wątków. Zamiast tego cała praca musi być wykonywana na głównych wątkach aplikacji zarządzanych przez serwer aplikacji. Jeśli aplikacje korzystają z produktu IBM MQ classes for Java, serwer aplikacji może nie być w stanie odróżnić kodu aplikacji od kodu IBM MQ classes for Java , dlatego opisane wcześniej wątki powodują, że aplikacja nie jest zgodna ze specyfikacją kontenera. IBM MQ classes for JMS nie przerywa tych specyfikacji Java EE i dlatego można ich używać zamiast nich.

## Ograniczenia bezpieczeństwa

Strategie bezpieczeństwa zaimplementowane przez serwer aplikacji mogą uniemożliwić wykonywanie pewnych operacji przez interfejs API IBM MQ classes for Java , takich jak tworzenie i obsługa nowych wątków sterowania (zgodnie z opisem w poprzednich sekcjach).

Na przykład serwery aplikacji są zwykle uruchamiane z domyślnie wyłączonymi zabezpieczeniami Java i umożliwiają ich włączenie w konfiguracji specyficznej dla serwera aplikacji (niektóre serwery aplikacji umożliwiają również bardziej szczegółową konfigurację strategii używanych w ramach zabezpieczeń Java ). Jeśli zabezpieczenia Java są włączone, IBM MQ classes for Java może złamać reguły wielowątkowości strategii zabezpieczeń Java zdefiniowane dla serwera aplikacji, a interfejs API może nie być w stanie utworzyć wszystkich wątków potrzebnych do działania. Aby zapobiec problemom z zarządzaniem wątkami, użycie opcji IBM MQ classes for Java nie jest obsługiwane w środowiskach, w których włączone są zabezpieczenia Java .

## Uwagi dotyczące odseparowania aplikacji

Zamierzoną korzyścią z uruchamiania aplikacji w środowisku Java EE jest izolacja aplikacji. Projekt i implementacja produktu IBM MQ classes for Java poprzedzają środowisko Java EE . Parametr IBM MQ classes for Java może być używany w sposób, który nie obsługuje izolowania aplikacji. Konkretne przykłady rozważań w tej dziedzinie obejmują:

- Użycie ustawień statycznych (dotyczących całego procesu JVM) w klasie MQEnvironment, takich jak:
  - ID użytkownika i hasło, które mają być używane do identyfikacji i uwierzytelniania połączenia.
  - Nazwa hosta, port i kanał używane dla połączeń klienckich
  - Konfiguracja TLS dla chronionych połączeń klienckich

  Modyfikowanie dowolnych właściwości środowiska MQEnvironment z korzyścią dla jednej aplikacji ma również wpływ na inne aplikacje korzystające z tych samych właściwości. W przypadku uruchamiania w środowisku z wieloma aplikacjami, takim jak Java EE, każda aplikacja musi używać własnej odrębnej konfiguracji, tworząc obiekty MQQueueManager z konkretnym zestawem właściwości, zamiast używać właściwości skonfigurowanych w klasie MQEnvironment dla całego procesu.
- Klasa MQEnvironment wprowadza wiele metod statycznych, które działają globalnie na wszystkich aplikacjach korzystających z języka IBM MQ classes for Java w ramach tego samego procesu maszyny JVM i nie ma możliwości przesłonięcia tego zachowania dla konkretnych aplikacji. Przykłady takich ograniczeń to:
  - konfigurowanie właściwości TLS, takich jak położenie magazynu kluczy
  - konfigurowanie wyjść kanału klienta
  - włączanie lub wyłączanie śledzenia diagnostycznego
  - zarządzanie domyślną pulą połączeń używaną do optymalizacji użycia połączeń z menedżerami kolejek

  Wywoływanie takich metod ma wpływ na wszystkie aplikacje działające w tym samym środowisku Java EE .
- Zestawianie połączeń jest włączone w celu zoptymalizowania procesu nawiązywania wielu połączeń z tym samym menedżerem kolejek. Domyślny menedżer puli połączeń jest w całym procesie i jest współużytkowany przez wiele aplikacji. Zmiany w konfiguracji puli połączeń, takie jak zastąpienie domyślnego menedżera połączeń dla jednej aplikacji przy użyciu metody MQEnvironment.setDefaultConnectionManager(), mają wpływ na inne aplikacje działające na tym samym serwerze aplikacji Java EE.
- Protokół TLS jest skonfigurowany dla aplikacji korzystających z produktu IBM MQ classes for Java przy użyciu klasy MQEnvironment i właściwości obiektu MQQueueManager . Nie jest on zintegrowany z konfiguracją zabezpieczeń zarządzanych samego serwera aplikacji. Należy upewnić się, że produkt IBM MQ classes for Java został odpowiednio skonfigurowany, aby zapewnić wymagany poziom zabezpieczeń, a nie używać konfiguracji serwera aplikacji.

### Ograniczenia trybu powiązań

Produkty IBM MQ i WebSphere Application Server można zainstalować na tym samym komputerze, tak aby wersje główne menedżera kolejek i adaptera zasobów IBM MQ (RA) dostarczanego z produktem WebSphere Application Server były różne.

Jeśli wersje główne menedżera kolejek i adaptera zasobów są różne, nie można używać połączeń powiązań. Wszystkie połączenia z produktu WebSphere Application Server do menedżera kolejek przy użyciu adaptera zasobów muszą używać połączeń typu klient. Połączenia powiązań mogą być używane, jeśli wersje są takie same.

## Konwersje łańcuchów znaków w programie IBM MQ classes for Java

IBM MQ classes for Java CharsetEncoders i CharsetDecoders są używane bezpośrednio do konwersji łańcuchów znaków. Domyślne zachowanie konwersji łańcucha znaków można skonfigurować przy użyciu dwóch właściwości systemowych. Obsługę komunikatów, które zawierają znaki niemożliwe do odwzorowania, można skonfigurować za pomocą programu `com.ibm.mq.MQMD`.

Przed wersją IBM MQ 8.0konwersje łańcuchów w pliku IBM MQ classes for Java były wykonywane przez wywołanie metod `java.nio.charset.Charset.decode(ByteBuffer)` i `Charset.encode(CharBuffer)` .

Użycie jednej z tych metod powoduje domyślne zastąpienie ( `REPLACE`) zniekształconych lub niemożliwych do przetłumaczenia danych. Takie zachowanie może przesłaniać błędy w aplikacjach i prowadzić do nieoczekiwanych znaków, na przykład `?`, w przetłumaczonych danych.

W programie IBM MQ 8.0, aby wykryć takie problemy wcześniej i bardziej efektywnie, IBM MQ classes for Java należy użyć CharsetEncoders i CharsetDecoders bezpośrednio i skonfigurować obsługę zniekształconych i niemożliwych do przetłumaczenia danych w sposób jawny. Domyślnym zachowaniem jest `REPORT` takie problemy przez zgłoszenie odpowiedniego `MQException`.

### Konfigurowanie

Translacja z formatu UTF-16 (reprezentacja znaków używana w Java) na rodzimy zestaw znaków, taki jak UTF-8, jest zwana *kodowaniem*, natomiast translacja w przeciwnym kierunku jest zwana *dekodowaniem*.

Dekodowanie przyjmuje domyślne zachowanie programu `CharsetDecoders`i zgłasza błędy, zgłaszając wyjątek.

Jedno ustawienie służy do określenia parametru `java.nio.charset.CodingErrorAction` , który steruje obsługą błędów zarówno przy kodowaniu, jak i dekodowaniu. Inne ustawienie jest używane do sterowania bajtem zastępującym (lub bajtami) podczas kodowania. W operacjach dekodowania zostanie użyty domyślny łańcuch zastępujący Java .

### Konfigurowanie obsługi danych, których nie można przetłumaczyć, w produkcie IBM MQ classes for Java

W produkcie IBM MQ 8.0produkt `com.ibm.mq.MQMD` zawiera następujące dwa pola:

**byte [] unMappableWymiana**
Sekwencja bajtów, która zostanie zapisana w zakodowanym łańcuchu, jeśli nie można przetłumaczyć znaku wejściowego, a podano parametr `REPLACE`.

**Wartość domyślna: "?".getBytes()**
W operacjach dekodowania używany jest domyślny łańcuch zastępujący Java .

**java.nio.charset.CodingErrorAction unmappableAction**
Określa działanie, które ma zostać podjęte dla niepodlegających tłumaczeniu danych przy kodowaniu i dekodowaniu:

**Wartość domyślna: CodingErrorAction.REPORT;**

### Właściwości systemowe służące do ustawiania systemowych wartości domyślnych

W produkcie IBM MQ 8.0dostępne są następujące dwie właściwości systemowe Java umożliwiające skonfigurowanie domyślnego zachowania w odniesieniu do konwersji łańcuchów znaków.

**com.ibm.mq.cfg.jmqi.UnmappableCharacterAction**
Określa działanie, które ma zostać podjęte dla niepodlegających tłumaczeniu danych przy kodowaniu i dekodowaniu. Możliwe wartości to REPORT, REPLACElub IGNORE.

**com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement**
Ustawia lub pobiera bajty zastępujące, które mają być stosowane, gdy nie można odwzorować znaku w operacji kodowania. W operacjach dekodowania używany jest domyślny łańcuch zastępujący Java .

Aby uniknąć pomyłek między reprezentacjami znaków Java i bajtów rodzimych, należy określić wartość `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement` jako liczbę dziesiętną reprezentującą bajt zastępujący w rodzimym zestawie znaków.

Na przykład wartość dziesiętna ?, jako bajt rodzimy, wynosi 63, jeśli rodzimy zestaw znaków jest oparty na kodzie ASCII, na przykład ISO-8859-1, a wartość 111, jeśli rodzimy zestaw znaków to EBCDIC.

**Uwaga:** Należy zauważyć, że jeśli obiekt MQMD lub MQMessage ma ustawione pola **unmappableAction** lub **unMappableReplacement** , to wartości tych pól mają pierwszeństwo przed właściwościami systemu Java . Umożliwia to nadpisanie wartości określonych przez właściwości systemowe Java dla każdego komunikatu, jeśli jest to wymagane.

**Pojęcia pokrewne**
“Konwersje łańcuchów znaków w programie IBM MQ classes for JMS” na stronie 143
IBM MQ classes for JMS CharsetEncoders i CharsetDecoders są używane bezpośrednio do konwersji łańcuchów znaków. Domyślne zachowanie konwersji łańcucha znaków można skonfigurować przy użyciu dwóch właściwości systemowych. Obsługę komunikatów, które zawierają znaki niemożliwe do odwzorowania, można skonfigurować za pomocą właściwości komunikatu w celu ustawienia działania UnmappableCharacteri bajtów zastępujących.

## Instalowanie i konfigurowanie produktu IBM MQ classes for Java

W tej sekcji opisano katalogi i pliki, które są tworzone podczas instalowania produktu IBM MQ classes for Java, oraz opisano sposób konfigurowania produktu IBM MQ classes for Java po instalacji.

### *Co jest instalowane w systemie IBM MQ classes for Java*

Najnowsza wersja produktu IBM MQ classes for Java jest instalowana razem z produktem IBM MQ. Aby to zrobić, może być konieczne nadpisanie domyślnych opcji instalacji.

Więcej informacji na temat instalowania produktu IBM MQ zawiera sekcja:

- Multi Instalowanie produktu IBM MQ
- z/OS Instalowanie produktu IBM MQ for z/OS

IBM MQ classes for Java znajdują się w plikach archiwum Java (JAR), `com.ibm.mq.jar`i `com.ibm.mq.jmqi.jar`.

Obsługa standardowych nagłówków komunikatów, takich jak Programmable Command Format (PCF), jest zawarta w pliku JAR `com.ibm.mq.headers.jar`.

Obsługa formatu Programmable Command Format (PCF) jest zawarta w pliku JAR `com.ibm.mq.pcf.jar`.

**Uwaga:** Nie zaleca się używania pliku IBM MQ classes for Java na serwerze aplikacji. Informacje na temat ograniczeń, które mają zastosowanie podczas uruchamiania w tym środowisku, zawiera sekcja “Uruchamianie aplikacji IBM MQ classes for Java w produkcie Java EE” na stronie 360. Więcej informacji na ten temat zawiera sekcja Używanie interfejsów produktu WebSphere MQ Java w środowiskach J2EE/JEE.

**Ważne:** Oprócz przemieszczalnych plików JAR opisanych w sekcji “Przemieszczalne pliki JAR IBM MQ classes for Java” na stronie 364, kopiowanie plików JAR IBM MQ classes for Java lub bibliotek rodzimych do innych komputerów lub do innego położenia na komputerze, na którym zainstalowano plik IBM MQ classes for Java , nie jest obsługiwane.

*Przemieszczalne pliki JAR IBM MQ classes for Java*
Przemieszczalne pliki JAR można przenieść do systemów, w których musi być uruchomiony program IBM MQ classes for Java.

**Ważne:**

- Oprócz przemieszczalnych plików JAR opisanych w sekcji Relocatable JAR files, kopiowanie plików JAR IBM MQ classes for Java lub bibliotek rodzimych do innych komputerów lub do innego położenia na komputerze, na którym zainstalowano IBM MQ classes for Java , nie jest obsługiwane.
- Aby uniknąć konfliktów programu ładującego klasy, nie zaleca się tworzenia pakunków przemieszczalnych plików JAR w obrębie wielu aplikacji w obrębie tego samego środowiska wykonawczego Java . W tym scenariuszu należy rozważyć udostępnienie przemieszczalnych plików JAR IBM MQ w ścieżce klasy środowiska wykonawczego Java .
- Nie należy dołączać przemieszczalnych plików JAR do aplikacji wdrożonych na serwerach aplikacji Java EE , takich jak WebSphere Application Server. W tych środowiskach adapter zasobów IBM MQ powinien zostać wdrożony i użyty, ponieważ zawiera on IBM MQ classes for Java. Należy zauważyć, że WebSphere Application Server osadza adapter zasobów IBM MQ , więc nie ma potrzeby wdrażania go ręcznie w tym środowisku. Oprócz tego produkt IBM MQ classes for Java nie jest obsługiwany w produkcie WebSphere Liberty. Więcej informacji na ten temat zawiera sekcja “Liberty i adapter zasobów IBM MQ” na stronie 456.
- Jeśli relokowalne pliki JAR są umieszczane w pakunku aplikacji, należy upewnić się, że zostały uwzględnione wszystkie wstępnie wymagane pliki JAR zgodnie z opisem w sekcji Relocatable JAR files(Pliki JAR przemieszczalne). Należy również upewnić się, że dostępne są odpowiednie procedury aktualizacji dołączonych plików JAR w ramach konserwacji aplikacji, aby zapewnić, że plik IBM MQ classes for Java pozostanie aktualny i znane problemy będą ponownie mediowane.

## Przemieszczalne pliki JAR

W przedsiębiorstwie następujące pliki można przenieść do systemów, w których mają być uruchamiane aplikacje IBM MQ classes for Java :

- JMS 2.0 `com.ibm.mq.allclient.jar` “1” na stronie 365
- V 9.3.0 JM 3.0 V 9.3.0 `com.ibm.mq.jakarta.client.jar` “2” na stronie 365
- V 9.3.3 Removed `com.ibm.mq.traceControl.jar`
- V 9.3.5 `bcpkix-jdk18on.jar` “3” na stronie 365
- V 9.3.0 V 9.3.0 `bcpkix-jdk15to18.jar` “4” na stronie 365
- V 9.3.5 `bcprov-jdk18on.jar` “3” na stronie 365
- V 9.3.0 V 9.3.0 `bcprov-jdk15to18.jar` “4” na stronie 365
- V 9.3.5 `bcutil-jdk18on.jar` “3” na stronie 365
- V 9.3.0 V 9.3.0 `bcutil-jdk15to18.jar` “4” na stronie 365
- V 9.3.3 `jackson-annotations.jar`
- V 9.3.3 `jackson-core.jar`
- V 9.3.3 `jackson-databind.jar`
- `org.json.jar`

**Uwagi:**

1. JMS 2.0 i JMS 1.1
2. Jakarta Messaging 3.0
3. Continuous Delivery z wersji IBM MQ 9.3.5
4. Long Term Support i Continuous Delivery przed IBM MQ 9.3.5

## Dostawca zabezpieczeń bouncy Castle i pliki JAR obsługi CMS

Dostawca zabezpieczeń Bouncy Castle i pliki JAR obsługi CMS są wymagane. Więcej informacji na ten temat zawiera serwis Support for non-IBM JREs with AMS.

V 9.3.5 W przypadku produktu Continuous Delivery z produktu IBM MQ 9.3.5wymagane są następujące pliki JAR:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

LTS W systemach Long Term Support i Continuous Delivery przed IBM MQ 9.3.5wymagane są następujące pliki JAR:

- `bcpkix-jdk15to18.jar`
- `bcprov-jdk15to18.jar`
- `bcutil-jdk15to18.jar`

## org.json.jar

Plik `org.json.jar` jest wymagany, jeśli aplikacja IBM MQ classes for Java używa tabeli definicji kanału klienta w formacie JSON.

## com.ibm.mq.allclient.jar i com.ibm.mq.jakarta.client.jar

Pliki `com.ibm.mq.allclient.jar` i `com.ibm.mq.jakarta.client.jar` zawierają klasy IBM MQ classes for JMS, IBM MQ classes for Javaoraz PCF i Headers. Jeśli te pliki zostaną przeniesione do nowego położenia, należy upewnić się, że zostały one zachowane przy użyciu nowych pakietów poprawek IBM MQ . Należy również upewnić się, że użycie plików jest znane działowi wsparcia IBM w przypadku uzyskania poprawki tymczasowej.

Aby określić wersję pliku `com.ibm.mq.allclient.jar` lub pliku `com.ibm.mq.jakarta.client.jar` , użyj następującej komendy:

V 9.3.0 JM 3.0 V 9.3.0

```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
java -jar com.ibm.mq.allclient.jar
```

W poniższym przykładzie przedstawiono przykładowe dane wyjściowe tej komendy:

```
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
```

## jackson-annotations.jar, jackson-core.jar i jackson-databind.jar

V 9.3.3

Trzy pliki JAR Jackson są wymagane, jeśli aplikacja IBM MQ classes for Java tworzy bezpieczne połączenia TLS z menedżerem kolejek.

*Katalogi instalacyjne produktu IBM MQ classes for Java*
Pliki IBM MQ classes for Java i przykłady są instalowane w różnych miejscach w zależności od platformy. Położenie środowiska Java Runtime Environment (JRE), które jest instalowane z produktem IBM MQ , również różni się w zależności od platformy.

## Katalogi instalacyjne dla plików IBM MQ classes for Java

Tabela 49 na stronie 366 przedstawia miejsce, w którym są zainstalowane pliki IBM MQ classes for Java .

*Tabela 49. IBM MQ classes for Java Katalogi instalacyjne*

| Platforma | Katalog |
|---|---|
| AIX AIX | *MQ_INSTALLATION_PATH*/java/lib |
| | /QIBM/ProdData/mqm/java/lib |
| Linux Linux | *MQ_INSTALLATION_PATH*/java/lib |
| Windows Windows | *MQ_INSTALLATION_PATH*\java\lib |
| z/OS z/OS | *MQ_INSTALLATION_PATH*/mqm/V8R0M0/java /lib |

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

## Katalogi instalacyjne dla przykładów

Niektóre aplikacje przykładowe, takie jak programy weryfikacji instalacji (IVP), są dostarczane wraz z produktem IBM MQ. Tabela 50 na stronie 366 przedstawia miejsce, w którym są zainstalowane aplikacje przykładowe. Przykłady IBM MQ classes for Java znajdują się w podkatalogu o nazwie `wmqjava`. Przykłady PCF znajdują się w podkatalogu o nazwie `pcf`.

*Tabela 50. Katalogi przykładów*

| Platforma | Katalog |
|---|---|
| AIX AIX | *MQ_INSTALLATION_PATH*/samp/wmqjava/ |
| IBM i IBM i | /QIBM/ProdData/mqm/java/samples |

| Tabela 50. Katalogi przykładów (kontynuacja) | |
|---|---|
| **Platforma** | **Katalog** |
| Linux | *MQ_INSTALLATION_PATH*/samp/wmqjava/ |
| Windows | *MQ_INSTALLATION_PATH*\tools\wmqjava\ |
| z/OS | *MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/samples |

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

## Katalogi instalacyjne środowiska JRE

IBM MQ classes for JMS wymaga środowiska Java 7 (lub nowszego) Java Runtime Environment (JRE). Odpowiednie środowisko JRE jest instalowane z produktem IBM MQ. Tabela 51 na stronie 367 pokazuje, gdzie jest zainstalowane środowisko JRE. Aby uruchomić programy Java , takie jak podane przykłady, za pomocą tego środowiska JRE jawnie wywołaj komendę *JRE_LOCATION*`/bin/java` lub dodaj *JRE_LOCATION*`/bin` do zmiennej środowiskowej PATH (lub jej odpowiednika) dla danej platformy, gdzie *JRE_LOCATION* jest katalogiem podanym w zmiennej Tabela 51 na stronie 367.

| Tabela 51. Katalogi JRE | |
|---|---|
| **Platforma** | **Katalog** |
| 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 |
| z/OS | *MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/jre |

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

*Zmienne środowiskowe dotyczące systemu IBM MQ classes for Java*
Jeśli mają być uruchamiane aplikacje IBM MQ classes for Java , ich ścieżka klasy musi zawierać katalogi IBM MQ classes for Java i samples.

Aby aplikacje IBM MQ classes for Java mogły działać, ich ścieżka klasy musi zawierać odpowiedni katalog IBM MQ classes for Java . Aby uruchomić przykładowe aplikacje, ścieżka klasy musi również zawierać odpowiednie katalogi przykładów. Te informacje można podać w komendzie wywołania Java lub w zmiennej środowiskowej **CLASSPATH** .

**Ważne:** Ustawienie Java opcji `-Xbootclasspath` w celu uwzględnienia IBM MQ classes for Java nie jest obsługiwane.

Tabela 52 na stronie 368 przedstawia odpowiednie ustawienie **CLASSPATH** , które ma być używane na każdej platformie do uruchamiania aplikacji IBM MQ classes for Java , w tym aplikacji przykładowych.

| *Tabela 52. Ustawienie **CLASSPATH** umożliwiające uruchamianie aplikacji IBM MQ classes for Java , w tym aplikacji przykładowych IBM MQ classes for Java* | |
|---|---|
| **Platforma** | **CLASSPATH ustawienie** |
| 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; |
| z/OS | CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/samples/wmqjava: *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/samples/pcf |

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

W przypadku kompilacji za pomocą opcji `-Xlint` może zostać wyświetlone ostrzeżenie informujące o braku opcji `com.ibm.mq.ese.jar` . Ostrzeżenie można zignorować. Ten plik jest dostępny tylko wtedy, gdy zainstalowano produkt Advanced Message Security.

Skrypty dostarczone z programem IBM MQ classes for JMS używają następujących zmiennych środowiskowych:

**ŚCIEŻKA_DANYCH MQ_JAVA_data**
Ta zmienna środowiskowa określa katalog dla danych wyjściowych dziennika i śledzenia.

**ŚCIEŻKA_INSTALACJI_MQJAVA**
Ta zmienna środowiskowa określa katalog, w którym zainstalowano produkt IBM MQ classes for Java , jak to pokazano w katalogach instalacyjnych produktuIBM MQ classes for Java.

**ŚCIEŻKA_BIBLIOTEKI_MQQ**
Ta zmienna środowiskowa określa katalog, w którym są przechowywane biblioteki produktu IBM MQ classes for Java , jak to pokazano w sekcji Położenie bibliotek produktu IBM MQ classes for Java dla każdej platformy. Niektóre skrypty dostarczone z programem IBM MQ classes for Java, takie jak IVTRun, używają tej zmiennej środowiskowej.

Windows W systemie Windowswszystkie zmienne środowiskowe są ustawiane automatycznie podczas instalacji.

Linux AIX W systemie AIX and Linuxdo ustawienia zmiennych środowiskowych można użyć skryptu `setjmsenv` (jeśli używana jest 32-bitowa maszyna JVM) lub `setjmsenv64` (jeśli używana jest 64-bitowa maszyna JVM). Te skrypty znajdują się w katalogu *MQ_INSTALLATION_PATH*/java/bin .

IBM i W systemie IBM izmienna środowiskowa **QIBM_MULTI_THREADED** musi być ustawiona na Y. Aplikacje wielowątkowe można następnie uruchamiać w taki sam sposób, jak aplikacje jednowątkowe. Więcej informacji na ten temat zawiera sekcja Konfigurowanie produktu IBM MQ z produktem Java i usługą JMS.

IBM MQ classes for Java wymagają środowiska Java 7 Java Runtime Environment (JRE). Informacje na temat położenia odpowiedniego środowiska JRE, które jest instalowane z produktem IBM MQ, zawiera sekcja “Katalogi instalacyjne produktu IBM MQ classes for Java” na stronie 366.

*IBM MQ classes for Java biblioteki*
Położenie bibliotek IBM MQ classes for Java zależy od platformy. Tę lokalizację należy określić podczas uruchamiania aplikacji.

Aby określić położenie bibliotek JNI ( Java Native Interface), należy uruchomić aplikację za pomocą komendy **java** w następującym formacie:

```
java -Djava.library.path= library_path application_name
```

gdzie *ścieżka_biblioteki* jest ścieżką do pliku IBM MQ classes for Java, który zawiera biblioteki JNI. Tabela 53 na stronie 369 przedstawia położenie bibliotek IBM MQ classes for Java dla każdej platformy. W tej tabeli *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym zainstalowano produkt IBM MQ .

*Tabela 53. Położenie bibliotek IBM MQ classes for Java dla każdej platformy*

| **Platforma** | **Katalog zawierający biblioteki IBM MQ classes for Java** |
|---|---|
| AIX AIX | *MQ_INSTALLATION_PATH*/java/lib (biblioteki 32-bitowe)<br>*MQ_INSTALLATION_PATH*/java/lib64 (biblioteki 64-bitowe) |
| Linux Linux (platformax86 ) | *MQ_INSTALLATION_PATH*/java/lib |
| Linux Linux (platformy POWER, x86-64 i zSeries s390x ) | *MQ_INSTALLATION_PATH*/java/lib (biblioteki 32-bitowe)<br>*MQ_INSTALLATION_PATH*/java/lib64 (biblioteki 64-bitowe) |
| Windows Windows | *MQ_INSTALLATION_PATH*\Java\lib (biblioteki 32-bitowe)<br>*MQ_INSTALLATION_PATH*\Java\lib64 (biblioteki 64-bitowe) |
| z/OS z/OS | *MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/lib (biblioteki 32-bitowe i 64-bitowe) |

**Uwaga:**

1. Linux AIX W systemach AIX lub Linux (platforma Power) należy użyć bibliotek 32-bitowych lub 64-bitowych. Bibliotek 64-bitowych należy używać tylko wtedy, gdy aplikacja działa w 64-bitowej wirtualnej maszynie języka Java (JVM) na platformie 64-bitowej. W przeciwnym razie należy użyć bibliotek 32-bitowych.
2. Windows W systemie Windowsmożna użyć zmiennej środowiskowej PATH do określenia położenia bibliotek IBM MQ classes for Java zamiast określania ich położenia w komendzie **java** .
3. IBM i Aby używać języka IBM MQ classes for Java w trybie powiązań w systemie IBM i, należy upewnić się, że biblioteka QMQMJAVA znajduje się na liście bibliotek.
4. z/OS W systemie z/OSmożna użyć 32-bitowej lub 64-bitowej wirtualnej maszyny języka Java (JVM). Nie ma potrzeby określania bibliotek, które mają być używane. Program IBM MQ classes for Java może samodzielnie określić, które biblioteki JNI mają zostać załadowane.

**Pojęcia pokrewne**
użycieIBM MQ classes for Java

Po zainstalowaniu produktu IBM MQ classes for Javamożna skonfigurować instalację w taki sposób, aby uruchamiać własne aplikacje.

*Obsługa środowiska OSGi z produktem IBM MQ classes for Java*
Środowisko OSGi udostępnia środowisko, które obsługuje wdrażanie aplikacji jako pakunków. Trzy pakunki OSGi są dostarczane jako część IBM MQ classes for Java.

Środowisko OSGi udostępnia ogólne przeznaczenie, bezpieczne i zarządzane środowisko Java , które obsługuje wdrażanie aplikacji w postaci pakunków. Urządzenia zgodne z OSGi mogą pobierać i instalować pakunki oraz usuwać je, gdy nie są już potrzebne. Środowisko zarządza instalowaniem i aktualizowaniem pakunków w sposób dynamiczny i skalowalny.

Plik IBM MQ classes for Java zawiera następujące pakunki OSGi.

**com.ibm.mq.osgi.java_*version_number*.jar**
Pliki JAR umożliwiające aplikacjom korzystanie z produktu IBM MQ classes for Java.

**com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar**
V 9.3.0 JM 3.0 V 9.3.0 W przypadku produktu Jakarta Messaging 3.0ten plik JAR umożliwia aplikacjom korzystanie zarówno z pliku IBM MQ classes for JMS , jak i z pliku IBM MQ classes for Java, a także zawiera kod do obsługi komunikatów PCF.

**com.ibm.mq.osgi.allclient_*version_number*.jar**
JMS 2.0 W przypadku systemu JMS 2.0ten plik JAR umożliwia aplikacjom używanie zarówno pliku IBM MQ classes for JMS , jak i pliku IBM MQ classes for Java, a także zawiera kod do obsługi komunikatów PCF.

**com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar**
V 9.3.0 JM 3.0 V 9.3.0 W przypadku systemu Jakarta Messaging 3.0ten plik JAR udostępnia wymagania wstępne dla pliku `com.ibm.mq.jakarta.osgi.allclient_version_number.jar`.

**com.ibm.mq.osgi.allclientprereqs_*version_number*.jar**
JMS 2.0 W przypadku systemu JMS 2.0ten plik JAR udostępnia wymagania wstępne dla pliku `com.ibm.mq.osgi.allclient_version_number.jar`.

gdzie *version_number* jest numerem wersji zainstalowanego produktu IBM MQ .

Pakunki są instalowane w podkatalogu `java/lib/OSGi` instalacji IBM MQ lub w folderze `java\lib\OSGi` w systemie Windows.

W produkcie IBM MQ 8.0należy używać pakunków `com.ibm.mq.osgi.allclient_8.0.0.0.jar`i `com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar` dla wszystkich nowych aplikacji. Użycie tych pakunków eliminuje ograniczenie polegające na tym, że nie można uruchomić zarówno produktu IBM MQ classes for JMS , jak i produktu IBM MQ classes for Java w ramach tego samego środowiska OSGi. Wszystkie inne ograniczenia nadal jednak mają zastosowanie. W przypadku wersji produktu IBM MQ wcześniejszych niż IBM MQ 8.0obowiązuje ograniczenie dotyczące używania produktu IBM MQ classes for JMS lub IBM MQ classes for Java .

Dziewięć innych pakunków jest również instalowanych w podkatalogu `java/lib/OSGi` instalacji IBM MQ lub w folderze `java\lib\OSGi` w systemie Windows. Te pakunki są częścią pliku IBM MQ classes for JMSi nie mogą być ładowane do środowiska wykonawczego OSGi z załadowanym pakunkiem IBM MQ classes for Java . Jeśli pakunek OSGi produktu IBM MQ classes for Java jest ładowany do środowiska wykonawczego OSGi, w którym są również załadowane pakunki IBM MQ classes for JMS , podczas uruchamiania aplikacji używających pakunku IBM MQ classes for Java lub pakunków IBM MQ classes for JMS wystąpią błędy, jak pokazano w poniższym przykładzie:

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

Pakunek OSGi dla IBM MQ classes for Java został zapisany w specyfikacji OSGi Release 4 i nie działa w środowisku OSGi Release 3.

Należy poprawnie ustawić ścieżkę systemową lub ścieżkę do biblioteki, aby środowisko wykonawcze OSGi mogło znaleźć wszystkie wymagane pliki DLL lub biblioteki współużytkowane.

Jeśli dla IBM MQ classes for Javaużywany jest pakunek OSGi, klasy wyjścia kanału napisane w języku Java nie są obsługiwane z powodu problemu związanego z ładowaniem klas w środowisku z wieloma programami ładującymi klasy, takim jak OSGi. Pakunek użytkownika może mieć informacje o pakunku IBM MQ classes for Java , ale pakunek IBM MQ classes for Java nie ma informacji o żadnym pakunku użytkownika. W rezultacie program ładujący klasy używany w pakunku IBM MQ classes for Java nie może załadować klasy wyjścia kanału, która znajduje się w pakunku użytkownika.

Więcej informacji na temat środowiska OSGi zawiera serwis WWW OSGi alliance .

z/OS *Instalacja produktu IBM MQ classes for Java w systemie z/OS*

W systemie z/OSbiblioteka STEPLIB używana w środowisku wykonawczym musi zawierać biblioteki SCSQAUTH i SCSQANLE systemu IBM MQ .

W produkcie z/OS UNIX System Servicesmożna dodać te biblioteki przy użyciu wiersza w pliku `.profile` , jak pokazano w poniższym przykładzie, zastępując łańcuch `thlqual` kwalifikatorem zestawu danych wysokiego poziomu wybranym podczas instalowania produktu IBM MQ:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

W innych środowiskach zwykle konieczna jest edycja kodu JCL uruchamiania w celu uwzględnienia SCSQAUTH w konkatenacji STEPLIB:

```
 STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
         DD DSN=thlqual.SCSQANLE,DISP=SHR
```

*Plik konfiguracyjny IBM MQ classes for Java*

Plik konfiguracyjny IBM MQ classes for Java określa właściwości, które są używane do konfigurowania produktu IBM MQ classes for Java.

Plik konfiguracyjny IBM MQ classes for Java ma format standardowego pliku właściwości Java .

Przykładowy plik konfiguracyjny, `mqjava.config`, znajduje się w podkatalogu `bin` katalogu instalacyjnego IBM MQ classes for Java . Plik ten zawiera wszystkie obsługiwane właściwości i ich wartości domyślne.

**Uwaga:** Przykładowy plik konfiguracyjny jest nadpisywany podczas aktualizowania instalacji produktu IBM MQ do przyszłego pakietu poprawek. Dlatego zaleca się utworzenie kopii przykładowego pliku konfiguracyjnego do użytku z aplikacjami.

Można wybrać nazwę i położenie pliku konfiguracyjnego IBM MQ classes for Java . Podczas uruchamiania aplikacji należy użyć komendy **java** w następującym formacie:

```
java -Dcom.ibm.msg.client.config.location=config_file_url application_name
```

W komendzie *adres_URL_pliku_konfiguracyjnego* to adres URL (URL) określający nazwę i położenie pliku konfiguracyjnego IBM MQ classes for Java . Obsługiwane są adresy URL następujących typów: `http`, `file`, `ftp`i `jar`.

W poniższym przykładzie przedstawiono komendę **java** :

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/mqjava.config MyAppClass
```

Ta komenda identyfikuje plik konfiguracyjny IBM MQ classes for Java jako plik `D:\mydir\mqjava.config` w lokalnym systemie Windows .

Po uruchomieniu aplikacji program IBM MQ classes for Java odczytuje treść pliku konfiguracyjnego i zapisuje określone właściwości w wewnętrznej składnicy właściwości. Jeśli komenda **java** nie zidentyfikuje pliku konfiguracyjnego lub jeśli nie można znaleźć pliku konfiguracyjnego, program IBM

MQ classes for Java użyje wartości domyślnych dla wszystkich właściwości. Jeśli jest to wymagane, można nadpisać dowolną właściwość w pliku konfiguracyjnym, określając ją jako właściwość systemową w komendzie **java** .

Plik konfiguracyjny IBM MQ classes for Java może być używany z dowolnym obsługiwanym transportem między aplikacją a menedżerem kolejek lub brokerem.

## Nadpisywanie właściwości określonych w pliku konfiguracyjnym IBM MQ MQI client

Plik konfiguracyjny IBM MQ MQI client może również określać właściwości, które są używane do konfigurowania produktu IBM MQ classes for Java. Jednak właściwości określone w pliku konfiguracyjnym IBM MQ MQI client mają zastosowanie tylko wtedy, gdy aplikacja nawiązuje połączenie z menedżerem kolejek w trybie klienta.

Jeśli jest to wymagane, można nadpisać dowolny atrybut w pliku konfiguracyjnym IBM MQ MQI client , określając go jako właściwość w pliku konfiguracyjnym IBM MQ classes for Java . Aby nadpisać atrybut w pliku konfiguracyjnym IBM MQ MQI client , należy użyć wpisu w pliku konfiguracyjnym IBM MQ classes for Java w następującym formacie:

```
com.ibm.mq.cfg.stanza.propName=propValue
```

Zmienne we wpisie mają następujące znaczenie:

***sekcja***
Nazwa sekcji w pliku konfiguracyjnym IBM MQ MQI client , która zawiera atrybut.

***propName***
Nazwa atrybutu określona w pliku konfiguracyjnym IBM MQ MQI client .

***propValue***
Wartość właściwości, która nadpisuje wartość atrybutu określoną w pliku konfiguracyjnym IBM MQ MQI client .

Alternatywnie można nadpisać atrybut w pliku konfiguracyjnym IBM MQ MQI client , określając właściwość jako właściwość systemową w komendzie **java** . Użyj poprzedniego formatu, aby określić właściwość jako właściwość systemową.

Tylko następujące atrybuty w pliku konfiguracyjnym IBM MQ MQI client odnoszą się do systemu IBM MQ classes for Java. Jeśli zostaną podane lub nadpisane inne atrybuty, nie będzie to miało żadnego efektu. W szczególności należy zauważyć, że `ChannelDefinitionFile` i `ChannelDefinitionDirectory` w sekcji CHANNELS w pliku konfiguracyjnym klienta nie są używane. Szczegółowe informacje na temat używania tabeli definicji kanału klienta z produktem IBM MQ classes for Javazawiera sekcja “Używanie tabeli definicji kanału klienta z produktem IBM MQ classes for Java” na stronie 388 .

*Tabela 54. W której sekcji pliku konfiguracyjnego klienta znajduje się atrybut*

| **sekcja** | **Atrybut** |
|---|---|
| Sekcja CHANNELS w pliku konfiguracyjnym klienta | Put1DefaultAlwaysSync |
| Sekcja CHANNELS w pliku konfiguracyjnym klienta | PasswordProtection |
| ClientExit-sekcja ścieżki pliku konfiguracyjnego klienta | ExitsDefaultPath |
| ClientExit-sekcja ścieżki pliku konfiguracyjnego klienta | ExitsDefaultPath64 |
| ClientExit-sekcja ścieżki pliku konfiguracyjnego klienta | Ścieżka klasy JavaExits |
| Sekcja JMQI pliku konfiguracyjnego klienta | useMQCSPauthentication |
| SekcjaMessageBuffer pliku konfiguracyjnego klienta | MaximumSize |

| *Tabela 54. W której sekcji pliku konfiguracyjnego klienta znajduje się atrybut (kontynuacja)* | |
|---|---|
| **sekcja** | **Atrybut** |
| SekcjaMessageBuffer pliku konfiguracyjnego klienta | PurgeTime |
| SekcjaMessageBuffer pliku konfiguracyjnego klienta | UpdatePercentage |
| Sekcja TCP pliku konfiguracyjnego klienta | ClntRcvBuffSize |
| Sekcja TCP pliku konfiguracyjnego klienta | ClntSndBuffSize |
| Sekcja TCP pliku konfiguracyjnego klienta | Limit_czasu_połączenia |
| Sekcja TCP pliku konfiguracyjnego klienta | KeepAlive |

Więcej informacji na temat konfiguracji IBM MQ MQI client zawiera plik konfiguracyjny IBM MQ MQI client , `mqclient.ini`.

**Zadania pokrewne**

Śledzenie klas IBM MQ dla aplikacji Java

*Korzystanie ze standardowego śledzenia środowiska Java do konfigurowania śledzenia Java*

Użyj sekcji ustawień śledzenia środowiska standardowego Java , aby skonfigurować narzędzie śledzenia IBM MQ classes for Java .

**com.ibm.msg.client.commonservices.trace.outputName = *traceOutputnazwa***

*traceOutputName* jest nazwą katalogu i pliku, do którego są wysyłane dane wyjściowe śledzenia.

Domyślnie informacje śledzenia są zapisywane w pliku śledzenia w bieżącym katalogu roboczym aplikacji. Nazwa pliku śledzenia zależy od środowiska, w którym działa aplikacja:

- W systemach IBM MQ 9.1.5 i IBM MQ 9.1.0 Fix Pack 5:
  - Jeśli aplikacja załadowała plik IBM MQ classes for Java z przemieszczalnego pliku JAR `com.ibm.mq.allclient.jar`, dane śledzenia są zapisywane w pliku o nazwie `mqjavaclient_%PID%.cl%u.trc`.
  - Jeśli aplikacja załadowała plik IBM MQ classes for Java z pliku JAR `com.ibm.mq.jar`, dane śledzenia są zapisywane w pliku o nazwie `mqjava_%PID%.cl%u.trc`.
- W systemie IBM MQ 9.0.0 Fix Pack 2:
  - Jeśli aplikacja załadowała plik IBM MQ classes for Java z przemieszczalnego pliku JAR `com.ibm.mq.allclient.jar`, dane śledzenia są zapisywane w pliku o nazwie `mqjavaclient_%PID%.trc`.
  - Jeśli aplikacja załadowała plik IBM MQ classes for Java z pliku JAR `com.ibm.mq.jar`, dane śledzenia są zapisywane w pliku o nazwie `mqjava_%PID%.trc`.
- W przypadku produktu IBM MQ classes for Java dla systemu IBM MQ 9.0.0 Fix Pack 1 lub wcześniejszego dane śledzenia są zapisywane w pliku o nazwie `mqjms_%PID%.trc`.

gdzie *%PID%* jest identyfikatorem procesu śledzonej aplikacji, a *%u* jest unikalną liczbą w celu odróżnienia plików wątków uruchamiających śledzenie w różnych programach ładujących klasy Java .

Jeśli identyfikator procesu jest niedostępny, generowana jest liczba losowa z przedrostkiem `f`. Aby dołączyć identyfikator procesu do określonej nazwy pliku, należy użyć łańcucha %PID%.

Jeśli zostanie podany katalog alternatywny, musi on istnieć i użytkownik musi mieć uprawnienia do zapisu w tym katalogu. Jeśli użytkownik nie ma uprawnień do zapisu, dane wyjściowe śledzenia są zapisywane w pliku `System.err`.

**com.ibm.msg.client.commonservices.trace.include = *includeList***

*includeList* to lista śledzonych pakietów i klas lub wartości specjalnych ALL lub NONE.

Nazwy pakietów lub klas należy oddzielać średnikiem `;`. *`includeList`* przyjmuje wartość domyślną `ALL`i śledzi wszystkie pakiety i klasy w programie IBM MQ classes for Java.

**Uwaga:** Można dołączyć pakiet, ale następnie wykluczyć podpakiety tego pakietu. Na przykład, jeśli zostanie dołączony pakiet `a.b` i zostanie wykluczony pakiet `a.b.x`, dane śledzenia będą obejmować wszystkie elementy `a.b.y` i `a.b.z`, ale nie `a.b.x` ani `a.b.x.1`.

**com.ibm.msg.client.commonservices.trace.exclude = *excludeList***
*`excludeList`* to lista pakietów i klas, które nie są śledzone, wartości specjalne `ALL` lub `NONE`.

Nazwy pakietów lub klas należy oddzielać średnikiem `;`. Wartością domyślną parametru *`excludeList`* jest `NONE`, co oznacza, że żadne pakiety i klasy w produkcie IBM MQ classes for JMS nie są śledzone.

**Uwaga:** Można wykluczyć pakiet, ale następnie dołączyć podpakiety tego pakietu. Na przykład, jeśli pakiet `a.b` zostanie wykluczony i zostanie dołączony pakiet `a.b.x`, dane śledzenia będą obejmować wszystkie elementy `a.b.x` i `a.b.x.1`, ale nie `a.b.y` ani `a.b.z`.

Każdy pakiet lub klasa, która jest określona na tym samym poziomie, jako zarówno uwzględniona, jak i wykluczona, jest włączana.

**com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayB***
*`maxArrayBytes`* to maksymalna liczba bajtów, które są śledzone z dowolnej tablicy bajtów.

Jeśli parametr *`maxArrayBytes`* jest ustawiony na dodatnią liczbę całkowitą, ogranicza on liczbę bajtów w tablicy bajtów, które są zapisywane w pliku śledzenia. Obcina ona tablicę bajtów po zapisaniu pliku *`maxArrayBytes`* . Ustawienie parametru *`maxArrayBytes`* zmniejsza wielkość wynikowego pliku śledzenia i zmniejsza wpływ śledzenia na wydajność aplikacji.

Wartość `0` tej właściwości oznacza, że do pliku śledzenia nie jest wysyłana żadna zawartość żadnej tablicy bajtów.

Wartością domyślną jest `-1`, co powoduje usunięcie limitu liczby bajtów w tablicy bajtów, które są wysyłane do pliku śledzenia.

**com.ibm.msg.client.commonservices.trace.limit = *maxTraceB***
*`maxTraceBytes`* to maksymalna liczba bajtów, które są zapisywane w pliku wyjściowym śledzenia.

Produkt *`maxTraceBytes`* współpracuje z produktem *`traceCycles`*. Jeśli liczba bajtów danych śledzenia jest bliska limitu, plik jest zamykany i uruchamiany jest nowy plik danych wyjściowych śledzenia.

Wartość `0` oznacza, że plik wyjściowy śledzenia ma zerową długość. Wartością domyślną jest `-1`, co oznacza, że ilość danych zapisywanych w pliku wyjściowym śledzenia jest nieograniczona.

**com.ibm.msg.client.commonservices.trace.count = *traceCycles***
*`traceCycles`* to liczba plików wyjściowych śledzenia, które mają być cyklicznie przeglądane.

Jeśli bieżący plik danych wyjściowych śledzenia osiągnie limit określony przez parametr *`maxTraceBytes`*, plik zostanie zamknięty. Dalsze dane wyjściowe śledzenia są zapisywane do następnego pliku wyjściowego śledzenia w kolejności. Każdy plik wyjściowy śledzenia jest wyróżniany przyrostkiem liczbowym dołączonym do nazwy pliku. Bieżący lub najnowszy plik wyjściowy śledzenia to `mqjms.trc.0`, a następny plik wyjściowy śledzenia to `mqjms.trc.1`. Starsze pliki śledzenia są zgodne z tym samym wzorcem numeracji aż do limitu.

Wartością domyślną parametru *`traceCycles`* jest 1. Jeśli *`traceCycles`* ma wartość 1, gdy bieżący plik danych wyjściowych śledzenia osiągnie maksymalną wielkość, plik zostanie zamknięty i usunięty. Zostanie uruchomiony nowy plik wyjściowy śledzenia o takiej samej nazwie. Dlatego w danym momencie istnieje tylko jeden plik wyjściowy śledzenia.

**com.ibm.msg.client.commonservices.trace.parameter = *traceParameters***
Parametr *`traceParameters`* określa, czy parametry metody i zwracane wartości są uwzględniane w danych śledzenia.

Wartością domyślną parametru *`traceParameters`* jest TRUE. Jeśli parametr *`traceParameters`* ma wartość FALSE, śledzone są tylko sygnatury metod.

**com.ibm.msg.client.commonservices.trace.startup = *uruchamianie***
Istnieje faza inicjowania IBM MQ classes for Java , podczas której przydzielane są zasoby. Główne narzędzie śledzenia jest inicjowane w fazie przydzielania zasobów.

Jeśli parametr *startup* ma wartość TRUE, używane jest śledzenie przy uruchamianiu. Informacje śledzenia są generowane natychmiast i obejmują konfigurację wszystkich komponentów, w tym samego narzędzia śledzenia. Informacje śledzenia uruchamiania mogą być używane do diagnozowania problemów z konfiguracją. Informacje śledzenia uruchamiania są zawsze zapisywane w pliku `System.err`.

Wartością domyślną parametru *startup* jest FALSE.

Parametr *startup* jest sprawdzany przed zakończeniem inicjowania. Z tego powodu właściwość tę należy określić tylko w wierszu komend jako właściwość systemową Java . Nie należy go podawać w pliku konfiguracyjnym IBM MQ classes for Java .

**com.ibm.msg.client.commonservices.trace.compress = *compressedTrace***
Ustaw parametr *compressedTrace* na wartość TRUE , aby skompresować dane wyjściowe śledzenia.

Wartością domyślną parametru *compressedTrace* jest FALSE.

Jeśli parametr *compressedTrace* ma wartość TRUE, dane wyjściowe śledzenia są kompresowane. Domyślna nazwa pliku wyjściowego śledzenia ma rozszerzenie `.trz`. Jeśli kompresja jest ustawiona na FALSE(wartość domyślna), plik ma rozszerzenie `.trc` , aby wskazać, że jest nieskompresowany. Jeśli jednak nazwa pliku dla danych wyjściowych śledzenia została określona w parametrze *traceOutputName* , zostanie użyta ta nazwa; do pliku nie zostanie zastosowany przyrostek.

Skompresowane dane wyjściowe śledzenia są mniejsze niż nieskompresowane. Ponieważ liczba operacji we/wy jest mniejsza, można ją zapisać szybciej niż w przypadku nieskompresowanych danych śledzenia. Śledzenie skompresowane ma mniejszy wpływ na wydajność programu IBM MQ classes for Java niż śledzenie nieskompresowane.

Jeśli zostaną ustawione opcje *maxTraceBytes* i *traceCycles* , w miejsce wielu plików tekstowych zostanie utworzonych wiele skompresowanych plików śledzenia.

Jeśli plik IBM MQ classes for Java kończy się w sposób niekontrolowany, skompresowany plik śledzenia może nie być poprawny. Z tego powodu kompresji śledzenia można używać tylko wtedy, gdy program IBM MQ classes for Java jest zamykany w sposób kontrolowany. Kompresji śledzenia należy używać tylko wtedy, gdy badane problemy nie powodują nieoczekiwanego zatrzymania maszyny JVM. Kompresji śledzenia nie należy używać podczas diagnozowania problemów, które mogą spowodować zamknięcie systemu `System.Halt()` lub nieprawidłowe, niekontrolowane zakończenia działania maszyny JVM.

**com.ibm.msg.client.commonservices.trace.level = *traceLevel***
*traceLevel* określa poziom filtrowania dla śledzenia. Zdefiniowane poziomy śledzenia są następujące:

- `TRACE_NONE`: `0`
- `TRACE_EXCEPTION`: `1`
- `TRACE_WARNING`: `3`
- `TRACE_INFO`: `6`
- `TRACE_ENTRYEXIT`: `8`
- `TRACE_DATA`: `9`
- `TRACE_ALL`: `Integer.MAX_VALUE`

Każdy poziom śledzenia obejmuje wszystkie niższe poziomy. Na przykład, jeśli poziom śledzenia jest ustawiony na `TRACE_INFO`, do śledzenia zapisywany jest każdy punkt śledzenia o zdefiniowanym poziomie `TRACE_EXCEPTION`, `TRACE_WARNING`lub `TRACE_INFO` . Wszystkie pozostałe punkty śledzenia są wykluczane.

**com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace***

*standaloneTrace* określa, czy usługa śledzenia klienta IBM MQ classes for Java jest używana w środowisku WebSphere Application Server .

Jeśli parametr *standaloneTrace* ma wartość TRUE, do określenia konfiguracji śledzenia używane są właściwości śledzenia klienta IBM MQ classes for Java .

Jeśli parametr *standaloneTrace* ma wartość FALSE, a klient IBM MQ classes for Java jest uruchomiony w kontenerze WebSphere Application Server , używana jest usługa śledzenia WebSphere Application Server . Generowane informacje śledzenia zależą od ustawień śledzenia serwera aplikacji.

Wartością domyślną parametru *standaloneTrace* jest FALSE.

*IBM MQ classes for Java i narzędzia do zarządzania oprogramowaniem*
Narzędzia do zarządzania oprogramowaniem, takie jak Apache Maven, mogą być używane z serwerem IBM MQ classes for Java.

Wiele dużych organizacji programistycznych używa tych narzędzi do centralnego zarządzania repozytoriami bibliotek innych firm.

Plik IBM MQ classes for Java składa się z wielu plików JAR. Podczas tworzenia aplikacji w języku Java za pomocą tego interfejsu API na komputerze, na którym jest opracowywana aplikacja, wymagana jest instalacja serwera IBM MQ , klienta IBM MQ lub klienta IBM MQ o wartości SupportPac .

Aby użyć narzędzia do zarządzania oprogramowaniem i dodać pliki JAR, które składają się na plik IBM MQ classes for Java , do repozytorium zarządzanego centralnie, należy przestrzegać następujących punktów:

- Repozytorium lub kontener musi być dostępny tylko dla programistów w organizacji. Dystrybucja poza organizacją nie jest dozwolona.
- Repozytorium musi zawierać pełny i spójny zestaw plików JAR z pojedynczej wersji produktu IBM MQ lub pakietu poprawek.
- Użytkownik jest odpowiedzialny za zaktualizowanie repozytorium przy użyciu dowolnej konserwacji udostępnionej przez dział wsparcia IBM .

W katalogu IBM MQ 8.0plik JAR `com.ibm.mq.allclient.jar` musi być zainstalowany w repozytorium.

Od wersji IBM MQ 9.0wymagany jest dostawca zabezpieczeń Bouncy Castle i pliki JAR obsługi CMS . Więcej informacji na ten temat zawierają “Przemieszczalne pliki JAR IBM MQ classes for Java” na stronie 364 i Wsparcie dla środowisk JRE innych niżIBM.

## *Konfiguracja poinstalacyjna dla aplikacji IBM MQ classes for Java*

Po zainstalowaniu produktu IBM MQ classes for Javamożna skonfigurować instalację w taki sposób, aby uruchamiać własne aplikacje.

Należy pamiętać, aby sprawdzić plik readme produktu IBM MQ w celu uzyskania najnowszych informacji lub bardziej szczegółowych informacji o używanym środowisku. Najnowsza wersja pliku readme produktu jest dostępna na stronie WWW IBM MQ, WebSphere MQi MQSeries -pliki readme .

Przed próbą uruchomienia aplikacji IBM MQ classes for Java w trybie powiązań należy upewnić się, że skonfigurowano IBM MQ zgodnie z opisem w sekcji Konfigurowanie.

*Konfigurowanie menedżera kolejek w celu akceptowania połączeń klientów z programu IBM MQ classes for Java*
Aby skonfigurować menedżer kolejek do akceptowania żądań połączeń przychodzących od klientów, należy zdefiniować i zezwolić na użycie kanału połączenia z serwerem oraz uruchomić program nasłuchujący.

Szczegółowe informacje można znaleźć w sekcji “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym” na stronie 1101.

*Uruchamianie aplikacji IBM MQ classes for Java w katalogu Java security manager*
Program IBM MQ classes for Java można uruchomić z włączoną opcją Java security manager .
Aby pomyślnie uruchomić aplikacje z włączoną opcją Java security manager , należy skonfigurować środowisko Java Virtual Machine (JVM) przy użyciu odpowiedniego pliku definicji strategii.

Najprostszym sposobem utworzenia odpowiedniego pliku definicji strategii jest zmiana pliku strategii dostarczonego wraz ze środowiskiem Java runtime environment (JRE). W większości systemów plik ten jest przechowywany w katalogu `path lib/security/java.policy`, względem katalogu JRE. Pliki strategii można edytować przy użyciu preferowanego edytora lub programu policytool dostarczanego ze środowiskiem JRE.

Należy nadać uprawnienia do pliku `com.ibm.mq.jmqi.jar` , aby mógł on:

- Tworzenie gniazd (w trybie klienta)
- Załaduj bibliotekę rodzimą (w trybie powiązań)
- Odczytywanie różnych właściwości ze środowiska

Właściwość systemowa **`os.name`** musi być dostępna dla IBM MQ classes for Java podczas uruchamiania w Java security manager.

Jeśli aplikacja Java używa pliku Java security manager, należy dodać następujące uprawnienie do pliku `java.security.policy` używanego przez aplikację. W przeciwnym razie do aplikacji zostaną zgłoszone wyjątki:

```
permission java.lang.RuntimePermission "modifyThread";
```

Uprawnienie RuntimePermission jest wymagane przez klienta w ramach zarządzania przypisaniami i zamykaniem konwersacji multipleksowanych w połączeniach TCP/IP z menedżerami kolejek.

## Przykładowa pozycja pliku strategii

Poniżej przedstawiono przykład wpisu w pliku strategii, który umożliwia pomyślne uruchomienie programu IBM MQ classes for Java w ramach domyślnego menedżera zabezpieczeń. Zastąp łańcuch *MQ_INSTALLATION_PATH* w tym przykładzie położeniem, w którym w systemie jest zainstalowany produkt IBM MQ classes for Java .

```
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";
};
```

Ten przykład pliku strategii umożliwia poprawne działanie serwera IBM MQ classes for Java w ramach menedżera zabezpieczeń, ale przed uruchomieniem aplikacji może być konieczne włączenie własnego kodu.

Przykładowy kod dostarczany z produktem IBM MQ classes for Java nie został specjalnie włączony do użycia z menedżerem zabezpieczeń, ale testy IVT są uruchamiane z tym plikiem strategii i domyślnym menedżerem zabezpieczeń.

**Ważne:**

Narzędzie śledzenia IBM MQ classes for Java wymaga dalszych uprawnień, ponieważ wykonuje dodatkowe zapytania dotyczące właściwości systemu, a także dalsze operacje systemu plików.

Odpowiedni plik strategii bezpieczeństwa szablonu do uruchomienia w menedżerze zabezpieczeń z włączonym śledzeniem znajduje się w katalogu `samples/wmqjava` instalacji produktu IBM MQ pod nazwą `example.security.policy`.

W przypadku instalacji domyślnej plik `example.security.policy` znajduje się w katalogu:

**Windows**
W kliencie `C:\Program Files\IBM\MQ\Tools\wmqjava\samples\example.security.policy`

**Linux**
W kliencie `/opt/mqm/samp/wmqjava/samples/example.security.policy`

**Solaris**
W kliencie `/opt/mqm/samp/wmqjava/samples/example.security.policy`

**AIX**
W kliencie `/usr/mqm/samp/wmqjava/samples/example.security.policy`

*Uruchamianie aplikacji IBM MQ classes for Java w produkcie CICS Transaction Server*
Aplikacja IBM MQ classes for Java może być uruchamiana jako transakcja w produkcie CICS Transaction Server.

Aby uruchomić aplikację IBM MQ classes for Java jako transakcję w produkcie CICS Transaction Server for z/OS, wykonaj następujące kroki:

1. Zdefiniuj aplikację i transakcję dla CICS , używając dostarczonej transakcji CEDA.
2. Upewnij się, że adapter IBM MQ CICS jest zainstalowany w systemie CICS . z/OS Szczegółowe informacje na ten temat zawiera sekcja Używanie języka IBM MQ z produktem CICS .
3. Upewnij się, że środowisko JVM określone w pliku CICS zawiera odpowiednie wpisy CLASSPATH i LIBPATH.
4. Zainicjuj transakcję, używając dowolnego normalnego procesu.

Więcej informacji na temat uruchamiania transakcji CICS Java zawiera dokumentacja systemu CICS .

## *Weryfikowanie instalacji produktu IBM MQ classes for Java*
Program weryfikujący instalację, MQIVP, jest dostarczany z produktem IBM MQ classes for Java. Za pomocą tego programu można przetestować wszystkie tryby połączenia programu IBM MQ classes for Java.

Program poprosi o podanie wielu opcji i innych danych w celu określenia, który tryb połączenia ma zostać sprawdzony. Aby zweryfikować instalację, wykonaj następującą procedurę:

1. Jeśli program ma być uruchamiany w trybie klienta, należy skonfigurować menedżer kolejek w sposób opisany w sekcji “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym” na stronie 1101. Używana kolejka to SYSTEM.DEFAULT.LOCAL.QUEUE
2. Jeśli program ma być uruchamiany w trybie klienta, należy zapoznać się także z sekcją “użycieIBM MQ classes for Java” na stronie 357.

   Wykonaj pozostałe kroki tej procedury w systemie, w którym zamierzasz uruchomić program.
3. Upewnij się, że zmienna środowiskowa CLASSPATH została zaktualizowana zgodnie z instrukcjami podanymi w sekcji “Zmienne środowiskowe dotyczące systemu IBM MQ classes for Java” na stronie 367.
4. Przejdź do katalogu *MQ_INSTALLATION_PATH*`/mqm/samp/wmqjava/samples`, gdzie *MQ_INSTALLATION_PATH* jest ścieżką instalacji IBM MQ . Następnie w wierszu komend wpisz:

   ```
   java -Djava.library.path= library_path MQIVP
   ```

   gdzie *ścieżka_biblioteki* jest ścieżką do bibliotek IBM MQ classes for Java (patrz sekcja “IBM MQ classes for Java biblioteki” na stronie 369 ).

   W pytaniu oznaczonym (1):

   - Aby użyć połączenia TCP/IP, wprowadź nazwę hosta serwera IBM MQ .
   - Aby użyć połączenia rodzimego (tryb powiązań), pozostaw to pole puste (nie wprowadzaj nazwy).

   Program próbuje:

   - 1. Nawiąż połączenie z menedżerem kolejek
   - 2. Otwórz kolejkę SYSTEM.DEFAULT.LOCAL.QUEUE, umieść komunikat w kolejce, pobierz komunikat z kolejki, a następnie zamknij kolejkę
   - 3. Rozłączanie z menedżerem kolejek
   - 4. Jeśli operacje zakończą się pomyślnie, zwróć komunikat

Poniżej przedstawiono przykład pytań i odpowiedzi, które mogą być wyświetlane. Rzeczywiste pytania i odpowiedzi zależą od sieci IBM MQ .

```
Please enter the IP address of the MQ server    : ipaddress(1)
Please enter the port to connect to             : (1414) (2)
Please enter the server connection channel name : channelname (2)
Please enter the queue manager name             : qmname
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...
```

**Uwaga:**

1. z/OS W systemie z/OSpozostaw puste pole w zachęcie oznaczonej jako (1).
2. Jeśli zostanie wybrane połączenie z serwerem, nie będą wyświetlane zachęty oznaczone (2).
3. IBM i W systemie IBM ikomendę `java MQIVP` można wprowadzić tylko z poziomu powłoki QShell. Aplikację można również uruchomić za pomocą komendy CL `RUNJVA CLASS(MQIVP)`.

## *Korzystanie z przykładowych aplikacji IBM MQ classes for Java*

Przykładowe aplikacje IBM MQ classes for Java udostępniają przegląd wspólnych funkcji interfejsu API języka IBM MQ classes for Java . Można ich używać do weryfikowania konfiguracji serwera instalacji i przesyłania komunikatów oraz do tworzenia własnych aplikacji.

## O tym zadaniu

Jeśli potrzebna jest pomoc przy tworzeniu własnych aplikacji, można użyć przykładowych aplikacji jako punktu początkowego. Dla każdej aplikacji udostępniono zarówno wersję źródłową, jak i skompilowaną. Zapoznaj się z przykładowym kodem źródłowym i zidentyfikuj kluczowe kroki, które należy wykonać, aby utworzyć każdy wymagany obiekt dla aplikacji (MQQueueManager, MQConstants, MQMessage, MQPutMessageOptions i MQDestination), a także aby ustawić konkretne właściwości, które są potrzebne do określenia sposobu działania aplikacji. Więcej informacji na ten temat zawiera “Pisanie aplikacji IBM MQ classes for Java” na stronie 383. Przykłady mogą ulec zmianie w przyszłych wersjach produktu IBM MQ Java.

Tabela 55 na stronie 380 przedstawia miejsce instalacji przykładowych aplikacji IBM MQ classes for Java na każdej platformie:

*Tabela 55. Katalogi instalacyjne przykładowych aplikacji IBM MQ classes for Java*

| Platforma | Katalog |
|---|---|
| AIX AIX<br>Linux Linux | *MQ_INSTALLATION_PATH*`/samp/wmqjava/samples` |
| Windows Windows | *MQ_INSTALLATION_PATH*`\tools\wmqjava\samples` |
| IBM i IBM i | `/qibm/proddata/mqm/java/samples/wmqjava/samples` |
| z/OS z/OS | *MQ_INSTALLATION_PATH*`/java/samples/wmqjava` |

Tabela 56 na stronie 381 przedstawia zestawy przykładowych aplikacji, które są dostarczane z produktem IBM MQ classes for Java.

| *Tabela 56. IBM MQ classes for Java aplikacje przykładowe* | |
|---|---|
| **Nazwa próbki** | **Opis** |
| IMSBridgeSample.java | Prosty program do demonstrowania przy użyciu mostu IMS z IBM MQ classes for Java. |
| MQIVP.java | Program sprawdzający instalację produktu IBM MQ Java . |
| MQMessagePropertiesSample.java | Demonstruje użycie interfejsu API właściwości komunikatu. |
| MQPubSubApiSample.java | Demonstruje użycie interfejsu API publikowania/subskrybowania. |
| MQSample.java | Prosty program demonstrujący umieszczanie i pobieranie komunikatu z kolejki. |
| MQSampleMessageManager.java | Klasa narzędziowa do obsługi komunikatów w IBM MQ podstawowych Java przykładach. |
| mqjcivp.properties | Ten pakunek zasobów zawiera komunikaty używane przez program sprawdzający instalację IBM MQ classes for Java (`MQIVP.java`). |

IBM MQ classes for Java udostępnia skrypt o nazwie `runjms` , który może być używany do uruchamiania przykładowych aplikacji. Ten skrypt konfiguruje środowisko IBM MQ , aby umożliwić uruchamianie przykładowych aplikacji IBM MQ classes for Java .

Tabela 57 na stronie 381 przedstawia położenie skryptu na każdej platformie:

| *Tabela 57. Położenie skryptu `runjms`* | |
|---|---|
| **Platforma** | **Katalog** |
| AIX AIX<br>Linux Linux | *MQ_INSTALLATION_PATH*`/java/bin/runjms` |
| Windows Windows | *MQ_INSTALLATION_PATH*`\java\bin\runjms.bat` |
| IBM i IBM i | `/qibm/proddata/mqm/java/bin/runjms`<br>lub wersji<br>`/qibm/proddata/mqm/java/bin/runjms64` |
| z/OS z/OS | *MQ_INSTALLATION_PATH*`java/bin/runjms` |

Aby użyć skryptu `runjms` do wywołania przykładowej aplikacji, wykonaj następujące kroki:

## Procedura

1. Otwórz wiersz komend i przejdź do katalogu zawierającego przykładową aplikację, która ma zostać uruchomiona.
2. Wprowadź następującą komendę:

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   Przykładowa aplikacja wyświetla listę parametrów, których potrzebuje.
3. Wprowadź następującą komendę, aby uruchomić przykład z następującymi parametrami:

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

**Przykład**

Linux Na przykład, aby uruchomić przykład MQIVP w systemie Linux, wprowadź następujące komendy:

```
cd /opt/mqm/samp/wmqjava/samples
/opt/mqm/java/bin/runjms MQIVP
```

**Pojęcia pokrewne**

“Co jest instalowane w systemie IBM MQ classes for JMS” na stronie 92
Podczas instalowania produktu IBM MQ classes for JMStworzona jest pewna liczba plików i katalogów. W systemie Windowsniektóre czynności konfiguracyjne są wykonywane podczas instalacji przez automatyczne ustawianie zmiennych środowiskowych. Na innych platformach i w niektórych środowiskach Windows należy ustawić zmienne środowiskowe przed uruchomieniem aplikacji IBM MQ classes for JMS .

## *Rozwiązywanie problemów z systemem IBM MQ classes for Java*

Początkowo uruchom program sprawdzający instalację. Może być również konieczne użycie narzędzia śledzenia.

Jeśli aplikacja nie zakończy się pomyślnie, uruchom program sprawdzający instalację i postępuj zgodnie z zaleceniami podanymi w komunikatach diagnostycznych. Program weryfikujący instalację jest opisany w sekcji “Weryfikowanie instalacji produktu IBM MQ classes for Java” na stronie 379.

Jeśli problemy będą się nadal pojawiać i konieczne będzie skontaktowanie się z zespołem serwisu IBM , może zostać wyświetlona prośba o włączenie narzędzia śledzenia. Należy to zrobić w sposób przedstawiony w poniższym przykładzie.

Aby śledzić program `MQIVP` :

- Utwórz plik właściwości `com.ibm.mq.commonservices` (patrz sekcja Korzystanie z interfejsu com.ibm.mq.commonservices).
- Wprowadź następującą komendę:

```
java -Dcom.ibm.mq.commonservices=commonservices_properties_file java
-Djava.library.path= library_path MQIVP -trace
```

gdzie:

- *commonservices_properties_file* to ścieżka (w tym nazwa pliku) do pliku właściwości `com.ibm.mq.commonservices` .
- *ścieżka_biblioteki* to ścieżka do bibliotek IBM MQ classes for Java (patrz sekcja “IBM MQ classes for Java biblioteki” na stronie 369).

Więcej informacji na temat używania funkcji śledzenia zawiera sekcja Śledzenie aplikacji produktu IBM MQ classes for Java.

## z/OS MQ Adv. VUE Połączenia klienta Java z aplikacjami wsadowymi działającymi w systemie z/OS

W pewnych warunkach aplikacja IBM MQ classes for Java w systemie z/OS może nawiązać połączenie z menedżerem kolejek w systemie z/OS przy użyciu połączenia klienta. Użycie połączenia klienckiego może uprościć topologie IBM MQ .

Jeśli przy użyciu połączenia klienckiego aplikacja IBM MQ classes for Java jest uruchomiona w środowisku wsadowym i spełniony jest jeden z następujących warunków, aplikacja może nawiązać połączenie ze zdalnym menedżerem kolejek produktu z/OS :

- V 9.3.4 LTS Kod IBM MQ classes for Java jest dostępny w wersji IBM MQ 9.3.4 lub nowszej albo w wersji Long Term Support z zastosowaną poprawką APAR PH56722 . Menedżer kolejek może być w dowolnej obsługiwanej wersji.

- Menedżer kolejek, z którym nawiązywane jest połączenie, działa z upoważnieniem IBM MQ Advanced for z/OS Value Unit Edition i dlatego ma parametr **ADVCAP** ustawiony na wartość ENABLED. Menedżer kolejek może być w dowolnej obsługiwanej wersji.

Więcej informacji na temat IBM MQ Advanced for z/OS Value Unit Edition zawiera sekcja Identyfikatory produktówIBM MQ i informacje o eksporcie.

See WYŚWIETLENIE QMGR for more information on **ADVCAP** and URUCHOM QMGR for more information on **QMGRPROD**.

Aplikacja IBM MQ classes for Java w systemie z/OS nie może używać połączenia w trybie klienta do nawiązywania połączenia z menedżerem kolejek, który nie jest uruchomiony w systemie z/OS

Jeśli aplikacja IBM MQ classes for Java w systemie z/OS próbuje nawiązać połączenie przy użyciu trybu klienta, ale nie jest to dozwolone, zwracana jest wartość MQRC_ENVIRONMENT_ERROR .

### Obsługa Advanced Message Security (AMS)

Aplikacje klienckie IBM MQ classes for Java mogą używać produktu AMS podczas nawiązywania połączenia ze zdalnymi menedżerami kolejek systemu z/OS , z zastrzeżeniem warunków opisanych wcześniej w tym temacie.

Aby używać produktu AMS w ten sposób, aplikacje klienckie muszą używać magazynu kluczy typu `jceracfks` w pliku `keystore.conf`, gdzie:

- Przedrostek nazwy właściwości to `jceracfks` , a w przedrostku nazwy nie jest rozróżniana wielkość liter.
- Magazyn kluczy jest plikiem kluczy RACF .
- Hasła nie są wymagane i zostaną zignorowane. Dzieje się tak dlatego, że pliki kluczy RACF nie używają haseł.
- Jeśli zostanie określony dostawca, musi to być dostawca `IBMJCE`.

W przypadku użycia opcji `jceracfks` z opcją AMSplik kluczy musi mieć następującą postać: `safkeyring://`*`user`*`/`*`keyring`*, gdzie:

- `safkeyring` jest literałem i w nazwie nie jest rozróżniana wielkość liter
- *`user`* to identyfikator użytkownika RACF , który jest właścicielem pliku kluczy.
- *`keyring`* to nazwa pliku kluczy RACF , a w nazwie pliku kluczy rozróżniana jest wielkość liter

W poniższym przykładzie używany jest standardowy plik kluczy AMS dla użytkownika `JOHNDOE`:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

**Pojęcia pokrewne**

“Połączenia klienta JMS/Jakarta Messaging z aplikacjami wsadowymi działającymi w systemie z/OS” na stronie 130
W pewnych warunkach aplikacja IBM MQ classes for JMS/Jakarta Messaging w systemie z/OS może nawiązać połączenie z menedżerem kolejek w systemie z/OS przy użyciu połączenia klienta. Użycie połączenia klienckiego może uprościć topologie IBM MQ .

## Pisanie aplikacji IBM MQ classes for Java

Ta kolekcja tematów zawiera informacje pomocne podczas pisania aplikacji Java do interakcji z systemami IBM MQ .

Aby uzyskać dostęp do kolejek systemu IBM MQ za pomocą programu IBM MQ classes for Java , należy napisać aplikacje Java zawierające wywołania, które umieszczają komunikaty w kolejkach systemu IBM MQ i pobierają z nich komunikaty. Szczegółowe informacje na temat poszczególnych klas zawiera sekcja IBM MQ classes for Java.

**Uwaga:** Automatyczne ponowne nawiązywanie połączenia przez klient nie jest obsługiwane przez produkt IBM MQ classes for Java.

## Interfejs IBM MQ classes for Java

Proceduralny aplikacyjny interfejs programistyczny IBM MQ korzysta z komend, które działają na obiektach. Interfejs programistyczny Java używa obiektów, na których można wykonywać działania, wywołując metody.

Proceduralny aplikacyjny interfejs programistyczny IBM MQ jest zbudowany na bazie komend, takich jak:

```
MQBACK, MQBEGIN, MQCLOSE, MQCONN, MQDISC,
MQGET, MQINQ, MQOPEN, MQPUT, MQSET, MQSUB
```

Wszystkie te komendy przyjmują jako parametr uchwyt do obiektu IBM MQ , na którym mają działać. Program składa się z zestawu obiektów IBM MQ , na które użytkownik działa, wywołując metody na tych obiektach.

Jeśli używany jest interfejs proceduralny, należy rozłączyć się z menedżerem kolejek za pomocą wywołania MQDISC (Hconn, CompCode, Reason), gdzie *Hconn* jest uchwytem menedżera kolejek.

W interfejsie produktu Java menedżer kolejek jest reprezentowany przez obiekt klasy MQQueueManager. Rozłączenie z menedżerem kolejek następuje przez wywołanie metody disconnect () dla tej klasy.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.disconnect();
```

### *Tryby połączenia IBM MQ classes for Java*

Sposób programowania w systemie IBM MQ classes for Java zależy od trybów połączenia, które mają być używane.

Jeśli używane są połączenia klienckie, istnieje wiele różnic w stosunku do wartości IBM MQ MQI client , ale jest ona koncepcyjnie podobna. Jeśli używany jest tryb powiązań, można użyć powiązań krótkiej ścieżki i wydać komendę MQBEGIN. Tryb, który ma być używany, określa się, ustawiając zmienne w klasie MQEnvironment.

#### *IBM MQ classes for Java połączenia klientów*

Jeśli IBM MQ classes for Java jest używany jako klient, jest podobny do IBM MQ MQI client, ale ma wiele różnic.

W przypadku programowania dla produktu *IBM MQ classes for Java* w celu użycia go jako klienta należy pamiętać o następujących różnicach:

- Obsługuje tylko protokół TCP/IP.
- Podczas uruchamiania nie są odczytywane żadne zmienne środowiskowe IBM MQ .
- Informacje, które będą przechowywane w definicji kanału i w zmiennych środowiskowych, mogą być przechowywane w klasie o nazwie Środowisko. Informacje te można również przekazać jako parametry podczas nawiązywania połączenia.
- Warunki błędów i wyjątków są zapisywane w dzienniku określonym w klasie `MQException` . Domyślnym miejscem docelowym błędów jest konsola Java .
- Tylko następujące atrybuty w pliku konfiguracyjnym klienta IBM MQ dotyczą systemu IBM MQ classes for Java. Jeśli zostaną podane inne atrybuty, będą one nieskuteczne.

| sekcja | Atrybut |
|---|---|
| ClientExit-sekcja ścieżki pliku konfiguracyjnego klienta | ExitsDefaultPath |
| ClientExit-sekcja ścieżki pliku konfiguracyjnego klienta | ExitsDefaultPath64 |
| ClientExit-sekcja ścieżki pliku konfiguracyjnego klienta | JavaExitsClasspath |
| SekcjaMessageBuffer pliku konfiguracyjnego klienta | MaximumSize |
| SekcjaMessageBuffer pliku konfiguracyjnego klienta | PurgeTime |
| SekcjaMessageBuffer pliku konfiguracyjnego klienta | UpdatePercentage |
| Sekcja TCP pliku konfiguracyjnego klienta | ClntRcvBuffSize |
| Sekcja TCP pliku konfiguracyjnego klienta | ClntSndBuffSize |
| Sekcja TCP pliku konfiguracyjnego klienta | Limit_czasu_połączenia |
| Sekcja TCP pliku konfiguracyjnego klienta | KeepAlive |

- Jeśli nawiązywane jest połączenie z menedżerem kolejek, który wymaga konwersji danych znakowych, klient produktu V7 Java jest teraz w stanie przeprowadzić konwersję, jeśli nie jest to możliwe. Maszyna JVM klienta musi obsługiwać konwersję między identyfikatorem CCSID klienta i identyfikatorem CCSID menedżera kolejek.
- Automatyczne ponowne nawiązywanie połączenia przez klient nie jest obsługiwane przez produkt IBM MQ classes for Java.

W przypadku użycia w trybie klienta produkt *IBM MQ classes for Java* nie obsługuje wywołania MQBEGIN.

*IBM MQ classes for Java tryb powiązań*
Tryb powiązań IBM MQ classes for Java różni się od trybu klienta na trzy główne sposoby.

W przypadku użycia w trybie powiązań produkt IBM MQ classes for Java używa rodzimego interfejsu języka Java (JNI) do wywoływania bezpośrednio w istniejącym interfejsie API menedżera kolejek, zamiast komunikowania się w sieci.

Domyślnie aplikacje, które używają IBM MQ classes for Java w trybie powiązań, łączą się z menedżerem kolejek przy użyciu opcji *ConnectOption*, MQCNO_STANDARD_BINDINGS.

IBM MQ classes for Java obsługuje następujące *ConnectOption*s:

- MQCNO_FASTPATH_BINDING,
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_IZOLOWANE_POWIĄZANIE

Więcej informacji na temat opcji *ConnectOption*szawiera sekcja “Nawiązywanie połączenia z menedżerem kolejek przy użyciu wywołania MQCONNX” na stronie 761.

Tryb powiązań obsługuje wywołanie MQBEGIN w celu zainicjowania globalnych jednostek pracy koordynowanych przez menedżer kolejek na wszystkich platformach oprócz platform IBM MQ for IBM i i IBM MQ for z/OS.

Większość parametrów udostępnianych przez klasę `MQEnvironment` nie dotyczy trybu powiązań i są ignorowane.

*Definiowanie używanego połączenia z systemem IBM MQ classes for Java*
Typ połączenia, które ma być używane, jest określany przez ustawienie zmiennych w klasie MQEnvironment.

Używane są dwie zmienne:

**MQEnvironment.properties**
Typ połączenia jest określany przez wartość powiązaną z nazwą klucza `CMQC.TRANSPORT_PROPERTY`. Lista poprawnych wartości:

**CMQC.TRANSPORT_MQSERIES_BINDINGS**
Połącz w trybie powiązań

**CMQC.TRANSPORT_MQSERIES_CLIENT**
Połącz w trybie klienta

**CMQC.TRANSPORT_MQSERIES**
Tryb połączenia jest określany przez wartość właściwości *hostname* .

**MQEnvironment.hostname**
Ustaw wartość tej zmiennej w następujący sposób:

- W przypadku połączeń klienckich ustaw wartość tej zmiennej na nazwę hosta serwera IBM MQ , z którym ma zostać nawiązane połączenie.
- W trybie powiązań nie ustawiaj tej zmiennej lub ustaw ją na wartość NULL

## *Operacje na menedżerach kolejek*
W tej kolekcji tematów opisano sposób nawiązywania połączenia z menedżerem kolejek i rozłączania się z nim przy użyciu programu IBM MQ classes for Java.

*Konfigurowanie środowiska IBM MQ dla systemu IBM MQ classes for Java*
Aby aplikacja mogła nawiązać połączenie z menedżerem kolejek w trybie klienta, musi ona określić nazwę kanału, nazwę hosta i numer portu.

**Uwaga:** Informacje zawarte w tym temacie mają zastosowanie tylko wtedy, gdy aplikacja nawiązuje połączenie z menedżerem kolejek w trybie klienta. Nie ma znaczenia, jeśli połączenie jest nawiązywane w trybie powiązań. Patrz sekcja “Tryby połączenia dla IBM MQ classes for JMS” na stronie 113.

Nazwę kanału, nazwę hosta i numer portu można określić na jeden z dwóch sposobów: jako pola w klasie MQEnvironment lub jako właściwości obiektu MQQueueManager .

Jeśli w klasie MQEnvironment zostaną ustawione pola, mają one zastosowanie do całej aplikacji, z wyjątkiem sytuacji, w której są nadpisywane przez tabelę mieszającą właściwości. Aby określić nazwę kanału i nazwę hosta w środowisku MQEnvironment, należy użyć następującego kodu:

```
MQEnvironment.hostname = "host.domain.com";
MQEnvironment.channel = "java.client.channel";
```

Jest to równoważne z ustawieniem zmiennej środowiskowej **MQSERVER** :

```
"java.client.channel/TCP/host.domain.com".
```

Domyślnie klienty Java próbują połączyć się z programem nasłuchującym IBM MQ na porcie 1414. Aby określić inny port, należy użyć następującego kodu:

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

gdzie nnnn jest wymaganym numerem portu

Jeśli właściwości są przekazywane do obiektu menedżera kolejek podczas jego tworzenia, mają one zastosowanie tylko do tego menedżera kolejek. Utwórz pozycje w obiekcie Hashtable z kluczami **hostname**, **channel**i, opcjonalnie, **port**oraz z odpowiednimi wartościami. Aby użyć portu domyślnego

1414, można pominąć pozycję **port** . Utwórz obiekt MQQueueManager przy użyciu konstruktora, który akceptuje tabelę mieszającą właściwości.

## Identyfikowanie połączenia z menedżerem kolejek przez ustawienie nazwy aplikacji

Aplikacja może ustawić nazwę identyfikującą jej połączenie z menedżerem kolejek. Ta nazwa aplikacji jest wyświetlana w komendzie **DISPLAY CONN MQSC/PCF** (gdzie pole jest nazywane **APPLTAG** ). lub na ekranie **Połączenia aplikacji** w Eksploratorze IBM MQ (gdzie pole ma nazwę **App name** ).

Nazwy aplikacji są ograniczone do 28 znaków, dlatego dłuższe nazwy są obcinane. Jeśli nazwa aplikacji nie zostanie podana, zostanie podana nazwa domyślna. Nazwa domyślna jest oparta na klasie wywołującej (main), ale jeśli ta informacja nie jest dostępna, używany jest tekst `IBM MQ Client for Java` .

Jeśli używana jest nazwa klasy wywołującej, jest ona dopasowywana w celu dopasowania do niej przez usunięcie początkowych nazw pakietów, jeśli jest to konieczne. Jeśli na przykład klasą wywołującą jest `com.example.MainApp`, używana jest pełna nazwa, ale jeśli klasą wywołującą jest `com.example.dictionaryAndThesaurus.multilingual.mainApp`, używana jest nazwa `multilingual.mainApp` , ponieważ jest to najdłuższa kombinacja nazwy klasy i nazwy pakietu po prawej stronie, która pasuje do dostępnej długości.

Jeśli sama nazwa klasy ma więcej niż 28 znaków, zostanie obcięta w celu dopasowania. Na przykład łańcuch `com.example.mainApplicationForSecondTestCase` będzie mieć postać `mainApplicationForSecondTest`.

Aby ustawić nazwę aplikacji w klasie MQEnvironment, dodaj nazwę do tabeli mieszającej MQEnvironment.properties z kluczem **MQConstants.APPNAME_PROPERTY**, używając następującego kodu:

```
MQEnvironment.properties.put(MQConstants.APPNAME_PROPERTY, "my_application_name");
```

Aby ustawić nazwę aplikacji w tabeli mieszającej właściwości, która jest przekazywana do konstruktora MQQueueManager , należy dodać nazwę do tabeli mieszającej właściwości z kluczem **MQConstants.APPNAME_PROPERTY**.

## Nadpisywanie właściwości określonych w pliku konfiguracyjnym klienta IBM MQ

Plik konfiguracyjny klienta IBM MQ może również określać właściwości, które są używane do konfigurowania produktu IBM MQ classes for Java. Jednak właściwości określone w pliku konfiguracyjnym IBM MQ MQI client mają zastosowanie tylko wtedy, gdy aplikacja nawiązuje połączenie z menedżerem kolejek w trybie klienta.

W razie potrzeby można nadpisać dowolny atrybut w pliku konfiguracyjnym IBM MQ w dowolny z następujących sposobów. Opcje są wyświetlane w kolejności wykonywania operacji.

- Ustaw właściwość systemową Java dla właściwości konfiguracyjnej.
- Ustaw właściwość w odwzorowaniu MQEnvironment.properties .
- W systemie Javaw wersji 5 i nowszych należy ustawić systemową zmienną środowiskową.

Tylko następujące atrybuty w pliku konfiguracyjnym klienta IBM MQ dotyczą systemu IBM MQ classes for Java. Jeśli zostaną podane lub nadpisane inne atrybuty, nie będzie to miało żadnego efektu.

| sekcja | Atrybut |
|---|---|
| ClientExit-sekcja ścieżki pliku konfiguracyjnego klienta | ExitsDefaultPath |
| ClientExit-sekcja ścieżki pliku konfiguracyjnego klienta | ExitsDefaultPath64 |

| sekcja | Atrybut |
|---|---|
| ClientExit-sekcja ścieżki pliku konfiguracyjnego klienta | JavaExitsClasspath |
| SekcjaMessageBuffer pliku konfiguracyjnego klienta | MaximumSize |
| SekcjaMessageBuffer pliku konfiguracyjnego klienta | PurgeTime |
| SekcjaMessageBuffer pliku konfiguracyjnego klienta | UpdatePercentage |
| Sekcja TCP pliku konfiguracyjnego klienta | ClntRcvBufSize |
| Sekcja TCP pliku konfiguracyjnego klienta | ClntSndBufSize |
| Sekcja TCP pliku konfiguracyjnego klienta | Limit_czasu_połączenia |
| Sekcja TCP pliku konfiguracyjnego klienta | KeepAlive |

*Nawiązywanie połączenia z menedżerem kolejek w programie IBM MQ classes for Java*
Nawiąż połączenie z menedżerem kolejek, tworząc nową instancję klasy MQQueueManager . Rozłącz się z menedżerem kolejek, wywołując metodę disconnect ().

Teraz można nawiązać połączenie z menedżerem kolejek, tworząc nową instancję klasy MQQueueManager :

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

Aby rozłączyć się z menedżerem kolejek, wywołaj metodę disconnect () w menedżerze kolejek:

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

Jeśli zostanie wywołana metoda rozłączenia, wszystkie otwarte kolejki i procesy, do których uzyskano dostęp za pośrednictwem tego menedżera kolejek, zostaną zamknięte. Dobrą praktyką programowania jest jednak jawne zamykanie tych zasobów po zakończeniu ich używania. W tym celu należy użyć metody close () dla odpowiednich obiektów.

Metody commit () i backout () w menedżerze kolejek są równoważne z wywołaniami MQCMIT i MQBACK używanymi z interfejsem proceduralnym.

*Używanie tabeli definicji kanału klienta z produktem IBM MQ classes for Java*
Aplikacja kliencka IBM MQ classes for Java może używać definicji kanału połączenia klienckiego zapisanych w tabeli definicji kanału klienta (CCDT).

Zamiast tworzyć definicję kanału połączenia klienckiego, ustawiając określone pola i właściwości środowiska w klasie `MQEnvironment` lub przekazując je do `MQQueueManager` w tabeli mieszającej właściwości, aplikacja kliencka systemu IBM MQ classes for Java może używać definicji kanału połączenia klienckiego przechowywanych w tabeli definicji kanału klienta. Definicje te są tworzone przez komendy skryptowe IBM MQ (MQSC) lub komendy języka IBM MQ Programmable Command Format (PCF) albo przy użyciu pliku IBM MQ Explorer .

Gdy aplikacja tworzy obiekt `MQQueueManager` , klient IBM MQ classes for Java przeszukuje tabelę definicji kanału klienta w poszukiwaniu odpowiedniej definicji kanału połączenia klienta i używa definicji kanału do uruchomienia kanału MQI. Więcej informacji na temat tabel definicji kanału klienta oraz sposobu ich tworzenia zawiera sekcja Tabela definicji kanału klienta.

Aby użyć tabeli definicji kanału klienta, aplikacja musi najpierw utworzyć obiekt URL . Obiekt URL hermetyzuje adres URL (URL) identyfikujący nazwę i położenie pliku zawierającego tabelę definicji kanału klienta oraz określający sposób dostępu do pliku.

Jeśli na przykład plik `ccdt1.tab` zawiera tabelę definicji kanału klienta i jest przechowywany w tym samym systemie, w którym działa aplikacja, aplikacja może utworzyć obiekt URL w następujący sposób:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
```

Inny przykład: załóżmy, że plik `ccdt2.tab` zawiera tabelę definicji kanału klienta i jest przechowywany w systemie innym niż ten, w którym działa aplikacja. Jeśli dostęp do pliku można uzyskać za pomocą protokołu FTP, aplikacja może utworzyć obiekt URL w następujący sposób:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
```

Po utworzeniu przez aplikację obiektu URL aplikacja może utworzyć obiekt `MQQueueManager` przy użyciu jednego z konstruktorów, który przyjmuje jako parametr obiekt URL . Oto przykład:

```
MQQueueManager mars = new MQQueueManager("MARS", chanTab2);
```

Ta instrukcja powoduje, że klient IBM MQ classes for Java uzyskuje dostęp do tabeli definicji kanału klienta identyfikowanej przez obiekt URL chanTab2, wyszukuje w tabeli odpowiednią definicję kanału połączenia klienta, a następnie używa definicji kanału do uruchomienia kanału MQI do menedżera kolejek o nazwie MARS.

Należy zwrócić uwagę na następujące punkty, które mają zastosowanie, jeśli aplikacja używa tabeli definicji kanału klienta:

- Gdy aplikacja tworzy obiekt `MQQueueManager` przy użyciu konstruktora, który przyjmuje jako parametr obiekt URL , nie należy ustawiać nazwy kanału w klasie `MQEnvironment` ani jako pola, ani jako właściwości środowiska. Jeśli nazwa kanału jest ustawiona, klient IBM MQ classes for Java zgłasza wyjątek `MQException`. Właściwość pola lub środowiska określająca nazwę kanału jest uważana za ustawioną, jeśli jej wartość jest inna niż NULL, pusty łańcuch lub łańcuch zawierający same znaki odstępu.
- Parametr **queueManagerName** w konstruktorze `MQQueueManager` może mieć jedną z następujących wartości:
  - Nazwa menedżera kolejek
  - Gwiazdka (*), po której następuje nazwa grupy menedżerów kolejek
  - Gwiazdka (*)
  - Null, pusty łańcuch lub łańcuch zawierający wszystkie znaki puste

  Są to te same wartości, które mogą być używane dla parametru **QMgrName** w wywołaniu MQCONN wywoływanym przez aplikację kliencką używającą interfejsu MQI (Message Queue Interface). Więcej informacji na temat znaczenia tych wartości zawiera sekcja“Przegląd interfejsu kolejki komunikatów” na stronie 745.

  Jeśli aplikacja używa zestawiania połączeń, należy zapoznać się z sekcją“Sterowanie domyślną pulą połączeń w programie IBM MQ classes for Java” na stronie 409.
- Gdy klient IBM MQ classes for Java znajdzie odpowiednią definicję kanału połączenia klienta w tabeli definicji kanału klienta, używa tylko informacji wyodrębnionych z tej definicji kanału do uruchomienia kanału MQI. Wszystkie pola powiązane z kanałem lub właściwości środowiska, które aplikacja mogła ustawić w klasie `MQEnvironment` , są ignorowane.

  W przypadku korzystania z protokołu TLS (Transport Layer Security) należy zwrócić uwagę na następujące kwestie:
  - Kanał MQI używa protokołu TLS tylko wtedy, gdy definicja kanału wyodrębniona z tabeli definicji kanału klienta określa nazwę CipherSpec obsługiwaną przez klient IBM MQ classes for Java .
  - Tabela definicji kanału klienta zawiera również informacje o położeniu serwerów LDAP (Lightweight Directory Access Protocol), które przechowują listy odwołań certyfikatów (CRL). Klient IBM MQ

classes for Java używa tych informacji tylko w celu uzyskania dostępu do serwerów LDAP, które przechowują listy CRL.

– Tabela definicji kanału klienta może również zawierać położenie programu odpowiadającego OCSP. Produkt IBM MQ classes for Java nie może używać informacji OCSP z pliku tabeli definicji kanału klienta. Można jednak skonfigurować protokół OCSP w sposób opisany w sekcji Korzystanie z protokołu Online Certificate Protocol .

Więcej informacji na temat używania protokołu TLS z tabelą definicji kanału klienta zawiera sekcja Określanie, że kanał MQI używa protokołu TLS.

Należy również zwrócić uwagę na następujące punkty, jeśli używane są wyjścia kanału:

– Kanał MQI używa wyjść kanału i powiązanych danych użytkownika określonych przez definicję kanału wyodrębnioną z tabeli definicji kanału klienta zamiast wyjść kanału i danych określonych za pomocą innych metod.

– Definicja kanału wyodrębniona z tabeli definicji kanału klienta może określać wyjścia kanału zapisane w języku Java, C lub C + +. Więcej informacji na temat zapisywania wyjścia kanału w języku Java zawiera sekcja“Tworzenie wyjścia kanału w programie IBM MQ classes for Java” na stronie 403. Więcej informacji na temat pisania wyjścia kanału w innych językach zawiera sekcja “Korzystanie z wyjść kanałów, które nie zostały napisane w języku Java przy użyciu języka IBM MQ classes for Java” na stronie 406.

*Określanie zakresu portów dla połączeń klienta IBM MQ classes for Java*
Można określić port lub zakres portów, z którymi aplikacja może być powiązana na jeden z dwóch sposobów.

Gdy aplikacja IBM MQ classes for Java próbuje połączyć się z menedżerem kolejek IBM MQ w trybie klienta, firewall może zezwalać tylko na połączenia pochodzące z określonych portów lub zakresu portów. W takiej sytuacji można określić port lub zakres portów, z którymi aplikacja może być powiązana. Porty można określić w następujący sposób:

- W klasie MQEnvironment można ustawić pole ustawień localAddress. Oto przykład:

```
MQEnvironment.localAddressSetting = "192.0.2.0(2000,3000)";
```

- Można ustawić właściwość środowiska CMQC.LOCAL_ADDRESS_PROPERTY. Oto przykład:

```
(MQEnvironment.properties).put(CMQC.LOCAL_ADDRESS_PROPERTY,
                               "192.0.2.0(2000,3000)");
```

- Podczas tworzenia obiektu MQQueueManager można przekazać tabelę mieszającą właściwości zawierającą właściwość LOCAL_ADDRESS_PROPERTY o wartości "192.0.2.0(2000,3000)"

W każdym z tych przykładów, gdy aplikacja łączy się później z menedżerem kolejek, aplikacja łączy się z lokalnym adresem IP i numerem portu z zakresu od 192.0.2.0(2000) do 192.0.2.0(3000).

W systemie z więcej niż jednym interfejsem sieciowym można również użyć pola ustawień localAddresslub właściwości środowiska CMQC.LOCAL_ADDRESS_PROPERTY, aby określić, który interfejs sieciowy ma być używany dla połączenia.

Jeśli zakres portów zostanie ograniczony, mogą wystąpić błędy połączenia. W przypadku wystąpienia błędu zgłaszany jest wyjątek MQException zawierający kod przyczyny IBM MQ MQRC_Q_MGR_NOT_AVAILABLE i następujący komunikat:

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

Błąd może wystąpić, jeśli wszystkie porty z podanego zakresu są używane lub jeśli podany adres IP, nazwa hosta lub numer portu nie są poprawne (na przykład ujemny numer portu).

### *Uzyskiwanie dostępu do kolejek, tematów i procesów w produkcie IBM MQ classes for Java*

Aby uzyskać dostęp do kolejek, tematów i procesów, należy użyć metod klasy MQQueueManager . Struktura deskryptora obiektu (object descriptor structure-MQOD) jest zwijana w parametry tych metod.

## Kolejki

Aby otworzyć kolejkę, można użyć metody accessQueue klasy MQQueueManager . Na przykład w przypadku menedżera kolejek o nazwie queueManagernależy użyć następującego kodu:

```
MQQueue queue = queueManager.accessQueue("qName",CMQC.MQOO_OUTPUT);
```

Metoda accessQueue zwraca nowy obiekt klasy MQQueue.

Po zakończeniu korzystania z kolejki użyj metody close (), aby ją zamknąć, tak jak w poniższym przykładzie:

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

Kolejkę można również utworzyć za pomocą konstruktora MQQueue. Parametry są dokładnie takie same, jak w przypadku metody accessQueue , z dodanym parametrem menedżera kolejek. Na przykład:

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            CMQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserID");
```

Podczas tworzenia kolejek można określić wiele opcji. Szczegółowe informacje na ten temat zawiera sekcja Class.com.ibm.mq.MQQueue. Skonstruowanie obiektu kolejki w ten sposób umożliwia napisanie własnych podklas kolejki MQQueue.

## Tematy

Podobnie można otworzyć temat przy użyciu metody accessTopic klasy MQQueueManager . Na przykład w menedżerze kolejek o nazwie queueManagernależy użyć następującego kodu, aby utworzyć subskrybent i publikator:

```
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);
```

Po zakończeniu korzystania z tematu użyj metody close (), aby go zamknąć.

Temat można również utworzyć przy użyciu konstruktora MQTopic. Parametry są dokładnie takie same, jak w przypadku metody accessTopic , z dodaniem parametru menedżera kolejek. Na przykład:

```
MQTopic subscriber = new
      MQTopic(queueManager,"TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

Podczas tworzenia tematów można określić wiele opcji. Szczegółowe informacje na ten temat zawiera sekcja Klasa com.ibm.mq.MQTopic. Skonstruowanie obiektu tematu w ten sposób umożliwia napisanie własnych podklas obiektu MQTopic.

Temat musi zostać otwarty w celu opublikowania lub subskrypcji. Klasa MQQueueManager ma osiem metod accessTopic , a klasa Topic ma osiem konstruktorów. W każdym przypadku cztery z nich mają parametr **destination** , a cztery mają parametr **subscriptionName** (w tym dwa, które mają oba te parametry). Można ich używać tylko do otwierania tematu na potrzeby subskrypcji. Dwie pozostałe metody mają parametr **openAs** , a temat można otworzyć dla publikacji lub subskrypcji w zależności od wartości parametru **openAs** .

Aby utworzyć temat jako trwały subskrybent, należy użyć metody accessTopic klasy MQQueueManager lub konstruktora MQTopic, który akceptuje nazwę subskrypcji i w obu przypadkach ustawia wartość CMQC.MQSO_DURABLE MQSO_DURABLE.

## Procesy

Aby uzyskać dostęp do procesu, należy użyć metody accessProcess menedżera MQQueueManager. Na przykład w menedżerze kolejek o nazwie queueManagerużyj następującego kodu, aby utworzyć obiekt MQProcess:

```
MQProcess process =
queueManager.accessProcess("PROCESSNAME",
CMQC.MQOO_FAIL_IF_QUIESCING);
```

Aby uzyskać dostęp do procesu, należy użyć metody accessProcess menedżera MQQueueManager.

Metoda accessProcess zwraca nowy obiekt klasy MQProcess.

Po zakończeniu korzystania z obiektu procesu użyj metody close (), aby go zamknąć, tak jak w poniższym przykładzie:

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

Proces można również utworzyć przy użyciu konstruktora MQProcess. Parametry są dokładnie takie same, jak w przypadku metody accessProcess , z dodaniem parametru menedżera kolejek. Na przykład:

```
MQProcess process =
      new MQProcess(queueManager,"PROCESSNAME",
      CMQC.MQOO_FAIL_IF_QUIESCING);
```

Skonstruowanie obiektu procesu w ten sposób umożliwia napisanie własnych podklas procesu MQProcess.

### *Obsługa komunikatów w produkcie IBM MQ classes for Java*

Komunikaty są reprezentowane przez klasę MQMessage. Komunikaty są umieszczane i odbierane za pomocą metod klasy MQDestination, która ma podklasy MQQueue i MQTopic.

Umieszczanie komunikatów w kolejkach lub tematach przy użyciu metody put () klasy MQDestination. Komunikaty są pobierane z kolejek lub tematów za pomocą metody get () klasy MQDestination.
W przeciwieństwie do interfejsu proceduralnego, w którym MQPUT i MQGET umieszczają i pobierają tablice bajtów, język programowania Java umieszcza i pobiera instancje klasy MQMessage. Klasa MQMessage hermetyzuje bufor danych, który zawiera rzeczywiste dane komunikatu, wraz ze wszystkimi parametrami deskryptora komunikatu (MQMD) i właściwościami komunikatu opisującymi ten komunikat.

Aby zbudować nowy komunikat, należy utworzyć nową instancję klasy MQMessage i użyć metod writeXXX do umieszczenia danych w buforze komunikatów.

Podczas tworzenia nowej instancji komunikatu wszystkie parametry MQMD są automatycznie ustawiane na wartości domyślne zgodnie z definicją w sekcji Wartości początkowe i deklaracje języka dla deskryptora MQMD. Metoda put () miejsca docelowego MQDestination przyjmuje również jako parametr

instancję klasy opcji MQPutMessage. Ta klasa reprezentuje strukturę MQPMO. Poniższy przykład tworzy komunikat i umieszcza go w kolejce:

```
// 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);
```

Metoda get () obiektu MQDestination zwraca nową instancję obiektu MQMessage, który reprezentuje komunikat właśnie pobierany z kolejki. Jako parametr przyjmuje również instancję klasy opcji MQGetMessage. Ta klasa reprezentuje strukturę MQGMO.

Nie ma potrzeby określania maksymalnej wielkości komunikatu, ponieważ metoda get () automatycznie dopasowuje wielkość swojego wewnętrznego buforu, aby dopasować ją do komunikatu przychodzącego. Użyj metod readXXX klasy MQMessage, aby uzyskać dostęp do danych w zwróconym komunikacie.

W poniższym przykładzie przedstawiono sposób pobierania komunikatu z kolejki:

```
// 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);
```

Format liczb używany przez metody odczytu i zapisu można zmienić, ustawiając zmienną elementu *encoding* .

Zestaw znaków, który ma być używany do odczytywania i zapisywania łańcuchów, można zmienić, ustawiając zmienną elementu *characterSet* .

Więcej informacji na ten temat zawiera sekcja Klasa MQMessage .

**Uwaga:** Metoda writeUTF() obiektu MQMessage automatycznie koduje długość łańcucha oraz zawarte w nim bajty Unicode. Jeśli komunikat zostanie odczytany przez inny program Java (przy użyciu metody readUTF()), jest to najprostszy sposób wysyłania informacji o łańcuchu.

*Zwiększanie wydajności nietrwałych komunikatów w produkcie IBM MQ classes for Java*
Aby zwiększyć wydajność podczas przeglądania komunikatów lub odbierania nietrwałych komunikatów z aplikacji klienckiej, można użyć opcji *odczytu z wyprzedzeniem*. Aplikacje klienckie korzystające z wywołania MQGET lub wykorzystania asynchronicznego będą korzystać z poprawy wydajności podczas przeglądania komunikatów lub odbierania komunikatów nietrwałych.

Informacje ogólne na temat funkcji odczytu z wyprzedzeniem zawiera temat pokrewny.

W produkcie IBM MQ classes for Javaużywany jest produkt CMQC.MQSO_READ_AHEAD i CMQC.MQSO_NO_READ_AHEAD obiektu MQQueue lub MQTopic w celu określenia, czy konsumenci komunikatów i przeglądarki kolejek mogą używać odczytu z wyprzedzeniem dla tego obiektu.

*Asynchroniczne umieszczanie komunikatów przy użyciu programu IBM MQ classes for Java*
Aby umieścić komunikat asynchronicznie, należy ustawić parametr MQPMO_ASYNC_RESPONSE.

Komunikaty są umieszczane w kolejkach lub tematach za pomocą metody put () klasy MQDestination. Aby umieścić komunikat asynchronicznie, co oznacza, że operacja może zostać zakończona bez

oczekiwania na odpowiedź z menedżera kolejek, można ustawić wartość MQPMO_ASYNC_RESPONSE w polu opcji opcji MQPutMessage. Aby określić powodzenie lub niepowodzenie operacji asynchronicznego umieszczania, należy użyć wywołania statusu MQQueueManager.getAsync.

### *Publikowanie/subskrypcja w produkcie IBM MQ classes for Java*

W produkcie IBM MQ classes for Javatemat jest reprezentowany przez klasę MQTopic i jest publikowany przy użyciu metod MQTopic.put().

Informacje ogólne na temat usługi publikowania/subskrypcji produktu IBM MQ zawiera sekcja Usługa przesyłania komunikatów w trybie publikowania/subskrypcji.

### *Obsługa nagłówków komunikatów produktu IBM MQ w produkcie IBM MQ classes for Java*

Dostępne są klasy języka Java reprezentujące różne typy nagłówków komunikatów. Dostępne są również dwie klasy pomocnicze.

## Interfejs MQHeader

Obiekty nagłówka są opisywane przez interfejs MQHeader, który udostępnia metody ogólnego przeznaczenia umożliwiające dostęp do pól nagłówka oraz odczytywanie i zapisywanie treści komunikatu. Każdy typ nagłówka ma własną klasę, która implementuje interfejs MQHeader i dodaje metody pobierające i ustawiające dla poszczególnych pól. Na przykład typ nagłówka MQRFH2 jest reprezentowany przez klasę MQRFH2 , typ nagłówka MQDLH przez klasę MQDLH itd. Klasy nagłówków przeprowadzają niezbędną konwersję danych automatycznie i mogą odczytywać lub zapisywać dane w dowolnym określonym kodowaniu numerycznym lub zestawie znaków (CCSID).

**Ważne:** Klasy nagłówków MQRFH2 traktują komunikat jako plik o dostępie bezpośrednim, co oznacza, że kursor musi być ustawiony na początku komunikatu. Przed użyciem klasy nagłówka komunikatu wewnętrznego, takiej jak MQRFH, MQRFH2, MQCIH, MQDEAD, MQIIH lub MQXMIT, należy upewnić się, że pozycja kursora komunikatu została zaktualizowana do poprawnego położenia przed przekazaniem komunikatu do klasy.

## Klasy pomocnicze

Dwie klasy pomocnicze, MQHeaderIterator i MQHeaderList, ułatwiają odczytywanie i dekodowanie (analizowanie) treści nagłówka w komunikatach:

- Klasa MQHeaderIterator działa jak klasa java.util.Iterator. Jeśli w komunikacie znajduje się więcej nagłówków, metoda next () zwraca wartość true, a metoda nextHeader() lub next () zwraca następny obiekt nagłówka.
- Klasa MQHeaderList działa jak klasa java.util.List. Podobnie jak iterator MQHeaderIteratoranalizuje treść nagłówka, ale umożliwia również wyszukiwanie konkretnych nagłówków, dodawanie nowych nagłówków, usuwanie istniejących nagłówków, aktualizowanie pól nagłówków, a następnie zapisywanie treści nagłówka z powrotem do komunikatu. Alternatywnie można utworzyć pustą listę MQHeaderList, a następnie zapełnić ją instancjami nagłówka i zapisać ją w komunikacie raz lub wielokrotnie.

Klasy MQHeaderIterator i MQHeaderList korzystają z informacji w rejestrze MQHeaderRegistry , aby dowiedzieć się, które klasy nagłówków IBM MQ są powiązane z określonymi typami i formatami komunikatów. Rejestr MQHeaderRegistry jest skonfigurowany z uwzględnieniem wszystkich bieżących formatów i typów nagłówków IBM MQ oraz ich klas implementacji. Można również rejestrować własne typy nagłówków.

Obsługiwane są następujące często używane nagłówki IBM MQ

- MQRFH-reguły i nagłówek formatowania
- MQRFH2 -podobnie jak MQRFH, używany do przekazywania komunikatów do i z brokera komunikatów należącego do produktu IBM Integration Bus. Używane również do przechowywania właściwości komunikatu

- MQCIH-most CICS
- MQDLH-nagłówek niedostarczonego komunikatu
- MQIIH-nagłówek informacji IMS
- MQRMH-nagłówek komunikatu referencyjnego
- MQSAPH-nagłówek SAP
- MQWIH-nagłówek informacji o pracy
- MQXQH-nagłówek kolejki transmisji
- MQDH-nagłówek dystrybucji
- MQEPH-hermetyzowany nagłówek PCF

Można również zdefiniować klasy reprezentujące własne nagłówki.

Aby użyć elementu MQHeaderIterator w celu uzyskania nagłówka RFH2 , należy ustawić opcję MQGMO_PROPERTIES_FORCE_MQRFH2 w opcjach GetMessagelub ustawić właściwość kolejki PROPCTL na wartość FORCE.

*Drukowanie wszystkich nagłówków w komunikacie przy użyciu programu IBM MQ classes for Java*
W tym przykładzie instancja obiektu MQHeaderIterator analizuje nagłówki komunikatu MQMessage odebranego z kolejki. Obiekty MQHeader zwracane przez metodę nextHeader() wyświetlają swoją strukturę i treść po wywołaniu metody toString .

```
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);
}
```

*Pomijanie nagłówków w komunikacie przy użyciu funkcji IBM MQ classes for Java*
W tym przykładzie metoda skipHeaders() MQHeaderIterator ustawia kursor odczytu komunikatu bezpośrednio za ostatnim nagłówkiem.

```
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 ();
```

*Znajdowanie kodu przyczyny w komunikacie niedostarczonych komunikatów przy użyciu funkcji IBM MQ classes for Java*
W tym przykładzie metoda read zapełnia obiekt MQDLH odczytem z komunikatu. Po wykonaniu operacji odczytu kursor odczytu komunikatu jest umieszczany bezpośrednio za treścią nagłówka MQDLH.

Komunikaty w kolejce niedostarczonych komunikatów menedżera kolejek są poprzedzone nagłówkiem niedostarczonego komunikatu (MQDLH). Aby podjąć decyzję o sposobie obsługi tych komunikatów- na przykład w celu określenia, czy mają one zostać ponowione, czy odrzucone-aplikacja do obsługi niewysłanych wiadomości musi sprawdzić kod przyczyny zawarty w pliku MQDLH.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
```

```
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH ();

dlh.read (message);

System.out.println ("Reason: " + dlh.getReason ());
```

Wszystkie klasy nagłówka udostępniają również konstruktor wygodny do inicjowania się bezpośrednio z komunikatu w pojedynczym kroku. Tak więc kod w tym przykładzie można uprościć w następujący sposób:

```
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 ());
```

*Odczytywanie i usuwanie nagłówka z komunikatu niedostarczonego komunikatu przy użyciu funkcji IBM MQ classes for Java*
W tym przykładzie nagłówek MQDLH jest używany do usuwania nagłówka z komunikatu niedostarczonego komunikatu.

Aplikacja obsługująca niedostarczone wiadomości zwykle ponownie wprowadza odrzucone komunikaty, jeśli ich kod przyczyny wskazuje na błąd przejściowy. Przed ponownym wprowadzeniem komunikatu musi on usunąć nagłówek MQDLH.

W tym przykładzie wykonywane są następujące kroki (patrz komentarze w przykładowym kodzie):

1. Lista MQHeaderList odczytuje cały komunikat, a każdy nagłówek napotkany w komunikacie staje się elementem listy.
2. Komunikaty niedostarczonych komunikatów zawierają komunikat MQDLH jako pierwszy nagłówek, więc można go znaleźć w pierwszym elemencie listy nagłówków. Plik MQDLH został już zapełniony na podstawie komunikatu podczas budowania elementu MQHeaderList , więc nie ma potrzeby wywoływania jego metody odczytu.
3. Kod przyczyny jest wyodrębniany przy użyciu metody getReason() udostępnianej przez klasę MQDLH.
4. Kod przyczyny został sprawdzony i wskazuje, że należy ponownie wysłać komunikat. Komenda MQDLH jest usuwana przy użyciu metody remove () klasy MQHeaderList .
5. Lista MQHeaderList zapisuje pozostałą treść w nowym obiekcie komunikatu. Nowy komunikat zawiera teraz wszystkie elementy oryginalnego komunikatu z wyjątkiem komunikatu MQDLH i może zostać zapisany w kolejce. Argument **true** dla konstruktora i metody write wskazuje, że treść komunikatu ma być przechowywana w obrębie listy MQHeaderList, a następnie zapisywana ponownie.
6. Pole formatu w deskryptorze nowego komunikatu zawiera teraz wartość, która wcześniej znajdowała się w polu formatu MQDLH. Dane komunikatu są zgodne z kodowaniem liczbowym i CCSID ustawionym w deskryptorze komunikatu.

```
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.
```

*Drukowanie treści komunikatu przy użyciu programu IBM MQ classes for Java*
W tym przykładzie użyto komendy MQHeaderList do wydrukowania treści komunikatu wraz z nagłówkami.

Dane wyjściowe zawierają widok całej treści nagłówka oraz treść komunikatu. Klasa MQHeaderList dekoduje wszystkie nagłówki w jednym kroku, podczas gdy klasa MQHeaderIterator przechodzi przez nie pojedynczo pod kontrolą aplikacji. Tej techniki można użyć do udostępnienia prostego narzędzia do debugowania podczas pisania aplikacji WebSphere MQ .

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from a queue.

System.out.println (new MQHeaderList (message, true));
```

W tym przykładzie są również drukowane pola deskryptora komunikatu przy użyciu klasy MQMD. Metoda copyFrom() klasy com.ibm.mq.headers.MQMD zapełnia obiekt nagłówka na podstawie pól deskryptora komunikatu komunikatu MQMessage, a nie przez odczytanie treści komunikatu.

```
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));
```

*Znajdowanie konkretnego typu nagłówka w komunikacie przy użyciu funkcji IBM MQ classes for Java*
W tym przykładzie użyto metody indexOf(String) klasy MQHeaderList w celu znalezienia nagłówka MQRFH2 w komunikacie, jeśli taki istnieje.

```
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);
        ...
}
```

*Analizowanie nagłówka MQRFH2 przy użyciu IBM MQ classes for Java*
W tym przykładzie przedstawiono sposób uzyskania dostępu do wartości znanego pola w nazwanym folderze przy użyciu klasy MQRFH2 .

Klasa MQRFH2 udostępnia wiele sposobów dostępu nie tylko do pól w stałej części struktury, ale także do treści folderu zakodowanej w formacie XML, które są przenoszone w polu danych NameValue. W tym przykładzie przedstawiono sposób uzyskania dostępu do znanej wartości pola w nazwanym folderze-w tej instancji jest to pole Rto w folderze jms, które reprezentuje nazwę kolejki odpowiedzi w komunikacie produktu MQ JMS .

```
MQRFH2 rfh = ...

String value = rfh.getStringFieldValue ("jms", "Rto");
```

Aby wykryć treść MQRFH2 (w przeciwieństwie do bezpośredniego żądania konkretnych pól), można użyć metody getFolders w celu zwrócenia listy MQRFH2.Elementreprezentujący strukturę folderu, który może zawierać pola i inne foldery. Ustawienie pola lub folderu na wartość NULL powoduje usunięcie

go z MQRFH2. Podczas manipulowania zawartością folderu danych NameValuew ten sposób pole StrucLength jest automatycznie aktualizowane.

*Odczytywanie i zapisywanie strumieni bajtów innych niż obiekty MQMessage przy użyciu produktu IBM MQ classes for Java*
W tych przykładach klasy nagłówków są używane do analizowania treści nagłówka IBM MQ i manipulowania nią, gdy źródło danych nie jest obiektem MQMessage.

Klas nagłówków można używać do analizowania treści nagłówka IBM MQ i manipulowania nią, nawet jeśli źródło danych jest czymś innym niż obiekt MQMessage. Interfejs MQHeader implementowany przez każdą klasę nagłówka udostępnia metody `int read (java.io.DataInput message, int encoding, int characterSet)` i `int write (java.io.DataOutput message, int encoding, int characterSet)`. Klasa com.ibm.mq.MQMessage implementuje interfejsy java.io.DataInput i java.io.DataOutput . Oznacza to, że można użyć dwóch metod MQHeader do odczytu i zapisu treści komunikatu MQMessage, przesłaniając kodowanie i identyfikator CCSID określone w deskryptorze komunikatu. Jest to przydatne w przypadku komunikatów, które zawierają łańcuch nagłówków w różnych kodowaniach.

Obiekty DataInput i DataOutput można również uzyskać z innych strumieni danych, na przykład strumieni plików lub gniazd, lub tablic bajtów przenoszonych w komunikatach systemu JMS . Klasy java.io.DataInputStream implementują interfejs DataInput , a klasy java.io.DataOutputStream implementują interfejs DataOutput. W tym przykładzie treść nagłówka IBM MQ jest odczytywany z tablicy bajtów:

```
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);
```

Wiersz rozpoczynający się od MQHeaderIterator można zastąpić wierszem

```
MQDLH dlh = new MQDLH (in, CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
// or any other header type
```

W tym przykładzie do tablicy bajtów jest zapisywany strumień DataOutput:

```
MQHeader header = ... // Could be any header type
ByteArrayOutputStream out = new ByteArrayOutputStream ();

header.write (new DataOutputStream (out), CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
byte [] bytes = out.toByteArray ();
```

Podczas pracy ze strumieniami w ten sposób należy zachować ostrożność, aby użyć poprawnych wartości kodowania i argumentów characterSet . Podczas odczytywania nagłówków należy określić kodowanie i identyfikator CCSID, z którym pierwotnie zapisano treść bajtową. Podczas zapisywania nagłówków należy określić kodowanie i identyfikator CCSID, które mają być produkowane. Konwersja danych jest wykonywana automatycznie przez klasy nagłówka.

*Tworzenie klas dla nowych typów nagłówków przy użyciu języka IBM MQ classes for Java*
Istnieje możliwość utworzenia klas Java dla typów nagłówków, które nie są dostarczane z produktem IBM MQ classes for Java.

Aby dodać klasę Java reprezentującą nowy typ nagłówka, który może być używany w taki sam sposób, jak dowolna klasa nagłówka dostarczana z produktem IBM MQ classes for Java, należy utworzyć klasę implementującą interfejs MQHeader. Najprostszym podejściem jest rozszerzenie klasy com.ibm.mq.headers.impl.Header . W tym przykładzie jest generowana w pełni funkcjonalna klasa reprezentująca strukturę nagłówka MQTM. Nie ma potrzeby dodawania pojedynczych metod pobierających i ustawiających dla każdego pola, ale jest to przydatne dla użytkowników klasy nagłówka. Ogólne metody getValue i setValue , które pobierają łańcuch dla nazwy pola, będą działać dla wszystkich

pól zdefiniowanych w typie nagłówka. Dziedziczone metody odczytu, zapisu i wielkości umożliwią odczytywanie i zapisywanie instancji nowego typu nagłówka i będą poprawnie obliczać wielkość nagłówka na podstawie jego definicji pola. Definicja typu jest tworzona tylko raz, jednak tworzonych jest wiele instancji tej klasy nagłówka. Aby nowa definicja nagłówka była dostępna do dekodowania przy użyciu klas MQHeaderIterator lub MQHeaderList , należy ją zarejestrować przy użyciu klasy MQHeaderRegistry. Należy jednak zauważyć, że klasa nagłówka MQTM jest już udostępniona w tym pakiecie i zarejestrowana w rejestrze domyślnym.

```
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.
}
```

### *Obsługa komunikatów PCF w produkcie IBM MQ classes for Java*

Klasy Java są udostępniane w celu tworzenia i analizowania komunikatów o strukturze PCF oraz w celu ułatwienia wysyłania żądań PCF i gromadzenia odpowiedzi PCF.

Klasy PCFMessage i MQCFGR reprezentują tablice struktur parametrów PCF. Zapewniają one wygodną metodę dodawania i pobierania parametrów PCF.

Struktury parametrów PCF są reprezentowane przez klasy MQCFH, MQCFIN, MQCFIN64, MQCFST, MQCFBS, MQCFIL, MQCFIL64 MQCFSL i MQCFGR. Współużytkują one podstawowe interfejsy operacyjne:

- Metody odczytu i zapisu treści wiadomości: read (), write () i size ()
- Metody manipulowania parametrami: getValue (), setValue (), getParameter () i inne
- Metoda enumeratora.nextParameter (), która analizuje treść PCF w komunikacie MQMessage

Parametr filtru PCF jest używany w komendach inquire w celu udostępnienia funkcji filtru. Jest on hermetyzowany w następujących klasach:

- MQCFIF-filtr całkowitoliczbowy
- MQCFSF-filtr łańcucha
- MQCFBF-filtr bajtów

Do zarządzania połączeniem z menedżerem kolejek, kolejką serwera komend i powiązaną kolejką odpowiedzi udostępniono dwie klasy agentów: PCFAgent i PCFMessageAgent . PCFMessageAgent -rozszerza klasę PCFAgent i powinna być używana zamiast niej. Klasa PCFMessageAgent przekształca

odebrane komunikaty MQMessage i przekazuje je z powrotem do programu wywołującego jako tablicę PCFMessage. PCFAgent zwraca tablicę komunikatów MQMessage, które należy przeanalizować przed użyciem.

### *Obsługa właściwości komunikatu w produkcie IBM MQ classes for Java*

Wywołania funkcji w celu przetworzenia uchwytów komunikatów nie mają odpowiednika w IBM MQ classes for Java. Aby ustawić, zwrócić lub usunąć właściwości uchwytu komunikatu, należy użyć metod klasy MQMessage.

Ogólne informacje na temat właściwości komunikatu zawiera sekcja “Nazwy właściwości” na stronie 28.

W programie IBM MQ classes for Java dostęp do komunikatów odbywa się za pośrednictwem klasy MQMessage. Uchwyty komunikatów nie są zatem udostępniane w środowisku Java i nie ma odpowiednika wywołań funkcji IBM MQ MQCRTMH, MQDLTMH, MQMHBUF i MQBUFMH

Aby ustawić właściwości uchwytu komunikatu w interfejsie proceduralnym, należy użyć wywołania MQSETMP. W produkcie IBM MQ classes for Javaużyj odpowiedniej metody klasy MQMessage:

- Właściwość setBoolean
- Właściwość setByte
- Właściwość setBytes
- Właściwość setShort
- Właściwość setInt
- setInt2Property
- setInt4Property
- setInt8Property
- Właściwość setLong
- Właściwość setFloat
- Właściwość setDouble
- Właściwość setString
- Właściwość setObject

Są one czasem określane zbiorczo jako metody *set*property* .

Aby zwrócić wartość właściwości uchwytu komunikatu w interfejsie proceduralnym, należy użyć wywołania MQINQMP. W produkcie IBM MQ classes for Javaużyj odpowiedniej metody klasy MQMessage:

- Właściwość getBoolean
- Właściwość getByte
- Właściwość getBytes
- Właściwość getShort
- Właściwość getInt
- getInt2Property
- getInt4Property
- getInt8Property
- Właściwość getLong
- Właściwość getFloat
- Właściwość getDouble
- Właściwość getString
- Właściwość getObject

Są one czasem określane zbiorczo jako metody *get*property* .

Aby usunąć wartość właściwości uchwytu komunikatu w interfejsie proceduralnym, należy użyć wywołania MQDLTMP. W produkcie IBM MQ classes for Javanależy użyć metody deleteProperty klasy MQMessage.

## *Obsługa błędów w produkcie IBM MQ classes for Java*

Obsługa błędów wynikających z IBM MQ classes for Java użycia bloków Java `try` i `catch` .

Metody w interfejsie Java nie zwracają kodu zakończenia ani kodu przyczyny. Zamiast tego zgłaszają wyjątek za każdym razem, gdy kod zakończenia i kod przyczyny wynikające z wywołania IBM MQ nie są zerowe. Upraszcza to logikę programu, dzięki czemu nie trzeba sprawdzać kodów powrotu po każdym wywołaniu funkcji IBM MQ. Możesz zdecydować, w którym punkcie programu chcesz poradzić sobie z możliwością wystąpienia awarii. W tych punktach można otoczyć kod blokami `try` i `catch` , tak jak w poniższym przykładzie:

```
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() );
}
```

Kody przyczyn wywołań IBM MQ zgłaszane z powrotem w Java wyjątkach dla z/OS są udokumentowane w sekcji Kody zakończenia i kody przyczyn interfejsu API.

Wyjątki, które są zgłaszane podczas działania aplikacji IBM MQ classes for Java , są również zapisywane w dzienniku. Jednak aplikacja może wywołać metodę MQException.logExclude(), aby zapobiec rejestrowaniu wyjątków powiązanych z konkretnym kodem przyczyny. Można to zrobić w sytuacjach, w których spodziewane jest zgłoszenie wielu wyjątków powiązanych z konkretnym kodem przyczyny, a użytkownik nie chce, aby dziennik był wypełniany tymi wyjątkami. Jeśli na przykład aplikacja próbuje pobrać komunikat z kolejki za każdym razem, gdy iteruje wokół pętli i w większości tych prób oczekuje się, że w kolejce nie będzie odpowiedniego komunikatu, można zapobiec rejestrowaniu wyjątków powiązanych z kodem przyczyny MQRC_NO_MSG_AVAILABLE. Jeśli aplikacja wcześniej uniemożliwiła rejestrowanie wyjątków powiązanych z konkretnym kodem przyczyny, może zezwolić na ponowne rejestrowanie tych wyjątków, wywołując metodę MQException.logInclude().

Czasami kod przyczyny nie przekazuje wszystkich szczegółów powiązanych z błędem. Dla każdego zgłoszonego wyjątku aplikacja powinna sprawdzić powiązany wyjątek. Sam powiązany wyjątek może mieć inny powiązany wyjątek, dlatego powiązane wyjątki tworzą łańcuch prowadzący z powrotem do pierwotnego problemu bazowego. Powiązany wyjątek jest implementowany przy użyciu mechanizmu dołączanego wyjątku klasy java.lang.Throwable , a aplikacja uzyskuje powiązany wyjątek, wywołując metodę Throwable.getCause(). Z wyjątku, który jest instancją wyjątku MQException, metoda MQException.getCause() pobiera bazową instancję klasy com.ibm.mq.jmqi.JmqiException, a metoda getCause z tego wyjątku pobiera bazowy wyjątek java.lang.Exception , który spowodował błąd.

## *Pobieranie i ustawianie wartości atrybutów w programie IBM MQ classes for Java*

Metody getXXX() i setXXX() są udostępniane dla wielu wspólnych atrybutów. Dostęp do innych można uzyskać za pomocą ogólnych metod inquire () i set ().

Dla wielu wspólnych atrybutów klasy MQManagedObject, MQDestination, MQQueue, MQTopic, MQProcess i MQQueueManager zawierają metody getXXX() i setXXX(). Te metody umożliwiają pobieranie i ustawianie ich wartości atrybutów. Należy zauważyć, że w przypadku MQDestination, MQQueue i MQTopic metody działają tylko wtedy, gdy podczas otwierania obiektu zostaną podane odpowiednie opcje zapytania i ustawienia.

W przypadku mniej powszechnych atrybutów wszystkie klasy MQQueueManager, MQDestination, MQQueue, MQTopic i MQProcess dziedziczą z klasy o nazwie MQManagedObject. Ta klasa definiuje interfejsy inquire () i set ().

Podczas tworzenia nowego obiektu menedżera kolejek przy użyciu operatora *nowy* jest on automatycznie otwierany do odpytywania. Jeśli do uzyskania dostępu do obiektu procesu używana jest metoda accessProcess(), obiekt ten jest automatycznie otwierany w celu wykonania zapytania. Jeśli do uzyskania dostępu do obiektu kolejki używana jest metoda accessQueue(), obiekt ten nie jest automatycznie otwierany dla operacji zapytania lub ustawiania. Dzieje się tak, ponieważ automatyczne dodanie tych opcji może spowodować problemy z niektórymi typami kolejek zdalnych. Aby użyć metod inquire, set, getXXXi setXXX w kolejce, należy podać odpowiednie opcje inquire i set w parametrze openOptions metody accessQueue(). To samo dotyczy obiektów docelowych i obiektów tematu.

Metody inquire i set przyjmują trzy parametry:

- tablica selektorów
- Tablica intAttrs
- Tablica charAttrs

Parametry długości SelectorCount, IntAttri CharAttrznalezione w MQINQ nie są potrzebne, ponieważ długość tablicy w Java jest zawsze znana. Poniższy przykład przedstawia sposób tworzenia zapytania w kolejce:

```
// 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));
```

### *Programy wielowątkowe w Java*

Środowisko wykonawcze Java jest z natury wielowątkowe. Program IBM MQ classes for Java umożliwia współużytkowanie obiektu menedżera kolejek przez wiele wątków, ale zapewnia synchronizację całego dostępu do docelowego menedżera kolejek.

Programy wielowątkowe są trudne do uniknięcia w systemie Java. Rozważmy prosty program łączący się z menedżerem kolejek i otwierający kolejkę podczas uruchamiania. Program wyświetli pojedynczy przycisk na ekranie. Gdy użytkownik kliknie ten przycisk, program pobiera komunikat z kolejki.

Środowisko wykonawcze Java jest z natury wielowątkowe. Dlatego inicjowanie aplikacji odbywa się w jednym wątku, a kod wykonywany w odpowiedzi na naciśnięcie przycisku jest wykonywany w osobnym wątku (wątku interfejsu użytkownika).

W przypadku systemu IBM MQ MQI clientopartego na języku C może to spowodować problem, ponieważ istnieją ograniczenia współużytkowania uchwytów przez wiele wątków. Program IBM MQ classes for Java rozluźnia to ograniczenie, umożliwiając współużytkowanie obiektu menedżera kolejek (i powiązanych z nim obiektów kolejki, tematu i procesu) przez wiele wątków.

Implementacja IBM MQ classes for Java zapewnia, że dla konkretnego połączenia (instancja obiektuMQQueueManager ) wszystkie dostępy do docelowego menedżera kolejek produktu IBM MQ są zsynchronizowane. Wątek, który chce wywołać menedżera kolejek, jest blokowany do momentu zakończenia wszystkich innych wywołań w toku dla tego połączenia. Jeśli wymagany jest jednoczesny dostęp do tego samego menedżera kolejek z wielu wątków w programie, należy utworzyć nowy obiekt

MQQueueManager dla każdego wątku, który wymaga dostępu współbieżnego. (Jest to równoważne z wywołaniem osobnego wywołania MQCONN dla każdego wątku).

**Uwaga:** Instancje klasy `com.ibm.mq.MQGetMessageOptions` nie mogą być współużytkowane przez wątki, które współbieżnie żądają komunikatów. Instancje tej klasy są aktualizowane przy użyciu danych podczas wykonywania odpowiedniego żądania MQGET, co może spowodować nieoczekiwane konsekwencje, jeśli wiele wątków działa współbieżnie na tej samej instancji obiektu.

## *Korzystanie z wyjść kanałów w programie IBM MQ classes for Java*

Przegląd sposobu używania wyjść kanałów w aplikacji za pomocą programu IBM MQ classes for Java.

W poniższych tematach opisano sposób zapisywania wyjścia kanału w produkcie Java, przypisywania go i przekazywania do niego danych. Następnie opisano sposób użycia wyjść kanału napisanych w języku C i sekwencji wyjść kanału.

Aby załadować klasę wyjścia kanału, aplikacja musi mieć odpowiednie uprawnienia zabezpieczeń.

### *Tworzenie wyjścia kanału w programie IBM MQ classes for Java*

Użytkownik może udostępnić własne wyjścia kanału, definiując klasę Java , która implementuje odpowiedni interfejs.

Aby zaimplementować wyjście, należy zdefiniować nową klasę Java , która implementuje odpowiedni interfejs. W pakiecie com.ibm.mq.exits zdefiniowano trzy interfejsy wyjścia:

- WMQSendExit
- WMQReceiveExit
- WMQSecurityExit

**Uwaga:** Wyjścia kanału są obsługiwane tylko dla połączeń klienckich; nie są one obsługiwane dla połączeń powiązań. Nie można użyć Java wyjścia kanału poza IBM MQ classes for Java, na przykład, jeśli używana jest aplikacja kliencka napisana w języku C.

Każde szyfrowanie TLS zdefiniowane dla połączenia jest wykonywane *po* wywołaniu wyjść wysyłania i zabezpieczeń. Podobnie deszyfrowanie jest wykonywane *przed* wywołaniem wyjścia odbierania i zabezpieczeń.

W poniższym przykładzie zdefiniowano klasę, która implementuje wszystkie trzy interfejsy:

```
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
  }
}
```

Do każdego wyjścia przekazywany jest obiekt MQCXP i obiekt MQCD. Te obiekty reprezentują struktury MQCXP i MQCD zdefiniowane w interfejsie proceduralnym.

Każda klasa wyjścia, którą piszesz, musi mieć konstruktor. Może to być konstruktor domyślny lub konstruktor, który przyjmuje argument łańcuchowy. Jeśli pobiera ona łańcuch, dane użytkownika zostaną przekazane do klasy wyjścia podczas jej tworzenia. Jeśli klasa wyjścia zawiera zarówno konstruktor domyślny, jak i konstruktor jednoargumentowy, priorytet ma konstruktor jednoargumentowy.

W przypadku wyjść wysyłania i zabezpieczeń kod wyjścia musi zwracać dane, które mają zostać wysłane do serwera. W przypadku wyjścia odbierania kod wyjścia musi zwracać zmodyfikowane dane, które mają być interpretowane przez program IBM MQ .

Najprostszą możliwą treścią wyjścia jest:

```
{ return agentBuffer; }
```

Nie należy zamykać menedżera kolejek z poziomu wyjścia kanału.

## Używanie istniejących klas wyjścia kanału

W wersjach produktu IBM MQ wcześniejszych niż 7.0należy zaimplementować te wyjścia przy użyciu interfejsów MQSendExit, MQReceiveExiti MQSecurityExit, tak jak w poniższym przykładzie. Ta metoda pozostaje poprawna, ale nowa metoda jest preferowana w celu zwiększenia funkcjonalności i wydajności.

```
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
  }
}
```

*Przypisywanie wyjścia kanału w programie IBM MQ classes for Java*
Do przypisania wyjścia kanału można użyć funkcji IBM MQ classes for Java.

Nie ma bezpośredniego odpowiednika kanału IBM MQ w IBM MQ classes for Java. Wyjścia kanału są przypisane do MQQueueManager. Na przykład po zdefiniowaniu klasy, która implementuje interfejs WMQSecurityExit , aplikacja może użyć wyjścia zabezpieczeń na jeden z czterech sposobów:

- Przez przypisanie instancji klasy do pola MQEnvironment.channelSecurityExit przed utworzeniem obiektu MQQueueManager .
- Przez ustawienie pola MQEnvironment.channelSecurityExit na łańcuch reprezentujący klasę wyjścia zabezpieczeń przed utworzeniem obiektu MQQueueManager .
- Przez utworzenie pary klucz/wartość w tabeli mieszającej właściwości przekazanej do MQQueueManager z kluczem CMQC.SECURITY_EXIT_PROPERTY .
- Korzystanie z tabeli definicji kanału klienta (CCDT)

Każde wyjście przypisane przez ustawienie pola MQEnvironment.channelSecurityExit na łańcuch, utworzenie pary klucz/wartość w tabeli mieszającej właściwości lub użycie tabeli mieszającej CCDT musi być zapisane przy użyciu konstruktora domyślnego. Wyjście przypisane jako instancja klasy nie wymaga konstruktora domyślnego, w zależności od aplikacji.

Aplikacja może używać wyjścia wysyłania lub odbierania w podobny sposób. Na przykład w poniższym fragmencie kodu przedstawiono sposób użycia procedur zewnętrznych zabezpieczeń, wysyłania i odbierania, które zostały zaimplementowane w klasie MyMQExits, która została wcześniej zdefiniowana przy użyciu środowiska MQEnvironment:

```
MyMQExits myexits = new MyMQExits();
MQEnvironment.channelSecurityExit = myexits;
MQEnvironment.channelSendExit = myexits;
MQEnvironment.channelReceiveExit = myexits;
:
MQQueueManager jupiter = new MQQueueManager("JUPITER");
```

Jeśli do przypisania wyjścia kanału używana jest więcej niż jedna metoda, kolejność wykonywania operacji jest następująca:

1. Jeśli URL tabeli CCDT zostanie przekazany do menedżera MQQueueManager, treść tabeli CCDT określa wyjścia kanału, które mają być używane, a wszystkie definicje wyjścia w środowisku MQEnvironment lub tabeli mieszającej właściwości są ignorowane.
2. Jeśli nie zostanie przekazany żaden URL tabeli CCDT, zostaną scalone definicje wyjścia ze środowiska MQEnvironment i tabeli mieszającej.
   - Jeśli ten sam typ wyjścia jest zdefiniowany zarówno w środowisku MQEnvironment, jak i w tabeli mieszającej, używana jest definicja w tabeli mieszającej.
   - Jeśli określono równoważne stare i nowe typy wyjścia (na przykład pole sendExit , które może być używane tylko dla typu wyjścia używanego w wersjach wcześniejszych niż IBM WebSphere MQ 7.0oraz pole wyjścia channelSend, które może być używane dla dowolnego wyjścia wysyłania), używane jest nowe wyjście (channelSendExit), a nie stare wyjście.

Jeśli program obsługi wyjścia kanału został zadeklarowany jako łańcuch, należy włączyć opcję IBM MQ , aby znaleźć program obsługi wyjścia kanału. Można to zrobić na różne sposoby, w zależności od środowiska, w którym działa aplikacja, oraz od sposobu pakowania programów obsługi wyjścia kanału.

- W przypadku aplikacji działającej na serwerze aplikacji należy zapisać pliki w katalogu pokazanym w pliku Tabela 58 na stronie 406 lub spakowanym w plikach JAR, do których odwołuje się plik **exitClasspath**.
- W przypadku aplikacji, która nie jest uruchomiona na serwerze aplikacji, obowiązują następujące reguły:
  - Jeśli klasy wyjścia kanału są spakowane w oddzielnych plikach JAR, te pliki JAR muszą zostać dołączone do pliku **exitClasspath**.
  - Jeśli klasy wyjścia kanału nie są spakowane w plikach JAR, pliki klas mogą być przechowywane w katalogu pokazanym w pliku Tabela 58 na stronie 406 lub w dowolnym katalogu w ścieżce klasy systemu JVM lub w katalogu **exitClasspath**.

Właściwość **exitClasspath** można określić na cztery sposoby. W kolejności priorytetów są to następujące sposoby:

1. Właściwość systemowa com.ibm.mq.exitClasspath (zdefiniowana w wierszu komend przy użyciu opcji `-D` )
2. Sekcja exitPath pliku `mqclient.ini`
3. Pozycja tabeli mieszającej z kluczem CMQC.EXIT_CLASSPATH_PROPERTY
4. Zmienna MQEnvironment **exitClasspath**

Wiele ścieżek należy rozdzielić znakiem java.io.File.pathSeparator .

| *Tabela 58. Katalog programów obsługi wyjścia kanału* | |
|---|---|
| **Platforma** | **Katalog** |
| AIX AIX<br>Linux Linux | `/var/mqm/exits` (32-bitowe programy obsługi wyjścia kanału)<br>`/var/mqm/exits64` (64-bitowe programy obsługi wyjścia kanału) |
| Windows | *katalog_danych_instalacji*\exits |

**Uwaga:** *katalog_danych_instalacji* to katalog, który został wybrany dla plików danych IBM MQ podczas instalacji. Katalog domyślny to `C:\ProgramData\IBM\MQ`.

*Przekazywanie danych do wyjść kanału w programie IBM MQ classes for Java*
Dane można przekazywać do wyjść kanału i zwracać dane z wyjść kanału do aplikacji.

## Parametr agentBuffer

Dla wyjścia wysyłania parametr *agentBuffer* zawiera dane, które mają zostać wysłane. Dla wyjścia odbierania lub wyjścia zabezpieczeń parametr *agentBuffer* zawiera odebrane dane. Parametr długości nie jest potrzebny, ponieważ wyrażenie agentBuffer.limit () wskazuje długość tablicy.

W przypadku wyjść wysyłania i zabezpieczeń kod wyjścia musi zwracać dane, które mają zostać wysłane do serwera. W przypadku wyjścia odbierania kod wyjścia musi zwracać zmodyfikowane dane, które mają być interpretowane przez program IBM MQ .

Najprostszą możliwą treścią wyjścia jest:

```
{ return agentBuffer; }
```

Wyjścia kanału są wywoływane z buforem, który ma macierz bazową. Aby uzyskać najlepszą wydajność, wyjście powinno zwrócić bufor z tablicą bazową.

## Dane użytkownika

Jeśli aplikacja łączy się z menedżerem kolejek, ustawiając wyjście channelSecurity, wyjście channelSendlub wyjście channelReceive, 32 bajty danych użytkownika mogą zostać przekazane do odpowiedniej klasy wyjścia kanału w momencie wywołania, za pomocą pól danych channelSecurityExitUser, channelSendExitUserlub channelReceiveExitUser. Te dane użytkownika są dostępne dla klasy wyjścia kanału, ale są odświeżane przy każdym wywołaniu wyjścia. W związku z tym wszystkie zmiany wprowadzone w danych użytkownika w wyjściu kanału zostaną utracone. Aby wprowadzić trwałe zmiany w danych w wyjściu kanału, należy użyć obszaru exitUsersystemu MQCXP. Dane w tym polu są zachowywane między wywołaniami wyjścia.

Jeśli aplikacja ustawia securityExit, sendExitlub receiveExit, do tych klas wyjścia kanału nie mogą być przekazywane żadne dane użytkownika.

Jeśli aplikacja używa tabeli definicji kanału klienta (CCDT) do nawiązania połączenia z menedżerem kolejek, wszystkie dane użytkownika określone w definicji kanału połączenia klienta są przekazywane do klas wyjścia kanału podczas ich wywołania. Więcej informacji na temat używania tabeli definicji kanału klienta zawiera sekcja “Używanie tabeli definicji kanału klienta z produktem IBM MQ classes for Java” na stronie 388.

*Korzystanie z wyjść kanałów, które nie zostały napisane w języku Java przy użyciu języka IBM MQ classes for Java*
Sposób korzystania z programów obsługi wyjścia kanału napisanych w języku C z aplikacji systemu Java .

W programie IBM MQmożna określić nazwę programu obsługi wyjścia kanału napisaną w języku C jako łańcuch przekazywany do pól wyjścia channelSecurity, wyjścia channelSendlub wyjścia channelReceivew

obiekcie MQEnvironment lub tabeli mieszającej właściwości. Nie można jednak użyć wyjścia kanału napisanego w języku Java w aplikacji napisanej w innym języku.

Podaj nazwę programu obsługi wyjścia w formacie `library(function)` i upewnij się, że położenie programu obsługi wyjścia jest określone zgodnie z opisem w sekcji Ścieżka do wyjścia.

Informacje na temat zapisywania wyjścia kanału w języku C zawiera sekcja “Kanał-programy obsługi wyjścia dla kanałów przesyłania komunikatów” na stronie 990.

*Używanie sekwencji wyjść wysyłania lub odbierania kanału w produkcie IBM MQ classes for Java*
Aplikacja IBM MQ classes for Java może używać sekwencji wyjść wysyłania lub odbierania kanału, które są uruchamiane po sobie.

Aby użyć sekwencji wyjść wysyłania, aplikacja może utworzyć listę lub łańcuch zawierający wyjścia wysyłania. Jeśli używana jest lista, każdy element listy może mieć jedną z następujących wartości:

- Instancja klasy zdefiniowanej przez użytkownika, która implementuje interfejs WMQSendExit .
- Instancja klasy zdefiniowanej przez użytkownika, która implementuje interfejs MQSendExit (dla wyjścia wysyłania napisanego w języku Java )
- Instancja klasy wyjścia MQExternalSend(dla wyjścia wysyłania, które nie jest zapisane w Java )
- Instancja klasy łańcucha MQSendExit.
- Instancja klasy String

Lista nie może zawierać innej listy.

Aplikacja może używać sekwencji wyjść odbierania w podobny sposób.

Jeśli używany jest łańcuch, musi on składać się z jednej lub większej liczby rozdzielanych przecinkami definicji wyjścia, z których każda może być nazwą klasy Java lub programu w języku C w formacie `library(function)`.

Następnie aplikacja przypisuje obiekt List lub String do pola MQEnvironment.channelSendExit przed utworzeniem obiektu MQQueueManager .

Kontekst informacji przekazywanych do wyjść znajduje się wyłącznie w domenie wyjść. Na przykład, jeśli program zewnętrzny Java i program zewnętrzny C są połączone w łańcuch, obecność programu zewnętrznego Java nie ma wpływu na program zewnętrzny C.

## Używanie klas łańcucha wyjścia

W wersjach wcześniejszych niż IBM WebSphere MQ 7.0udostępniono dwie klasy umożliwiające sekwencje wyjść:

- MQSendExitŁańcuch, który implementuje interfejs MQSendExit
- MQReceiveExitŁańcuch implementujący interfejs MQReceiveExit

Użycie tych klas pozostaje poprawne, ale preferowana jest nowa metoda. Użycie klas IBM MQ dla interfejsów Java oznacza, że aplikacja nadal jest zależna od `com.ibm.mq.jar` Jeśli używany jest nowy zestaw interfejsów w pakiecie `com.ibm.mq.exits` , nie ma zależności od `com.ibm.mq.jar`.

Aby użyć sekwencji wyjść wysyłania, aplikacja utworzyła listę obiektów, w której każdy obiekt był jednym z następujących obiektów:

- Instancja klasy zdefiniowanej przez użytkownika, która implementuje interfejs MQSendExit (dla wyjścia wysyłania napisanego w języku Java )
- Instancja klasy wyjścia MQExternalSend(dla wyjścia wysyłania, które nie jest zapisane w Java )
- Instancja klasy łańcucha MQSendExit.

Aplikacja utworzyła obiekt łańcucha MQSendExit, przekazując tę listę obiektów jako parametr w konstruktorze. Aplikacja przypisałaby następnie obiekt łańcucha MQSendExitdo pola MQEnvironment.sendExit przed utworzeniem obiektu MQQueueManager .

### *Kompresja kanału w programie IBM MQ classes for Java*

Kompresja danych przepływających przez kanał może zwiększyć wydajność kanału i zmniejszyć ruch w sieci. IBM MQ classes for Java należy użyć funkcji kompresji wbudowanej w produkt IBM MQ.

Za pomocą funkcji dostarczanych wraz z produktem IBM MQmożna kompresować dane przepływające przez kanały komunikatów i kanały MQI, a w każdym typie kanału można kompresować dane nagłówków i dane komunikatów niezależnie od siebie. Domyślnie w kanale nie są kompresowane żadne dane. Pełny opis kompresji kanału, w tym opis sposobu jej implementacji w programie IBM MQ, zawiera sekcja Kompresja danych (COMPMSG) i sekcja Kompresja nagłówka (COMPHDR).

Aplikacja IBM MQ classes for Java określa techniki, które mogą być używane do kompresowania danych nagłówka lub komunikatu w połączeniu klienta przez utworzenie obiektu java.util.Collection . Każda technika kompresji jest obiektem typu Integer w kolekcji, a kolejność, w jakiej aplikacja dodaje techniki kompresji do kolekcji, określa kolejność, w jakiej techniki kompresji są negocjowane z menedżerem kolejek podczas uruchamiania połączenia klienckiego. Aplikacja może następnie przypisać kolekcję do pola listy hdrCompdla danych nagłówka lub do pola listy msgCompdla danych komunikatu w klasie MQEnvironment. Gdy aplikacja jest gotowa, może uruchomić połączenie klienta, tworząc obiekt MQQueueManager .

Poniższe fragmenty kodu ilustrują opisane podejście. Pierwszy fragment kodu przedstawia sposób implementowania kompresji danych nagłówka:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(CMQXC.MQCOMPRESS_SYSTEM));
:
MQEnvironment.hdrCompList = headerComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

Drugi fragment kodu przedstawia sposób implementowania kompresji danych komunikatu:

```
Collection msgComp = new Vector();
msgComp.add(new Integer(CMQXC.MQCOMPRESS_RLE));
msgComp.add(new Integer(CMQXC.MQCOMPRESS_ZLIBHIGH));

:
MQEnvironment.msgCompList = msgComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

W drugim przykładzie techniki kompresji są negocjowane w kolejności RLE, a następnie ZLIBHIGH, gdy połączenie klienta jest uruchamiane. Wybrana technika kompresji nie może być zmieniona w czasie życia obiektu MQQueueManager .

Techniki kompresji danych nagłówka i komunikatu, które są obsługiwane przez klienta i menedżera kolejek w połączeniu klienta, są przekazywane do wyjścia kanału jako kolekcje w polach listy hdrCompi listy msgCompobiektu MQChannelDefinition . Rzeczywiste techniki obecnie używane do kompresowania danych nagłówka i komunikatu w połączeniu klienckim są przekazywane do wyjścia kanału w polach kompresji CurHdri kompresji CurMsgobiektu MQChannelExit .

Jeśli w połączeniu klienta używana jest kompresja, dane są kompresowane przed przetwarzaniem wyjść wysyłania kanału i wyodrębniane po przetworzeniu wyjść odbierania kanału. Dane przekazywane do wyjść wysyłania i odbierania są zatem w stanie skompresowanym.

Więcej informacji na temat określania technik kompresji i dostępnych technik kompresji zawiera sekcja Klasa com.ibm.mq.MQEnvironment i sekcja Interfejs com.ibm.mq.MQC.

### *Współużytkowanie połączenia TCP/IP w programie IBM MQ classes for Java*

Aby współużytkować pojedyncze połączenie TCP/IP, można utworzyć wiele instancji kanału MQI.

W produkcie IBM MQ classes for Javazmienna MQEnvironment.sharingConversations służy do sterowania liczbą konwersacji, które mogą współużytkować jedno połączenie TCP/IP.

Atrybut SHARECNV jest najlepszym podejściem do współużytkowania połączeń. Dlatego w przypadku użycia wartości SHARECNV większej niż 0 z serwerem IBM MQ classes for Java nie ma gwarancji, że nowe żądanie połączenia będzie zawsze współużytkować już nawiązane połączenie.

## *Zestawianie połączeń w produkcie IBM MQ classes for Java*

IBM MQ classes for Java umożliwia zestawianie połączeń zapasowych w celu ich ponownego wykorzystania.

Produkt IBM MQ classes for Java udostępnia dodatkową obsługę aplikacji obsługujących wiele połączeń z menedżerami kolejek systemu IBM MQ . Jeśli połączenie nie jest już potrzebne, zamiast je niszczyć, można je umieścić w puli, a następnie ponownie wykorzystać. Może to znacznie zwiększyć wydajność aplikacji i oprogramowania pośredniego, które łączą się szeregowo z dowolnymi menedżerami kolejek.

IBM MQ udostępnia domyślną pulę połączeń. Aplikacje mogą aktywować lub dezaktywować tę pulę połączeń, rejestrując i wyrejestrowując znaczniki za pomocą klasy MQEnvironment. Jeśli pula jest aktywna, gdy IBM MQ classes for Java tworzy obiekt MQQueueManager , przeszukuje tę domyślną pulę i ponownie wykorzystuje wszystkie odpowiednie połączenia. Gdy wystąpi wywołanie metody MQQueueManager.disconnect (), połączenie bazowe jest zwracane do puli.

Alternatywnie aplikacje mogą utworzyć pulę połączeń menedżera MQSimpleConnectiondla konkretnego użycia. Następnie aplikacja może określić tę pulę podczas konstruowania obiektu MQQueueManager lub przekazać tę pulę do środowiska MQEnvironment w celu użycia jako domyślną pulę połączeń.

Aby zapobiec użyciu zbyt dużej ilości zasobów przez połączenia, można ograniczyć łączną liczbę połączeń, które może obsłużyć obiekt menedżera MQSimpleConnection, a także ograniczyć wielkość puli połączeń. Ustawienie limitów jest przydatne w przypadku wystąpienia konfliktu żądań połączeń w obrębie maszyny JVM.

Domyślnie metoda getMaxConnections () zwraca wartość zero, co oznacza, że nie ma ograniczenia liczby połączeń, które może obsłużyć obiekt menedżera MQSimpleConnection. Limit można ustawić za pomocą metody setMaxConnections (). Jeśli limit zostanie ustawiony, a limit zostanie osiągnięty, żądanie dalszego połączenia może spowodować zgłoszenie wyjątku MQException z kodem przyczyny MQRC_MAX_CONNS_LIMIT_REACHED.

### *Sterowanie domyślną pulą połączeń w programie IBM MQ classes for Java*

W tym przykładzie przedstawiono sposób użycia domyślnej puli połączeń.

Rozważmy następującą przykładową aplikację MQApp1:

```
import com.ibm.mq.*;
public class MQApp1
{
      public static void main(String[] args) throws MQException
      {
         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
      }
}
```

MQApp1 pobiera listę lokalnych menedżerów kolejek z wiersza komend, łączy się z każdym z nich po kolei i wykonuje pewne operacje. Jeśli jednak w wierszu komend ten sam menedżer kolejek jest wyświetlany wiele razy, bardziej wydajne jest nawiązanie połączenia tylko raz i wielokrotne ponowne wykorzystanie tego połączenia.

IBM MQ classes for Java udostępnia domyślną pulę połączeń, której można użyć w tym celu. Aby włączyć pulę, należy użyć jednej z metod MQEnvironment.addConnectionPoolToken(). Aby wyłączyć pulę, należy użyć metody MQEnvironment.removeConnectionPoolToken().

Następująca przykładowa aplikacja, MQApp2, jest funkcjonalnie identyczna z aplikacją MQApp1, ale łączy się tylko raz z każdym menedżerem kolejek.

```
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);

      }
}
```

Pierwszy pogrubiony wiersz aktywuje domyślną pulę połączeń, rejestrując obiekt MQPoolToken w środowisku MQEnvironment.

Konstruktor MQQueueManager wyszukuje w tej puli odpowiednie połączenie i tworzy połączenie z menedżerem kolejek tylko wtedy, gdy nie może znaleźć istniejącego połączenia. Wywołanie metody qmgr.disconnect() zwraca połączenie do puli w celu późniejszego ponownego wykorzystania. Te wywołania API są takie same jak w przypadku przykładowej aplikacji MQApp1.

Druga podświetlona linia dezaktywuje domyślną pulę połączeń, która niszczy wszystkie połączenia menedżera kolejek przechowywane w puli. Jest to ważne, ponieważ w przeciwnym razie aplikacja zostanie zakończona z pewną liczbą aktywnych połączeń menedżera kolejek w puli. Ta sytuacja może spowodować wystąpienie błędów w dziennikach menedżera kolejek.

Jeśli aplikacja używa tabeli definicji kanału klienta (CCDT) do nawiązywania połączenia z menedżerem kolejek, konstruktor MQQueueManager najpierw wyszukuje w tabeli odpowiednią definicję kanału połączenia klienta. W przypadku znalezienia takiego połączenia konstruktor wyszukuje w domyślnej puli połączeń połączenie, które może być używane dla kanału. Jeśli konstruktor nie może znaleźć odpowiedniego połączenia w puli, przeszukuje tabelę definicji kanału klienta w poszukiwaniu następnej odpowiedniej definicji kanału połączenia klienta i kontynuuje działanie zgodnie z wcześniejszym opisem. Jeśli konstruktor zakończy wyszukiwanie w tabeli definicji kanału klienta i nie znajdzie żadnego odpowiedniego połączenia w puli, rozpocznie drugie wyszukiwanie w tabeli. Podczas tego wyszukiwania konstruktor próbuje utworzyć nowe połączenie dla każdej odpowiedniej definicji kanału połączenia klienckiego i używa pierwszego połączenia, które zarządza.

Domyślna pula połączeń przechowuje maksymalnie dziesięć nieużywanych połączeń i utrzymuje aktywność nieużywanych połączeń przez maksymalnie pięć minut. Aplikacja może to zmienić (szczegółowe informacje na ten temat zawiera sekcja “Podawanie innej puli połączeń w programie IBM MQ classes for Java” na stronie 411 ).

Zamiast używać środowiska MQEnvironment do dostarczania elementu MQPoolToken, aplikacja może utworzyć własny element:

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

Niektóre aplikacje lub dostawcy oprogramowania pośredniego udostępniają podklasy klasy MQPoolToken w celu przekazania informacji do niestandardowej puli połączeń. Można je konstruować i przekazywać do metody addConnectionPoolToken() w ten sposób, aby można było przekazywać dodatkowe informacje do puli połączeń.

*Domyślna pula połączeń i wiele komponentów w produkcie IBM MQ classes for Java*
W tym przykładzie przedstawiono sposób dodawania lub usuwania elementu MQPoolTokens ze statycznego zestawu zarejestrowanych obiektów MQPoolToken .

MQEnvironment przechowuje statyczny zestaw zarejestrowanych obiektów MQPoolToken . Aby dodać lub usunąć MQPoolTokens z tego zestawu, należy użyć następujących metod:

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

Aplikacja może składać się z wielu komponentów, które istnieją niezależnie i wykonują pracę przy użyciu menedżera kolejek. W takiej aplikacji każdy komponent powinien dodać MQPoolToken do zestawu MQEnvironment na czas jego życia.

Na przykład przykładowa aplikacja MQApp3 tworzy dziesięć wątków i uruchamia każdy z nich. Każdy wątek rejestruje własny MQPoolToken, oczekuje przez pewien czas, a następnie nawiązuje połączenie z menedżerem kolejek. Po rozłączeniu wątek usuwa własny MQPoolToken.

Domyślna pula połączeń pozostaje aktywna, gdy w zestawie MQPoolTokensznajduje się co najmniej jeden znacznik, więc pozostaje ona aktywna przez czas trwania tej aplikacji. Aplikacja nie musi utrzymywać obiektu głównego w ogólnej kontroli nad wątkami.

```
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);
      }
}
```

*Podawanie innej puli połączeń w programie IBM MQ classes for Java*
W tym przykładzie przedstawiono sposób użycia klasy **com.ibm.mq.MQSimpleConnectionManager** do dostarczenia innej puli połączeń.

Ta klasa udostępnia podstawowe narzędzia do zestawiania połączeń, a aplikacje mogą używać tej klasy do dostosowywania zachowania puli.

Po utworzeniu instancji menedżera MQSimpleConnectionmożna określić w konstruktorze MQQueueManager . Następnie menedżer MQSimpleConnectionzarządza połączeniem, które jest podstawą utworzonego menedżera MQQueueManager. Jeśli menedżer MQSimpleConnectionzawiera

odpowiednie połączenie z puli, jest ono ponownie wykorzystywane i zwracane do menedżera MQSimpleConnectionpo wywołaniu metody MQQueueManager.disconnect ().

Poniższy fragment kodu demonstruje to zachowanie:

```
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);
```

Połączenie, które zostało sfałszowane podczas pierwszego konstruktora MQQueueManager , jest przechowywane w programie myConnMan po wywołaniu metody qmgr.disconnect(). Połączenie jest następnie ponownie wykorzystywane podczas drugiego wywołania konstruktora MQQueueManager .

Drugi wiersz włącza menedżera MQSimpleConnection. Ostatni wiersz wyłącza menedżera MQSimpleConnection, niszcząc wszystkie połączenia utrzymywane w puli. Menedżer MQSimpleConnectionjest domyślnie w trybie MODE_AUTO, który został opisany w dalszej części tej sekcji.

Menedżer MQSimpleConnectionprzydziela połączenia w oparciu o ostatnio używane i niszczy połączenia w oparciu o najdawniej używane. Domyślnie połączenie jest niszczone, jeśli nie było używane przez pięć minut lub jeśli w puli jest więcej niż dziesięć nieużywanych połączeń. Wartości te można zmienić, wywołując metodę MQSimpleConnectionManager.setTimeout().

Można również skonfigurować menedżer MQSimpleConnectionjako domyślną pulę połączeń, która będzie używana, gdy w konstruktorze MQQueueManager nie zostanie dostarczony żaden menedżer połączeń.

Demonstruje to następująca aplikacja:

```
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);
      }
}
```

Pogrubione linie tworzą i konfigurują obiekt menedżera MQSimpleConnection. Konfiguracja wykonuje następujące czynności:

- Kończy połączenia, które nie są używane przez godzinę
- Ogranicza liczbę połączeń zarządzanych przez program myConnMan do 75
- Ogranicza liczbę nieużywanych połączeń w puli do 50
- Ustawia tryb MODE_AUTO, który jest wartością domyślną. Oznacza to, że pula jest aktywna tylko wtedy, gdy jest domyślnym menedżerem połączeń, a w zestawie MQPoolTokens znajdującym się w środowisku MQEnvironment znajduje się co najmniej jeden znacznik.

Nowy menedżer MQSimpleConnectionjest następnie ustawiany jako domyślny menedżer połączeń.

W ostatnim wierszu aplikacja wywołuje funkcję MQApp3.main(). Powoduje to uruchomienie pewnej liczby wątków, w których każdy wątek używa IBM MQ niezależnie od siebie. Wątki te używają programu myConnMan podczas tworzenia połączeń.

## *Koordynacja JTA/JDBC przy użyciu IBM MQ classes for Java*

Produkt IBM MQ classes for Java obsługuje metodę MQQueueManager.begin (), która umożliwia produktowi IBM MQ działanie jako koordynator bazy danych udostępniającej sterownik JDBC typu 2 lub JDBC typu 4.

Ta obsługa nie jest dostępna na wszystkich platformach. Aby sprawdzić, które platformy obsługują koordynację JDBC , należy zapoznać się z sekcją Wymagania systemowe produktu IBM MQ.

Aby korzystać z obsługi agenta XA-JTA, należy użyć specjalnej biblioteki przełącznika JTA. Metoda korzystania z tej biblioteki zależy od tego, czy używany jest system Windows , czy jedna z innych platform.

*Konfigurowanie koordynacji JTA/JDBC w systemie Windows*

Biblioteka XA jest dostarczana jako biblioteka DLL o nazwie w formacie `jdbcxxx.dll`.

Dostarczona baza danych `jdbcora12.dll` zapewnia kompatybilność z bazą danych Oracle 12Cw przypadku instalacji serwera IBM MQ for Windows .

W systemach Windows biblioteka XA jest dostarczana jako pełna biblioteka DLL. Nazwa tej biblioteki DLL to `jdbcxxx.dll` , gdzie `xxx` wskazuje bazę danych, dla której skompilowano bibliotekę przełącznika. Ta biblioteka znajduje się w katalogu `java\lib\jdbc` lub `java\lib64\jdbc` instalacji produktu IBM MQ classes for Java . Należy zadeklarować w menedżerze kolejek bibliotekę XA, która jest również opisana jako plik ładowania przełącznika. Należy używać komponentu IBM MQ Explorer. Określ szczegóły pliku ładowania przełącznika w panelu właściwości menedżera kolejek w obszarze menedżera zasobów XA. Należy podać tylko nazwę biblioteki. Na przykład:

W przypadku bazy danych Db2 ustaw w polu SwitchFile wartość: `dbcdb2`

W przypadku bazy danych Oracle ustaw w polu SwitchFile wartość: `jdbcora`

**Uwagi:**

1. Baza danych Oracle 12C jest obsługiwana przez IBM MQ classes for Java, tylko w systemie IBM MQ dla Windows.
2. Obsługiwana wersja produktu Oracle 12C to 12.1.0.1.0 Enterprise Edition i przyszłe pakiety poprawek.
3. Oracle w 64-bitowym systemie Windows wymagają 32-bitowego klienta Oracle .
4. Za pomocą IBM MQ classes for Java IBM MQ może działać jako koordynator transakcji. Nie jest jednak możliwe uczestniczenie w transakcji w stylu JTA.

*Konfigurowanie koordynacji JTA/JDBC na platformach innych niż Windows*

Dostarczane są zbiory obiektów. Dowiąż odpowiedni plik przy użyciu dostarczonego pliku makefile i zadeklaruj go do menedżera kolejek przy użyciu pliku konfiguracyjnego.

Dla każdego systemu zarządzania bazami danych produkt IBM MQ udostępnia dwa pliki obiektów. Należy dowiązać jeden plik wynikowy, aby utworzyć 32-bitową bibliotekę przełącznika, a drugi plik wynikowy, aby utworzyć 64-bitową bibliotekę przełącznika. W systemie Db2każdy plik obiektu ma nazwę `jdbcdb2.o`, a w systemie Oracle każdy plik obiektu ma nazwę `jdbcora.o`.

Każdy plik obiektu należy dowiązać za pomocą odpowiedniego pliku makefile dostarczonego z produktem IBM MQ. Biblioteka przełącznika wymaga innych bibliotek, które mogą być przechowywane w różnych miejscach w różnych systemach. Jednak biblioteka przełącznika nie może użyć zmiennej środowiskowej ścieżki do biblioteki w celu znalezienia tych bibliotek, ponieważ biblioteka przełącznika jest ładowana przez menedżer kolejek, który działa w środowisku produktu `setuid` . Dostarczony plik makefile zapewnia, że biblioteka przełącznika zawiera pełne nazwy ścieżek do tych bibliotek.

Aby utworzyć bibliotekę przełącznika, wprowadź komendę **make** w następującym formacie. Aby utworzyć 32-bitową bibliotekę przełącznika, wprowadź komendę w katalogu `/java/lib/jdbc` instalacji produktu

IBM MQ . Aby utworzyć 64-bitową bibliotekę przełącznika, wprowadź komendę w katalogu `/java/lib64/jdbc` .

```
make DBMS
```

gdzie *DBMS* to system zarządzania bazami danych, dla którego tworzona jest biblioteka przełącznika. Poprawne wartości to db2 dla Db2 i `oracle` dla Oracle.

**Uwaga:**

- Aby uruchomić aplikacje 32-bitowe, należy utworzyć zarówno 32-bitową, jak i 64-bitową bibliotekę przełącznika dla każdego używanego systemu zarządzania bazami danych. Aby uruchomić aplikacje 64-bitowe, należy utworzyć tylko 64-bitową bibliotekę przełącznika. W przypadku systemu Db2nazwa każdej biblioteki przełącznika to `jdbcdb2` , a w przypadku systemu Oraclenazwa każdej biblioteki przełącznika to `jdbcora`. Pliki makefile zapewniają, że 32-bitowe i 64-bitowe biblioteki przełączników są przechowywane w różnych katalogach IBM MQ . 32-bitowa biblioteka przełącznika jest przechowywana w katalogu `/java/lib/jdbc` , a 64-bitowa biblioteka przełącznika jest przechowywana w katalogu `/java/lib64/jdbc` .
- Ponieważ produkt Oracle można zainstalować w dowolnym miejscu w systemie, pliki makefile używają zmiennej środowiskowej **ORACLE_HOME** , aby znaleźć miejsce instalacji produktu Oracle .
- Jeśli produkt IBM MQ jest zainstalowany w położeniu innym niż domyślne, zmień wartość **MQ_INSTALLATION_PATH** w pliku makefile.

Po utworzeniu bibliotek przełącznika dla produktu Db2, Oraclelub obu tych elementów należy zadeklarować je w menedżerze kolejek. Jeśli plik konfiguracyjny menedżera kolejek (qm.ini) zawiera już sekcje `XAResourceManager` dla baz danych Db2 lub Oracle , należy zastąpić wpis `SwitchFile` w każdej sekcji jedną z następujących sekcji:

**Dla bazy danych Db2**

```
SwitchFile=jdbcdb2
```

**Dla bazy danych Oracle**

```
SwitchFile=jdbcora
```

Nie należy podawać pełnej nazwy ścieżki 32-bitowej lub 64-bitowej biblioteki przełącznika. Należy podać tylko nazwę biblioteki.

Jeśli plik konfiguracyjny menedżera kolejek nie zawiera jeszcze sekcji `XAResourceManager` dla baz danych Db2 lub Oracle lub jeśli mają zostać dodane dodatkowe sekcje `XAResourceManager` , należy zapoznać się z sekcją Administrowanie programem IBM MQ , która zawiera informacje na temat tworzenia sekcji `XAResourceManager` . Jednak każda pozycja `SwitchFile` w nowej sekcji `XAResourceManager` musi być dokładnie taka sama, jak opisana wcześniej dla bazy danych Db2 lub Oracle . Należy również dołączyć wpis `ThreadOfControl=PROCESS`.

Po zaktualizowaniu pliku konfiguracyjnego menedżera kolejek i upewnieniu się, że zostały ustawione wszystkie odpowiednie zmienne środowiskowe bazy danych, można zrestartować menedżer kolejek.

*Korzystanie z koordynacji JTA/JDBC*
Zakoduj wywołania interfejsu API, tak jak w podanym przykładzie.

Podstawowa sekwencja wywołań API dla aplikacji użytkownika jest następująca:

```
  qMgr = new MQQueueManager("QM1")
  Connection con = qMgr.getJDBCConnection( xads );
  qMgr.begin()

 < Perform MQ and DB operations to be grouped in a unit of work >

  qMgr.commit() or qMgr.backout();
```

```
con.close()
qMgr.disconnect()
```

xads w wywołaniu getJDBCConnection jest specyficzną dla bazy danych implementacją interfejsu XADataSource , który definiuje szczegóły bazy danych, z którą ma zostać nawiązane połączenie. Zapoznaj się z dokumentacją bazy danych, aby określić sposób tworzenia odpowiedniego obiektu XADataSource do przekazania do metody getJDBCConnection.

Należy również zaktualizować ścieżkę klasy przy użyciu odpowiednich plików jar specyficznych dla bazy danych na potrzeby wykonywania pracy z interfejsem JDBC .

Jeśli konieczne jest nawiązanie połączenia z wieloma bazami danych, należy kilkakrotnie wywołać metodę getJDBCConnection , aby wykonać transakcję na kilku różnych połączeniach.

Istnieją dwie formy getJDBCConnection, odzwierciedlające dwie formy XADataSource.getXAConnection:

```
public java.sql.Connection getJDBCConnection(javax.sql.XADataSource xads)
  throws MQException, SQLException, Exception

public java.sql.Connection getJDBCConnection(XADataSource dataSource,
                                             String userid, String password)
  throws MQException, SQLException, Exception
```

Te metody deklarują wyjątek w swoich klauzulach throws, aby uniknąć problemów z weryfikatorem maszyny JVM dla klientów, którzy nie używają funkcji JTA. Rzeczywistym zgłoszonym wyjątkiem jest wyjątek javax.transaction.xa.XAException , który wymaga dodania pliku jta.jar do ścieżki klasy dla programów, które go wcześniej nie wymagały.

Aby korzystać z obsługi interfejsu JTA/JDBC , należy dołączyć do aplikacji następującą instrukcję:

```
MQEnvironment.properties.put(CMQC.THREAD_AFFINITY_PROPERTY, new Boolean(true));
```

*Znane problemy i ograniczenia dotyczące koordynacji JTA/JDBC*
Niektóre problemy i ograniczenia związane z obsługą interfejsu JTA/JDBC zależą od używanego systemu zarządzania bazami danych, na przykład przetestowane sterowniki JDBC zachowują się inaczej, gdy baza danych jest wyłączana podczas działania aplikacji. Jeśli połączenie z bazą danych używaną przez aplikację jest zerwane, istnieją kroki, które aplikacja może wykonać w celu ponownego nawiązania nowego połączenia z menedżerem kolejek i bazą danych, aby mogła następnie użyć tych nowych połączeń do wykonania wymaganej pracy transakcyjnej.

Ponieważ obsługa interfejsu JTA/JDBC wywołuje sterowniki JDBC , implementacja tych sterowników JDBC może mieć znaczący wpływ na zachowanie systemu. W szczególności przetestowane sterowniki JDBC zachowują się inaczej, gdy baza danych jest wyłączona podczas działania aplikacji.

**Ważne:** Zawsze należy unikać nagłego zamykania bazy danych, gdy istnieją aplikacje utrzymujące otwarte połączenia z bazą danych.

**Uwaga:** Aplikacja IBM MQ classes for Java musi nawiązać połączenie przy użyciu trybu powiązań, aby produkt IBM MQ działał jako koordynator bazy danych.

**Wiele sekcji XAResourceManager**
Użycie więcej niż jednej sekcji XAResourceManager w pliku konfiguracyjnym menedżera kolejek qm.ininie jest obsługiwane. Każda sekcja XAResourceManager inna niż pierwsza jest ignorowana.

**Db2**

Czasami program Db2 zwraca błąd SQL0805N . Ten problem można rozwiązać za pomocą następującej komendy CLP:

```
DB2 bind @db2cli.lst blocking all grant public
```

Więcej informacji na ten temat zawiera dokumentacja produktu Db2 .

Sekcja XAResourceManager musi być skonfigurowana do używania ustawienia ThreadOfControl=PROCESS. W przypadku systemu Db2 8.1 lub nowszego nie jest to zgodne z domyślnym wątkiem ustawienia sterowania dla Db2, dlatego parametr toc=p musi być określony w łańcuchu Open String interfejsu XA. Przykładowa sekcja XAResourceManager dla systemu Db2 z koordynacją JTA/JDBC jest następująca:

```
XAResourceManager:
     Name=jdbcdb2
     SwitchFile=jdbcdb2
     XAOpenString=uid=userid,db=dbalias,pwd=password,toc=p
     ThreadOfControl=PROCESS
```

Nie uniemożliwia to wielowątkowości aplikacji Java , które używają koordynacji JTA/JDBC .

**Oracle**
Wywołanie metody JDBC Connection.close() po wywołaniu metody MQQueueManager.disconnect () powoduje wygenerowanie wyjątku SQLException. Wywołaj metodę Connection.close() przed wywołaniem metody MQQueueManager.disconnect () lub pomiń wywołanie metody Connection.close().

## Obsługa problemów z połączeniami z bazą danych

Jeśli aplikacja IBM MQ classes for Java używa obsługi JTA/JDBC udostępnianej przez IBM MQ, zazwyczaj wykonuje następujące kroki:

1. Tworzy nowy obiekt MQQueueManager , który reprezentuje połączenie z menedżerem kolejek i będzie działać jako menedżer transakcji.
2. Tworzy obiekt XADataSource , który zawiera szczegółowe informacje na temat nawiązywania połączenia z bazą danych, która będzie rejesttowana w transakcji.
3. Wywołuje metodę MQQueueManager.getJDBCConnection(XADataSource), przekazując wcześniej utworzone XADataSource . Powoduje to, że IBM MQ classes for Java nawiązuje połączenie z bazą danych.
4. Wywołuje metodę MQQueueManager.begin () w celu uruchomienia transakcji XA.
5. Wykonuje pracę związaną z przesyłaniem komunikatów i bazą danych.
6. Po zakończeniu wszystkich wymaganych prac wywoływana jest metoda MQQueueManager.commit (). Spowoduje to zakończenie transakcji XA.
7. Jeśli w tym momencie wymagana jest nowa transakcja XA, aplikacja może powtórzyć kroki 4, 5 i 6.
8. Po zakończeniu aplikacja powinna zamknąć połączenie z bazą danych utworzone w kroku 3, a następnie wywołać metodę MQQueueManager.disconnect () w celu rozłączenia się z menedżerem kolejek.

IBM MQ classes for Java przechowuje wewnętrzną listę wszystkich połączeń z bazą danych, które zostały utworzone, gdy aplikacja wywołała MQQueueManager.getJDBCConnection(XADataSource). Jeśli menedżer kolejek musi komunikować się z bazą danych podczas przetwarzania transakcji XA, wykonywane jest następujące przetwarzanie:

1. Menedżer kolejek wywołuje program IBM MQ classes for Java, przekazując szczegóły wywołania XA, które musi zostać przekazane do bazy danych.
2. Następnie IBM MQ classes for Java odszukaj odpowiednie połączenie na liście, a następnie użyj tego połączenia do przepływu wywołania XA do bazy danych.

Jeśli połączenie z bazą danych zostanie utracone w dowolnym momencie podczas tego przetwarzania, aplikacja powinna:

1. Wycofał wszystkie istniejące prace wykonane w ramach transakcji, wywołując metodę MQQueueManager.backout ().
2. Zamknij połączenie z bazą danych. Powinno to spowodować usunięcie przez IBM MQ classes for Java szczegółów zerwanego połączenia z bazą danych z listy wewnętrznej.

3. Rozłącz się z menedżerem kolejek, wywołując metodę MQQueueManager.disconnect ().
4. Nawiąż nowe połączenie z menedżerem kolejek, konstruując nowy obiekt MQQueueManager .
5. Utwórz nowe połączenie z bazą danych, wywołując metodę MQQueueManager.getJDBCConnection(XADataSource).
6. Wykonaj ponownie pracę transakcyjną.

Umożliwia to aplikacji ponowne nawiązanie nowego połączenia z menedżerem kolejek i bazą danych, a następnie użycie tych połączeń do wykonania wymaganej pracy transakcyjnej.

## *Obsługa protokołu TLS (Transport Layer Security) w produkcie IBM MQ classes for Java*

Aplikacje klienckie IBM MQ classes for Java obsługują szyfrowanie TLS. Do korzystania z szyfrowania TLS wymagany jest dostawca JSSE.

Aplikacje klienckie IBM MQ classes for Java korzystające z protokołu TRANSPORT (CLIENT) obsługują szyfrowanie TLS. Protokół TLS zapewnia szyfrowanie komunikacji, uwierzytelnianie i integralność komunikatów. Jest on zwykle używany do zabezpieczania komunikacji między dwoma węzłami w sieci Internet lub w obrębie intranetu.

IBM MQ classes for Java używa rozszerzenia Java Secure Socket Extension (JSSE) do obsługi szyfrowania TLS, dlatego wymaga dostawcy JSSE. JSE v1.4 Maszyny JVM mają wbudowanego dostawcę JSSE. Szczegóły dotyczące sposobu zarządzania certyfikatami i ich przechowywania mogą być różne w zależności od dostawcy. Więcej informacji na ten temat zawiera dokumentacja dostawcy JSSE.

W tej sekcji przyjęto założenie, że dostawca JSSE jest poprawnie zainstalowany i skonfigurowany oraz że odpowiednie certyfikaty zostały zainstalowane i udostępnione dostawcy JSSE.

Jeśli aplikacja kliencka systemu IBM MQ classes for Java używa tabeli definicji kanału klienta (CCDT) do nawiązania połączenia z menedżerem kolejek, należy zapoznać się z sekcją “Używanie tabeli definicji kanału klienta z produktem IBM MQ classes for Java” na stronie 388.

*Włączanie protokołu TLS w produkcie IBM MQ classes for Java*

Aby włączyć protokół TLS, należy określić zestaw algorytmów szyfrowania CipherSuite. Istnieją dwa sposoby określania CipherSuite.

Protokół TLS jest obsługiwany tylko dla połączeń klienckich. Aby włączyć obsługę protokołu TLS, należy określić zestaw algorytmów szyfrowania CipherSuite , który ma być używany podczas komunikacji z menedżerem kolejek, a zestaw algorytmów szyfrowania CipherSuite musi być zgodny ze specyfikacją CipherSpec ustawioną w kanale docelowym. Dodatkowo pakiet CipherSuite musi być obsługiwany przez dostawcę JSSE. Jednak CipherSuites różnią się od CipherSpecs i dlatego mają różne nazwy. Plik “TLS CipherSpecs i CipherSuites w produkcie IBM MQ classes for Java” na stronie 422 zawiera tabelę z odwzorowaniem CipherSpecs obsługiwanych przez produkt IBM MQ na odpowiadające im CipherSuites znane w środowisku JSSE.

Aby włączyć obsługę protokołu TLS, należy określić zestaw CipherSuite za pomocą zmiennej elementu statycznego sslCipherSuite w środowisku MQEnvironment. Poniższy przykład przyłącza się do kanału SVRCONN o nazwie SECURE.SVRCONN.CHANNEL, który został skonfigurowany do obsługi protokołu TLS z wartością CipherSpec ustawioną na TLS_RSA_WITH_AES_128_CBC_SHA256:

```
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");
```

Chociaż atrybut CipherSpec kanału ma wartość TLS_RSA_WITH_AES_128_CBC_SHA256, aplikacja Java musi określać parametr CipherSuite o wartości SSL_RSA_WITH_AES_128_CBC_SHA256. Listę odwzorowań między CipherSpecs i CipherSuiteszawiera sekcja “TLS CipherSpecs i CipherSuites w produkcie IBM MQ classes for Java” na stronie 422 .

Aplikacja może również określić CipherSuite , ustawiając właściwość środowiska CMQC.SSL_CIPHER_SUITE_PROPERTY.

Alternatywnie można użyć tabeli definicji kanału klienta (CCDT). Więcej informacji na ten temat zawiera sekcja “Używanie tabeli definicji kanału klienta z produktem IBM MQ classes for Java” na stronie 388.

Jeśli wymagane jest połączenie klienckie z użyciem pakietu CipherSuite obsługiwanego przez dostawcę IBM Java JSSE FIPS (IBMJSSEFIPS), aplikacja może ustawić pole wymagane sslFipsw klasie MQEnvironment na wartość `true`. Alternatywnie aplikacja może ustawić właściwość środowiska CMQC.SSL_FIPS_REQUIRED_PROPERTY. Wartością domyślną jest `false`, co oznacza, że połączenie klienckie może korzystać z dowolnego CipherSuite obsługiwanego przez produkt IBM MQ.

Jeśli aplikacja używa więcej niż jednego połączenia klienckiego, wartość pola sslFipsRequired (Wymagane), które jest używane podczas tworzenia pierwszego połączenia klienckiego, określa wartość używaną podczas tworzenia kolejnego połączenia klienckiego przez aplikację. Dlatego gdy aplikacja tworzy kolejne połączenie klienckie, wartość w polu sslFipsRequired (Wymagane) jest ignorowana. Należy zrestartować aplikację, jeśli wymagane jest użycie innej wartości w polu sslFips.

Aby pomyślnie nawiązać połączenie przy użyciu protokołu TLS, magazyn zaufanych certyfikatów JSSE musi być skonfigurowany z certyfikatami głównymi ośrodka certyfikacji, z których można uwierzytelnić certyfikat prezentowany przez menedżer kolejek. Podobnie, jeśli opcja SSLClientAuth w kanale SVRCONN ma wartość MQSSL_CLIENT_AUTH_REQUIRED, magazyn kluczy JSSE musi zawierać certyfikat identyfikujący, który jest zaufany przez menedżer kolejek.

**Odsyłacze pokrewne**

Standardy FIPS (Federal Information Processing Standards) dla AIX, Linux, and Windows

*Używanie nazwy wyróżniającej menedżera kolejek w programie IBM MQ classes for Java*

Menedżer kolejek identyfikuje się za pomocą certyfikatu TLS, który zawiera nazwę wyróżniającą (DN). Aplikacja kliencka IBM MQ classes for Java może użyć tej nazwy wyróżniającej, aby upewnić się, że komunikuje się z poprawnym menedżerem kolejek.

Wzorzec nazwy wyróżniającej jest określany za pomocą zmiennej nazwy sslPeerśrodowiska MQEnvironment. Na przykład ustawienie:

```
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

umożliwia pomyślne nawiązanie połączenia tylko wtedy, gdy menedżer kolejek przedstawi certyfikat o nazwie zwykłej rozpoczynającej się od łańcucha QMGR., oraz co najmniej dwie nazwy jednostek organizacyjnych, z których pierwsza musi być IBM , a druga WebSphere.

Jeśli ustawiono nazwę sslPeer, połączenia są nawiązywane pomyślnie tylko wtedy, gdy ustawiono poprawny wzorzec, a menedżer kolejek przedstawia zgodny certyfikat.

Aplikacja może również określić nazwę wyróżniającą menedżera kolejek, ustawiając właściwość środowiska CMQC.SSL_PEER_NAME_PROPERTY. Więcej informacji na temat nazw wyróżniających zawiera sekcja Nazwy wyróżniające.

*Korzystanie z list odwołań certyfikatów w programie IBM MQ classes for Java*

Określ listy odwołań certyfikatów, które mają być używane za pośrednictwem klasy java.security.cert.CertStore . Następnie program IBM MQ classes for Java sprawdza certyfikaty przy użyciu określonej listy CRL.

Lista odwołań certyfikatów (CRL) jest zestawem certyfikatów, które zostały unieważnione przez wydający ośrodek certyfikacji lub przez organizację lokalną. Listy CRL są zwykle udostępniane na serwerach LDAP. W przypadku wersji Java 2 v1.4serwer CRL może zostać określony w czasie połączenia, a certyfikat prezentowany przez menedżer kolejek jest porównywany z listą CRL, zanim połączenie zostanie dozwolone. Więcej informacji na temat list odwołań certyfikatów i IBM MQzawiera sekcja Praca z listami odwołań certyfikatów i listami odwołań uprawnień oraz sekcja Uzyskiwanie dostępu do list odwołań certyfikatów i ARL w systemach IBM MQ classes for Java i IBM MQ classes for JMS.

**Uwaga:** Aby pomyślnie użyć CertStore z listą CRL udostępnioną na serwerze LDAP, upewnij się, że pakiet Java Software Development Kit (SDK) jest zgodny z listą CRL. Niektóre pakiety SDK wymagają, aby lista CRL była zgodna z dokumentem RFC 2587, który definiuje schemat dla LDAP v2. Większość serwerów LDAP v3 używa standardu RFC 2256.

Używane listy CRL są określone przez klasę java.security.cert.CertStore . Szczegółowe informacje na temat uzyskiwania instancji klasy CertStoremożna znaleźć w dokumentacji tej klasy. Aby utworzyć CertStore na podstawie serwera LDAP, należy najpierw utworzyć instancję parametrów LDAPCertStore, zainicjowanych ustawieniami serwera i portu, które mają być używane. Na przykład:

```
import java.security.cert.*;
CertStoreParameters csp = new LDAPCertStoreParameters("crl_server", 389);
```

Po utworzeniu instancji parametrów CertStorenależy użyć konstruktora statycznego w konstruktorze CertStore , aby utworzyć CertStore typu LDAP:

```
CertStore cs = CertStore.getInstance("LDAP", csp);
```

Obsługiwane są również inne typy CertStore (na przykład Collection). Zwykle istnieje kilka serwerów CRL z identycznymi informacjami CRL w celu zapewnienia nadmiarowości. Jeśli dla każdego z tych serwerów CRL istnieje obiekt CertStore , umieść je wszystkie w odpowiedniej kolekcji. W poniższym przykładzie przedstawiono obiekty CertStore umieszczone na liście ArrayList:

```
import java.util.ArrayList;
Collection crls = new ArrayList();
crls.add(cs);
```

Tę kolekcję można ustawić w zmiennej statycznej środowiska MQEnvironment, magazynach sslCert, przed nawiązaniem połączenia w celu włączenia sprawdzania listy CRL:

```
MQEnvironment.sslCertStores = crls;
```

Poprawność certyfikatu prezentowanego przez menedżer kolejek podczas konfigurowania połączenia jest sprawdzana w następujący sposób:

1. Pierwszy obiekt CertStore w kolekcji identyfikowany przez magazyny sslCertjest używany do identyfikacji serwera CRL.
2. Podjęto próbę skontaktowania się z serwerem CRL.
3. Jeśli próba się powiedzie, serwer zostanie wyszukany pod kątem zgodności certyfikatu.
   a. Jeśli certyfikat zostanie unieważniony, proces wyszukiwania zostanie zakończony, a żądanie połączenia zakończy się niepowodzeniem z kodem przyczyny MQRC_SSL_CERTIFICATE_ODWOŁANE.
   b. Jeśli certyfikat nie zostanie znaleziony, proces wyszukiwania zostanie zakończony i połączenie będzie kontynuowane.
4. Jeśli próba nawiązania kontaktu z serwerem nie powiedzie się, następny obiekt CertStore zostanie użyty do zidentyfikowania serwera CRL, a proces zostanie powtórzony w kroku 2.

   Jeśli była to ostatnia CertStore w kolekcji lub jeśli kolekcja nie zawiera obiektów CertStore , proces wyszukiwania nie powiódł się i żądanie połączenia zakończyło się niepowodzeniem z kodem przyczyny MQRC_SSL_CERT_STORE_ERROR.

Obiekt Collection określa kolejność, w jakiej używane są CertStores .

Kolekcję CertStores można również ustawić za pomocą właściwości CMQC.SSL_CERT_STORE_PROPERTY. Dla wygody ta właściwość umożliwia również określenie pojedynczej składnicy CertStore bez konieczności bycia elementem kolekcji.

Jeśli dla magazynu sslCertustawiono wartość NULL, sprawdzanie listy CRL nie jest wykonywane. Ta właściwość jest ignorowana, jeśli nie jest ustawiony zestaw sslCipher.

*Renegocjowanie klucza tajnego w produkcie IBM MQ classes for Java*
Aplikacja kliencka IBM MQ classes for Java może kontrolować, kiedy następuje renegocjacja klucza tajnego używanego do szyfrowania połączenia klienckiego, pod względem łącznej liczby wysłanych i odebranych bajtów.

Aplikacja może to zrobić w jeden z następujących sposobów: jeśli aplikacja używa więcej niż jednego z tych sposobów, mają zastosowanie zwykłe reguły pierwszeństwa.

- Przez ustawienie pola liczby sslResetw klasie MQEnvironment.
- Przez ustawienie właściwości środowiska MQC.SSL_RESET_COUNT_PROPERTY w obiekcie Hashtable. Następnie aplikacja przypisuje tabelę mieszającą do pola `properties` w klasie MQEnvironment lub przekazuje ją do obiektu MQQueueManager w jej konstruktorze.

Wartość pola sslResetlub właściwości środowiska MQC.SSL_RESET_COUNT_PROPERTY reprezentuje łączną liczbę bajtów wysłanych i odebranych przez kod klienta IBM MQ classes for Java przed ponownym negocjowaniem klucza tajnego. Liczba wysłanych bajtów jest liczbą przed szyfrowaniem, a liczba odebranych bajtów jest liczbą po deszyfrowaniu. Liczba bajtów obejmuje również informacje sterujące wysyłane i odbierane przez klienta IBM MQ classes for Java .

Jeśli licznik resetowania jest równy zero, co jest wartością domyślną, klucz tajny nigdy nie jest renegocjowany. Licznik resetowania jest ignorowany, jeśli nie określono opcji CipherSuite .

*Podawanie niestandardowej fabryki SSLSocketFactory w pliku IBM MQ classes for Java*
Jeśli używana jest dostosowana fabryka gniazd JSSE, należy ustawić właściwość MQEnvironment.sslSocketFactory na dostosowany obiekt fabryki. Szczegóły różnią się w zależności od różnych implementacji JSSE.

Różne implementacje JSSE mogą udostępniać różne funkcje. Na przykład wyspecjalizowana implementacja JSSE może umożliwiać skonfigurowanie konkretnego modelu sprzętu szyfrującego. Ponadto niektórzy dostawcy JSSE umożliwiają dostosowywanie magazynów kluczy i magazynów zaufanych certyfikatów według programu lub umożliwiają zmianę wyboru certyfikatu tożsamości z magazynu kluczy. W środowisku JSSE wszystkie te dostosowania są wyodrębnione do klasy fabryki javax.net.ssl.SSLSocketFactory.

Szczegółowe informacje na temat tworzenia niestandardowej implementacji SSLSocketFactory można znaleźć w dokumentacji rozszerzenia JSSE. Szczegóły różnią się w zależności od dostawcy, ale typowa sekwencja kroków może być następująca:

1. Tworzenie obiektu SSLContext przy użyciu metody statycznej w kontekście SSLContext
2. Zainicjuj ten kontekst SSL z odpowiednimi implementacjami KeyManager i TrustManager (utworzonymi na podstawie ich własnych klas fabrycznych)
3. Tworzenie fabryki SSLSocketFactory na podstawie kontekstu SSLContext

Jeśli używany jest obiekt SSLSocketFactory , ustaw wartość właściwości MQEnvironment.sslSocketFactory na dostosowany obiekt fabryki. Na przykład:

```
javax.net.ssl.SSLSocketFactory sf = sslContext.getSocketFactory();
MQEnvironment.sslSocketFactory = sf;
```

IBM MQ classes for Java użyj tej fabryki SSLSocketFactory , aby nawiązać połączenie z menedżerem kolejek produktu IBM MQ . Tę właściwość można również ustawić za pomocą właściwości CMQC.SSL_SOCKET_FACTORY_PROPERTY. Jeśli fabryka sslSocketjest ustawiona na wartość NULL, używana jest domyślna fabryka SSLSocketFactory maszyny JVM. Ta właściwość jest ignorowana, jeśli nie jest ustawiony zestaw sslCipher.

W przypadku używania niestandardowych fabryk SSLSocketFactoriesnależy wziąć pod uwagę wpływ współużytkowania połączeń TCP/IP. Jeśli współużytkowanie połączenia jest możliwe, nowe gniazdo nie jest żądane od podanej fabryki SSLSocketFactory , nawet jeśli utworzone gniazdo byłoby inne w pewien sposób w kontekście kolejnego żądania połączenia. Jeśli na przykład w kolejnym połączeniu ma być prezentowany inny certyfikat klienta, współużytkowanie połączenia nie może być dozwolone.

*Wprowadzanie zmian w magazynie kluczy lub magazynie zaufanych certyfikatów JSSE w pliku IBM MQ classes for Java*
W przypadku zmiany magazynu kluczy lub magazynu zaufanych certyfikatów JSSE należy wykonać pewne działania, aby zmiany zostały uwzględnione.

Jeśli zawartość magazynu kluczy lub magazynu zaufanych certyfikatów JSSE zostanie zmieniona lub zostanie zmienione położenie pliku kluczy lub magazynu zaufanych certyfikatów, aplikacje IBM MQ classes for Java , które są uruchomione w tym czasie, nie będą automatycznie wprowadzać zmian. Aby zmiany odniosły skutek, należy wykonać następujące czynności:

- Aplikacje muszą zamknąć wszystkie swoje połączenia i zniszczyć nieużywane połączenia w pulach połączeń.
- Jeśli dostawca JSSE buforuje informacje z magazynu kluczy i magazynu zaufanych certyfikatów, należy je odświeżyć.

Po wykonaniu tych działań aplikacje mogą ponownie utworzyć swoje połączenia.

W zależności od sposobu projektowania aplikacji i funkcji udostępnianej przez dostawcę JSSE, może być możliwe wykonanie tych działań bez zatrzymywania i restartowania aplikacji. Jednak zatrzymanie i zrestartowanie aplikacji może być najprostszym rozwiązaniem.

*Obsługa błędów podczas używania protokołu TLS z produktem IBM MQ classes for Java*
Program IBM MQ classes for Java może użyć wielu kodów przyczyny podczas nawiązywania połączenia z menedżerem kolejek przy użyciu protokołu TLS.

Zostały one wyjaśnione na poniższej liście:

**MQRC_SSL_NOT_ALLOWED (NIEWŁĄCZONY)**
Ustawiono właściwość pakietu sslCipher, ale użyto połączenia powiązań. Tylko połączenie klienta obsługuje protokół TLS.

**BŁĄD MQRC_JSSE_ERROR**
Dostawca JSSE zgłosił błąd, który nie został obsłużony przez IBM MQ. Przyczyną może być problem z konfiguracją funkcji JSSE lub brak możliwości sprawdzenia poprawności certyfikatu prezentowanego przez menedżer kolejek. Wyjątek generowany przez rozszerzenie JSSE można pobrać przy użyciu metody getCause() w przypadku wyjątku MQException.

**MQRC_SSL_INITIALIZATION_ERROR (błąd)**
Wywołano wywołanie MQCONN lub MQCONNX z określonymi opcjami konfiguracyjnymi TLS, ale wystąpił błąd podczas inicjowania środowiska TLS.

**MQRC_SSL_PEER_NAME_MISMATCH**
Wzorzec nazwy wyróżniającej określony we właściwości nazwy sslPeernie jest zgodny z nazwą wyróżniającą przedstawioną przez menedżer kolejek.

**MQRC_SSL_PEER_NAME_ERROR BŁĄD**
Wzorzec nazwy wyróżniającej podany we właściwości nazwy sslPeerjest niepoprawny.

**MQRC_UNSUPPORTED_CIPHER_SUITE,**
Zestaw CipherSuite o nazwie sslCipherSuite nie został rozpoznany przez dostawcę JSSE. Pełna lista CipherSuites obsługiwanych przez dostawcę JSSE może zostać uzyskana przez program przy użyciu metody SSLSocketFactory.getSupportedCipherSuites(). Lista CipherSuites , których można użyć do komunikacji z produktem IBM MQ , znajduje się w katalogu "TLS CipherSpecs i CipherSuites w produkcie IBM MQ classes for Java" na stronie 422.

**MQRC_SSL_CERTIFICATE_ODWOŁANO**
Certyfikat prezentowany przez menedżer kolejek został znaleziony na liście CRL określonej za pomocą właściwości sslCertStores. Zaktualizuj menedżer kolejek, aby używał zaufanych certyfikatów.

**BŁĄD MQRC_SSL_CERT_STORE_ERROR**
Żaden z podanych CertStores nie mógł zostać przeszukany w poszukiwaniu certyfikatu prezentowanego przez menedżera kolejek. Metoda MQException.getCause() zwraca błąd, który wystąpił podczas przeszukiwania pierwszej próby wykonania operacji CertStore . Jeśli przyczyną jest wyjątek NoSuchElementException, ClassCastlub wyjątek NullPointer, sprawdź, czy kolekcja określona we właściwości sslCertStores zawiera co najmniej jeden poprawny obiekt CertStore .

*TLS CipherSpecs i CipherSuites w produkcie IBM MQ classes for Java*
Możliwość nawiązywania połączeń z menedżerem kolejek przez aplikacje produktu IBM MQ classes for Java zależy od parametru CipherSpec określonego na końcu kanału MQI serwera i parametru CipherSuite określonego na końcu klienta.

Poniższa tabela zawiera listę CipherSpecs obsługiwanych przez produkt IBM MQ oraz ich odpowiedniki w sekcji CipherSuites.

Deprecated Należy zapoznać się z tematem Nieaktualne specyfikacje szyfrowania CipherSpecs , aby sprawdzić, czy którekolwiek z CipherSpecswymienionych w poniższej tabeli zostały uznane za nieaktualne przez produkt IBM MQ , a jeśli tak, to aktualizacja specyfikacji szyfrowania CipherSpec stała się nieaktualna.

**Ważne:** Wymienione CipherSuites są obsługiwane przez środowisko wykonawcze IBM Java Runtime Environment (JRE) dostarczane z produktem IBM MQ. Wymienione CipherSuites obejmują te, które są obsługiwane przez środowisko Oracle Java JRE. Więcej informacji na temat konfigurowania aplikacji do korzystania ze środowiska Oracle Java JRE zawiera sekcja “Konfigurowanie aplikacji do korzystania z odwzorowań IBM Java lub Oracle Java CipherSuite” na stronie 444.

Tabela zawiera również informacje o protokole używanym do komunikacji oraz o tym, czy pakiet CipherSuite jest zgodny ze standardem FIPS 140-2.

**Uwaga:** W systemie AIX, Linux, and Windows IBM MQ zapewnia zgodność ze standardem FIPS 140-2 za pośrednictwem modułu szyfrującego IBM Crypto for C (ICC) . Certyfikat dla tego modułu został przeniesiony do statusu historycznego. Klienci powinni zapoznać się z informacjami w sekcji Certyfikat IBM Crypto for C (ICC) i zapoznać się z poradami NIST. Zastępczy moduł FIPS 140-3 jest obecnie w toku, a jego status można wyświetlić, wyszukując go na liście Moduły NIST CMVP na liście procesów.

Zestawy algorytmów szyfrowania oznaczone jako zgodne ze standardem FIPS 140-2 mogą być używane, jeśli aplikacja nie została skonfigurowana do wymuszania zgodności ze standardem FIPS 140-2, ale jeśli dla aplikacji została skonfigurowana zgodność ze standardem FIPS 140-2 (patrz poniższe uwagi dotyczące konfiguracji), można skonfigurować tylko te CipherSuites , które są oznaczone jako zgodne ze standardem FIPS 140-2. Próba użycia innych CipherSuites spowoduje wystąpienie błędu.

**Uwaga:** Każde środowisko JRE może mieć wielu dostawców zabezpieczeń szyfrowania, z których każdy może wnosić implementację tego samego pakietu CipherSuite. Jednak nie wszyscy dostawcy zabezpieczeń mają certyfikat FIPS 140-2. Jeśli zgodność ze standardem FIPS 140-2 nie jest wymuszana w przypadku aplikacji, możliwe jest użycie niecertyfikowanej implementacji CipherSuite . Implementacje niecertyfikowane mogą nie działać zgodnie ze standardem FIPS 140-2, nawet jeśli zestaw algorytmów szyfrowania CipherSuite teoretycznie spełnia minimalny poziom zabezpieczeń wymagany przez standard. Więcej informacji na temat konfigurowania wymuszania FIPS 140-2 w aplikacjach IBM MQ Java zawierają poniższe uwagi.

Więcej informacji na temat zgodności ze standardami FIPS 140-2 i Suite-B dla CipherSpecs i CipherSuiteszawiera sekcja Określanie specyfikacji szyfrowania CipherSpecs. Konieczne może być również zapoznanie się z informacjami dotyczącymi Federal Information Processing Standardsw Stanach Zjednoczonych.

Aby używać pełnego zestawu algorytmów szyfrowania CipherSuites i działać z certyfikowanym standardem FIPS 140-2 i/lub Suite-B, wymagane jest odpowiednie środowisko JRE. IBM Java 7 Service Refresh 4 z pakietem poprawek 2 lub nowszym IBM JRE zapewnia odpowiednią obsługę TLS 1.2 CipherSuites wymienionych w sekcji Tabela 59 na stronie 423.

Aby można było używać szyfrów TLS 1.3 , środowisko JRE, w którym działa aplikacja, musi obsługiwać protokół TLS 1.3.

**Uwaga:** Aby użyć niektórych CipherSuites, w środowisku JRE należy skonfigurować pliki nieograniczonej strategii. Więcej informacji na temat konfigurowania plików strategii w środowisku SDK lub JRE zawiera temat *IBM Pliki strategii SDK* w publikacji *Security Reference for IBM SDK, Java Technology Edition* dla używanej wersji.

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites*

| **CipherSpec** **"1" na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| ECDHE_ECDSA_NULL_SHA256 | SSL_ECDHE_ECDSA_WITH_NULL_SHA | TLS_ECDHE_ECDSA_WITH_NULL_SHA | TLS 1.2 | no |
| ECDHE_ECDSA_RC4_128_SHA256 | SSL_ECDHE_ECDSA_WITH_RC4_128_SHA | TLS_ECDHE_ECDSA_WITH_RC4_128_SHA | TLS 1.2 | no |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| 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 | yes |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| ECDHE_RSA_NULL_SHA256 | SSL_ECDHE_RSA_WITH_NULL_SHA | TLS_ECDHE_RSA_WITH_NULL_SHA | TLS 1.2 | no |
| ECDHE_RSA_RC4_128_SHA256 | SSL_ECDHE_RSA_WITH_RC4_128_SHA | TLS_ECDHE_RSA_WITH_RC4_128_SHA | TLS 1.2 | no |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_RSA_WITH_3DES_EDE_CBC_SHA “2” na stronie 443 | SSL_RSA_WITH_3DES_EDE_CBC_SHA | TLS_RSA_WITH_3DES_EDE_CBC_SHA | TLS 1.0 | nie “4” na stronie 443 |
| TLS_RSA_WITH_AES_128_CBC_SHA | SSL_RSA_WITH_AES_128_CBC_SHA | TLS_RSA_WITH_AES_128_CBC_SHA | TLS 1.0 | nie “4” na stronie 443 |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_RSA_WITH_AES_128_CBC_SHA256 | SSL_RSA_WITH_AES_128_CBC_SHA256 | TLS_RSA_WITH_AES_128_CBC_SHA256 | TLS 1.2 | nie "4" na stronie 443 |
| TLS_RSA_WITH_AES_128_GCM_SHA256 | SSL_RSA_WITH_AES_128_GCM_SHA256 | TLS_RSA_WITH_AES_128_GCM_SHA256 | TLS 1.2 | nie "4" na stronie 443 |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_RSA_WITH_AES_256_CBC_SHA | SSL_RSA_WITH_AES_256_CBC_SHA | TLS_RSA_WITH_AES_256_CBC_SHA | TLS 1.0 | nie "4" na stronie 443 |
| TLS_RSA_WITH_AES_256_CBC_SHA256 | SSL_RSA_WITH_AES_256_CBC_SHA256 | TLS_RSA_WITH_AES_256_CBC_SHA256 | TLS 1.2 | nie "4" na stronie 443 |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_RSA_WITH_AES_256_GCM_SHA384 | SSL_RSA_WITH_AES_256_GCM_SHA384 | TLS_RSA_WITH_AES_256_GCM_SHA384 | TLS 1.2 | nie "4" na stronie 443 |
| TLS_RSA_WITH_DES_CBC_SHA | SSL_RSA_WITH_DES_CBC_SHA | SSL_RSA_WITH_DES_CBC_SHA | TLS 1.0 | no |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **"1" na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_RSA_WITH_NULL_SHA256 | SSL_RSA_WITH_NULL_SHA256 | TLS_RSA_WITH_NULL_SHA256 | TLS 1.2 | no |
| TLS_RSA_WITH_RC4_128_SHA256 | SSL_RSA_WITH_RC4_128_SHA | SSL_RSA_WITH_RC4_128_SHA | TLS 1.2 | no |
| ANY_TLS12 | *TLS12 | *TLS12 | TLS 1.2 | yes |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_AES_128_GCM_SHA256 “3” na stronie 443 | TLS_AES_128_GCM_SHA256 | TLS_AES_128_GCM_SHA256 | TLS 1.3 | no |
| TLS_AES_256_GCM_SHA384 “3” na stronie 443 | TLS_AES_256_GCM_SHA384 | TLS_AES_256_GCM_SHA384 | TLS 1.3 | no |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_CHACHA20_POLY1305_SHA256 “3” na stronie 443 | TLS_CHACHA20_POLY1305_SHA256 | TLS_CHACHA20_POLY1305_SHA256 | TLS 1.3 | no |
| TLS_AES_128_CCM_SHA256 “3” na stronie 443 | TLS_AES_128_CCM_SHA256 | TLS_AES_128_CCM_SHA256 | TLS 1.3 | no |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** **“1” na stronie 443** | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| TLS_AES_128_CCM_8_SHA256 “3” na stronie 443 | TLS_AES_128_CCM_8_SHA256 | TLS_AES_128_CCM_8_SHA256 | TLS 1.3 | no |
| Dowolna“3” na stronie 443 | *ANY | *ANY | Wielokrotny | no |
| ANY_TLS13 “3” na stronie 443 | *TLS13 | *TLS13 | TLS V13 | no |
| ANY_TLS12_OR_HIGHER “3” na stronie 443 | *TLS12ORHIGHER | *TLS12ORHIGHER | TLS 1.2 i nowsze | no |

*Tabela 59. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki w produkcie CipherSuites (kontynuacja)*

| **CipherSpec** “1” na stronie 443 | **Odpowiednik CipherSuite (środowisko JRE firmyIBM )** | **Odpowiednik CipherSuite (Oracle JRE)** | **Protokół** | **Zgodny ze standardem FIPS 140-2** |
|---|---|---|---|---|
| ANY_TLS13_OR_HIGHER “3” na stronie 443 | *TLS13ORHIGHER | *TLS13ORHIGHER | TLS 1.3 i nowsze | no |

**Uwagi:**

1. Jest to wartość skonfigurowana w kanale w pliku IBM MQ, w tym w tabeli CCDT (binarnej lub JSON).
2. Deprecated CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA jest nieaktualna. Jednak nadal może być używany do przesyłania do 32 GB danych, zanim połączenie zostanie przerwane i zostanie zgłoszony błąd AMQ9288. Aby uniknąć tego błędu, należy unikać używania potrójnego algorytmu szyfrowania DES lub włączyć resetowanie klucza tajnego podczas korzystania z tej CipherSpec.
3. Aby można było używać szyfrów TLS v1.3 , środowisko Java runtime environment (JRE), na którym działa aplikacja, musi obsługiwać protokół TLS v1.3.
4. V 9.3.0.17 V 9.3.5.1 W systemach IBM MQ 9.3.5 CSU 1 i IBM MQ 9.3.0 CSU 17środowisko IBM Java 8 JRE usuwa obsługę wymiany kluczy RSA podczas pracy w trybie FIPS.

## Konfigurowanie zestawów algorytmów szyfrowania i zgodności ze standardami FIPS w aplikacji IBM MQ classes for Java

- Aplikacja, która używa produktu IBM MQ classes for Java , może użyć jednej z dwóch metod ustawienia CipherSuite dla połączenia:
  - W polu sslCipherSuite w klasie MQEnvironment ustaw nazwę CipherSuite .
  - Ustaw właściwość CMQC.SSL_CIPHER_SUITE_PROPERTY w tabeli mieszającej właściwości przekazywanej do konstruktora MQQueueManager na nazwę CipherSuite .
- Aplikacja, która używa IBM MQ classes for Java , może użyć jednej z dwóch metod w celu wymuszenia zgodności ze standardem FIPS 140-2:
  - W polu Wymagane sslFipsustaw wartość true w klasie MQEnvironment.

– Ustaw wartość true dla właściwości CMQC.SSL_FIPS_REQUIRED_PROPERTYin tabeli mieszającej właściwości przekazanej do konstruktora MQQueueManager .

## Konfigurowanie aplikacji do korzystania z odwzorowań IBM Java lub Oracle Java CipherSuite

**Uwaga:**

V 9.3.3 W przypadku produktu Continuous Delivery z produktu IBM MQ 9.3.3 Java właściwość systemowa `com.ibm.mq.cfg.useIBMCipherMappings`, która steruje wykorzystywanym odwzorowaniem, jest usuwana z produktu. W produkcie IBM MQ 9.3.3szyfr może być zdefiniowany jako nazwa CipherSpec lub CipherSuite i jest poprawnie obsługiwany przez produkt IBM MQ. Następujące trzy pliki JAR Jacksona są wymagane, jeśli aplikacja IBM MQ classes for Java tworzy bezpieczne połączenia TLS z menedżerem kolejek:

- `jackson-annotations.jar`
- `jackson-core.jar`
- `jackson-databind.jar`

**Ważne:** Poniższe informacje o systemie `com.ibm.mq.cfg.useIBMCipherMappings` dotyczą tylko systemów Long Term Support i Continuous Delivery przed IBM MQ 9.3.3 .

Istnieje możliwość określenia, czy aplikacja ma używać domyślnych odwzorowań IBM Java CipherSuite na IBM MQ CipherSpec , czy odwzorowań Oracle CipherSuite na IBM MQ CipherSpec . Dlatego można użyć CipherSuites TLS, niezależnie od tego, czy aplikacja używa środowiska JRE firmy IBM , czy środowiska JRE firmy Oracle . Java Właściwość systemowa `com.ibm.mq.cfg.useIBMCipherMappings` określa, które odwzorowania są używane. Właściwość może mieć jedną z następujących wartości:

**Prawda**
Użyj odwzorowań IBM Java CipherSuite na IBM MQ CipherSpec .

Jest to wartość domyślna.

**Fałsz**
Użyj odwzorowań Oracle CipherSuite na IBM MQ CipherSpec .

Więcej informacji na temat używania szyfrów IBM MQ Java i TLS zawiera wpis na blogu MQdev MQ Java, TLS Ciphers, inne niżIBM JRE i APAR IT06775, IV66840, IT09423, IT10837.

## Ograniczenia współdziałania

Niektóre CipherSuites mogą być zgodne z więcej niż jedną specyfikacją szyfrowania produktu IBM MQ CipherSpec, w zależności od używanego protokołu. Jednak obsługiwana jest tylko kombinacja CipherSuite/CipherSpec , która używa wersji TLS określonej w tabeli 1. Próba użycia nieobsługiwanych kombinacji zestawów algorytmów szyfrowania ( CipherSuites i CipherSpecs ) nie powiedzie się i zostanie zgłoszony odpowiedni wyjątek. Instalacje używające dowolnej z tych kombinacji CipherSuite/CipherSpec powinny zostać przeniesione do obsługiwanej kombinacji.

W poniższej tabeli przedstawiono CipherSuites , do których ma zastosowanie to ograniczenie.

*Tabela 60. CipherSuites oraz ich obsługiwane i nieobsługiwane specyfikacje szyfrowania CipherSpecs*

| CipherSuite | Obsługiwany TLS CipherSpec | Nieobsługiwana CipherSpec SSL |
|---|---|---|
| SSL_RSA_WITH_3DES_EDE_CBC_SHA | TLS_RSA_WITH_3DES_EDE_CBC_SHA "1" na stronie 445 | 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 |

**Uwaga:**

1. Deprecated Ta CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA jest nieaktualna. Jednak nadal może być używany do przesyłania do 32 GB danych, zanim połączenie zostanie przerwane i zostanie zgłoszony błąd AMQ9288. Aby uniknąć tego błędu, należy unikać używania potrójnego algorytmu szyfrowania DES lub włączyć resetowanie klucza tajnego podczas korzystania z tej CipherSpec.

### *Uruchamianie aplikacji IBM MQ classes for Java*

Jeśli aplikacja (klasa zawierająca metodę main ()) jest napisana przy użyciu trybu klienta lub trybu powiązań, program należy uruchomić przy użyciu interpretera języka Java .

Użyj komendy:

```
java -Djava.library.path= library_path MyClass
```

gdzie *ścieżka_biblioteki* jest ścieżką do bibliotek IBM MQ classes for Java . Więcej informacji na ten temat zawiera sekcja “IBM MQ classes for Java biblioteki” na stronie 369.

**Zadania pokrewne**

Śledzenie aplikacji IBM MQ classes for Java

Śledzenie adaptera zasobów IBM MQ

## Zachowanie zależne od środowiska IBM MQ classes for Java

IBM MQ classes for Java umożliwia tworzenie aplikacji, które mogą być uruchamiane dla różnych wersji produktu IBM MQ. W tej kolekcji tematów opisano zachowanie klas Java zależnych od tych różnych wersji.

Produkt IBM MQ classes for Java udostępnia rdzeń klas, który zapewnia spójne funkcje i zachowanie we wszystkich środowiskach. Funkcje spoza tego podstawowego modułu zależą od możliwości menedżera kolejek, z którym połączona jest aplikacja.

Z wyjątkiem sytuacji opisanych w tym miejscu zachowanie jest zgodne z opisem w sekcji Odwołanie do aplikacji MQI odpowiedniej dla menedżera kolejek.

### *Klasy podstawowe w produkcie IBM MQ classes for Java*

Plik IBM MQ classes for Java zawiera podstawowy zestaw klas, które mogą być używane we wszystkich środowiskach.

Poniższy zestaw klas jest klasami podstawowymi i może być używany we wszystkich środowiskach, w których występują tylko niewielkie odmiany wymienione w sekcji “Ograniczenia i warianty dla klas podstawowych IBM MQ classes for Java” na stronie 446.

- Środowisko MQ
- Wyjątek MQException
- Opcje MQGetMessage

  Wykluczanie:

  – MatchOptions
  – GroupStatus
  – SegmentStatus
  – Segmentacja
- MQManagedObject

  Wykluczanie:

  – Funkcja inquire ()
  – Funkcja set ()
- Komunikat MQ

Wykluczanie:

- groupId
- messageFlags
- messageSequenceNumer
- Przesunięcie
- originalLength

- MQPoolServices
- Zdarzenie MQPoolServices
- MQPoolServicesEventListener
- MQPoolToken
- Opcje MQPutMessage

  Wykluczanie:

  - Liczba: knownDest
  - Liczba: unknownDest
  - Liczba: invalidDest
  - recordFields
- Proces MQProcess
- MQQUEUE
- MQQueueManager

  Wykluczanie:

  - Funkcja begin ()
  - accessDistributionList ()
- Menedżer MQSimpleConnection
- Temat MQ
- ZMQC

**Uwaga:**

1. Niektóre stałe nie są uwzględniane w rdzeniu (szczegółowe informacje można znaleźć w sekcji "Ograniczenia i warianty dla klas podstawowych IBM MQ classes for Java" na stronie 446 ); nie należy ich używać w programach w pełni przenośnych.
2. Niektóre platformy nie obsługują wszystkich trybów połączenia. Na tych platformach można używać tylko klas podstawowych i opcji powiązanych z obsługiwanymi trybami.

## *Ograniczenia i warianty dla klas podstawowych IBM MQ classes for Java*

Klasy podstawowe zwykle zachowują się spójnie we wszystkich środowiskach, nawet jeśli w przypadku równoważnych wywołań MQI występują zwykle różnice w środowisku. Zachowanie jest takie, jakby używany był menedżer kolejek w systemie AIX, Linuxlub Windows , z wyjątkiem następujących niewielkich ograniczeń i wariantów.

*Ograniczenia dotyczące wartości MQGMO_ * w produkcie IBM MQ classes for Java*
Niektóre wartości MQGMO_ * nie są obsługiwane przez wszystkie menedżery kolejek.

Użycie następujących wartości MQGMO_ * może spowodować zgłoszenie wyjątku MQException z metody MQQueue.get():

MQGMO_SYNCPOINT_IF_PERSISTENT,
MQGMO_MARK_SKIP_BACKOUT
MQGMO_BROWSE_MSG_UNDER_CURSOR
BLOKADA MQGMO_LOCK

MQGMO_UNLOCK (odblokowanie MQGMO)
MQGMO_LOGICAL_ORDER (KOLEJNA_KOLEJNA_STRUKTURA)
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

Dodatkowo opcja MQGMO_SET_SIGNAL nie jest obsługiwana, jeśli jest używana z produktu Java.

*Ograniczenia dotyczące wartości MQPMRF_ * w produkcie IBM MQ classes for Java*
Są one używane tylko podczas umieszczania komunikatów na liście dystrybucyjnej i są obsługiwane tylko przez menedżery kolejek obsługujące listy dystrybucyjne. Na przykład menedżery kolejek systemu z/OS nie obsługują list dystrybucyjnych.

*Ograniczenia dotyczące wartości MQPMO_ * w produkcie IBM MQ classes for Java*
Niektóre wartości MQPMO_ * nie są obsługiwane przez wszystkie menedżery kolejek

Użycie następujących wartości MQPMO_ * może spowodować zgłoszenie wyjątku MQException z metody MQQueue.put() lub MQQueueManager.put ():

MQPMO_LOGICAL_ORDER,
MQPMO_NEW_CORREL_ID
MQPMO_NEW_MESSAGE_ID
MQPMO_RESOLVE_LOCAL_Q,

*Ograniczenia i warianty wartości MQCNO_ * w sekcji IBM MQ classes for Java*
Niektóre wartości `MQCNO_*` nie są obsługiwane.

- Automatyczne ponowne łączenie klienta nie jest obsługiwane przez IBM MQ classes for Java. Niezależnie od ustawionej wartości `MQCNO_RECONNECT_*` połączenie zachowuje się tak, jakby została ustawiona wartość `MQCNO_RECONNECT_DISABLED`.
- Parametr `MQCNO_FASTPATH` jest ignorowany w przypadku menedżerów kolejek, które nie obsługują produktu `MQCNO_FASTPATH`. Jest on również ignorowany przez połączenia klienckie.

*Ograniczenia dotyczące wartości MQRO_ * w produkcie IBM MQ classes for Java*
Można ustawić następujące opcje raportu.

MQRO_EXCEPTION_WITH_FULL_DATA
MQRO_EXPIRATION_WITH_FULL_DATA
MQRO_KOA_WITH_FULL_DATA
MQRO_KOD_WITH_FULL_DATA
MQRO_DISCARD_MSG
MQRO_PASS_DISCARD_AND_WAŻNOŚCI

Więcej informacji na ten temat zawiera sekcja Raport.

*Różne różnice między systemem IBM MQ classes for Java na platformie z/OS i innymi platformami*
W niektórych obszarach działanie programu IBM MQ for z/OS różni się od działania programu IBM MQ na innych platformach.

**BackoutCount**
Menedżer kolejek systemu z/OS zwraca maksymalną liczbę BackoutCount równą 255, nawet jeśli komunikat został wycofany więcej niż 255 razy.

**Domyślny przedrostek kolejki dynamicznej**
W przypadku połączenia z menedżerem kolejek produktu z/OS przy użyciu połączenia powiązań domyślnym przedrostkiem kolejki dynamicznej jest CSQ. *. W przeciwnym razie domyślnym przedrostkiem kolejki dynamicznej jest AMQ. *.

**Konstruktor MQQueueManager**
Połączenie klienta nie jest obsługiwane w systemie z/OS. Próba nawiązania połączenia z opcjami klienta powoduje wystąpienie wyjątku MQException z opcjami MQCC_FAILED i MQRC_ENVIRONMENT_ERROR.

Konstruktor MQQueueManager może również zakończyć się niepowodzeniem z błędem MQRC_CHAR_CONVERSION_ERROR (jeśli nie zainicjuje konwersji między stronami kodowymi IBM-1047 i ISO8859-1 ) lub MQRC_UCS2_CONVERSION_ERROR (jeśli nie zainicjuje konwersji między stroną kodową menedżera kolejek a kodem Unicode). Jeśli działanie aplikacji zakończy się niepowodzeniem z jednym z tych kodów przyczyny, należy upewnić się, że zainstalowany jest komponent zasobów języków narodowych środowiska językowego oraz że dostępne są poprawne tabele konwersji.

Tabele konwersji dla kodu Unicode są instalowane jako część opcjonalnego składnika z/OS C/C + +. Więcej informacji na temat włączania konwersji UCS-2 zawiera publikacja *z/OS C/C++ Programming Guide*, SC09-4765.

## *Funkcje poza podstawowymi klasami produktu IBM MQ classes for Java*

Produkt IBM MQ classes for Java zawiera pewne funkcje zaprojektowane specjalnie do używania rozszerzeń interfejsu API, które nie są obsługiwane przez wszystkie menedżery kolejek. W tej kolekcji tematów opisano ich zachowanie podczas używania menedżera kolejek, który ich nie obsługuje.

*Warianty w opcji konstruktora MQQueueManager*
Niektóre konstruktory MQQueueManager zawierają opcjonalny argument będący liczbą całkowitą. Niektóre wartości tego argumentu nie są akceptowane na wszystkich platformach.

Jeśli konstruktor MQQueueManager zawiera opcjonalny argument w postaci liczby całkowitej, jest on odwzorowywany na pole opcji MQCNO interfejsu MQI i używany do przełączania między połączeniem normalnym a połączeniem krótkiej ścieżki. Ta rozszerzona forma konstruktora jest akceptowana we wszystkich środowiskach, jeśli używane są tylko opcje MQCNO_STANDARD_BINDING lub MQCNO_FASTPATH_BINDING. Wszystkie inne opcje powodują niepowodzenie konstruktora z błędem MQRC_OPTIONS_ERROR. Opcja krótkiej ścieżki CMQC.MQCNO_FASTPATH_BINDING jest uwzględniana tylko w przypadku połączenia powiązań z obsługiwanym menedżerem kolejek. W innych środowiskach jest on ignorowany.

*Ograniczenia dotyczące metody MQQueueManager.begin ()*
Ta metoda może być używana tylko dla menedżera kolejek IBM MQ w systemach AIX, Linux, and Windows w trybie powiązań. W przeciwnym razie działanie komendy kończy się niepowodzeniem z błędem MQRC_ENVIRONMENT_ERROR.

Więcej informacji na temat zawiera sekcja “Koordynacja JTA/JDBC przy użyciu IBM MQ classes for Java” na stronie 413.

*Warianty w polach opcji MQGetMessage*
Niektóre menedżery kolejek nie obsługują struktury MQGMO w wersji 2, dlatego należy ustawić niektóre pola na ich wartości domyślne.

Jeśli używany jest menedżer kolejek, który nie obsługuje struktury MQGMO w wersji 2, należy pozostawić następujące pola ustawione na wartości domyślne:

GroupStatus
SegmentStatus
Segmentacja

Ponadto pole MatchOptions obsługuje tylko wartości MQMO_MATCH_MSG_ID i MQMO_MATCH_CORREL_ID. Jeśli w tych polach zostaną umieszczone nieobsługiwane wartości, kolejne

wywołania MQDestination.get() zakończą się niepowodzeniem z błędem MQRC_GMO_ERROR. Jeśli menedżer kolejek nie obsługuje struktury MQGMO w wersji 2, te pola nie są aktualizowane po pomyślnym wykonaniu metody MQDestination.get().

*Ograniczenia dotyczące list dystrybucyjnych w programie IBM MQ classes for Java*
Nie wszystkie menedżery kolejek umożliwiają otwarcie MQDistributionList.

Do tworzenia list dystrybucyjnych używane są następujące klasy:

MQDistributionList
Element MQDistributionList
MQMessageTracker

Można tworzyć i zapełniać elementy MQDistributionLists i MQDistributionListw dowolnym środowisku, ale nie wszystkie menedżery kolejek umożliwiają otwarcie MQDistributionList. W szczególności menedżery kolejek systemu z/OS nie obsługują list dystrybucyjnych. Próba otwarcia MQDistributionList , gdy używany jest taki menedżer kolejek, powoduje błąd MQRC_OD_ERROR.

*Warianty w polach opcji MQPutMessage*
Jeśli menedżer kolejek nie obsługuje list dystrybucyjnych, niektóre pola MQPMO są traktowane inaczej.

Cztery pola w obiekcie MQPMO są renderowane jako następujące zmienne składowe klasy opcji MQPutMessage:

Liczba: knownDest
Liczba: unknownDest
Liczba: invalidDest
recordFields

Te pola są przeznaczone przede wszystkim do użycia z listami dystrybucyjnymi. Jednak menedżer kolejek, który obsługuje listy dystrybucyjne, również wypełnia pola DestCount po wykonaniu operacji MQPUT dla pojedynczej kolejki. Na przykład, jeśli kolejka jest tłumaczona na kolejkę lokalną, knownDestLiczba jest ustawiona na 1, a pozostałe dwa pola są ustawione na 0.

Jeśli menedżer kolejek nie obsługuje list dystrybucyjnych, wartości te są symulowane w następujący sposób:

- Jeśli wykonanie put () powiedzie się, parametr unknownDestLiczba jest ustawiony na 1, a pozostałe są ustawione na 0.
- Jeśli wykonanie put () nie powiedzie się, liczba invalidDestzostanie ustawiona na 1, a liczba pozostałych na 0.

Zmienna recordFields jest używana z listami dystrybucyjnymi. Wartość można zapisać w polu recordFields w dowolnym momencie, bez względu na środowisko. Jest on ignorowany, jeśli obiekt MQPutMessageOptions jest używany w kolejnej metodzie MQDestination.put() lub MQQueueManager.put (), a nie w metodzie MQDistributionList.put ().

*Ograniczenia w polach MQMD dotyczące produktu IBM MQ classes for Java*
Jeśli używany jest menedżer kolejek, który nie obsługuje segmentacji, niektóre pola MQMD związane z segmentacją komunikatów powinny mieć wartość domyślną.

Następujące pola MQMD są w dużym stopniu związane z segmentacją komunikatów:

GroupId
Numer_kolejny_komunikatu
Depozycja
MsgFlags
OriginalLength

Jeśli aplikacja ustawi dowolne z tych pól MQMD na wartości inne niż ich wartości domyślne, a następnie wykona metodę put () lub get () w menedżerze kolejek, który ich nie obsługuje, metody put () lub get () zgłosi wyjątek MQException z opcją MQRC_MD_ERROR. Pomyślna operacja put () lub get () z takim menedżerem kolejek zawsze pozostawia pola MQMD ustawione na wartości domyślne. Nie należy

wysyłać zgrupowanego lub segmentowanego komunikatu do aplikacji Java , która działa w odniesieniu do menedżera kolejek, który nie obsługuje grupowania komunikatów i segmentacji komunikatów.

Jeśli aplikacja Java próbuje pobrać komunikat z menedżera kolejek, który nie obsługuje tych pól, a komunikat fizyczny, który ma zostać pobrany, jest częścią grupy komunikatów segmentowanych (tzn. ma wartości inne niż domyślne dla pól MQMD), jest on pobierany bez błędu. Jednak pola MQMD w komunikacie MQMessage nie są aktualizowane, właściwość formatu MQMessage jest ustawiona na wartość MQFMT_MD_EXTENSION, a prawdziwe dane komunikatu są poprzedzone strukturą MQMDE, która zawiera wartości nowych pól.

### *Ograniczenia dotyczące produktu IBM MQ classes for Java w produkcie CICS Transaction Server*

W środowisku produktu CICS Transaction Server for z/OS tylko główny (pierwszy) wątek może wywołać wywołania CICS lub IBM MQ .

Należy zauważyć, że klasy IBM MQ JMS nie są obsługiwane do użytku w aplikacji CICS Java .

Z tego powodu nie jest możliwe współużytkowanie obiektów MQQueueManager lub MQQueue przez wątki w tym środowisku ani utworzenie nowego obiektu MQQueueManager w wątku potomnym.

z/OS "Różne różnice między systemem IBM MQ classes for Java na platformie z/OS i innymi platformami" na stronie 447 określa niektóre ograniczenia i warianty, które mają zastosowanie do programu IBM MQ classes for Java uruchamianego dla menedżera kolejek systemu z/OS . Ponadto w przypadku uruchamiania w systemie CICSmetody sterowania transakcjami w menedżerze MQQueueManager nie są obsługiwane. Zamiast wywoływania metody MQQueueManager.commit () lub MQQueueManager.backout (), aplikacje używają metod synchronizacji zadań JCICS , Task.commit() i Task.rollback(). Klasa Task jest dostarczana przez JCICS w pakiecie com.ibm.cics.server .

## Korzystanie z adaptera zasobów IBM MQ

Adapter zasobów umożliwia aplikacjom działającym na serwerze aplikacji dostęp do zasobów IBM MQ . Obsługuje on komunikację przychodzącą i wychodzącą.

### Co zawiera adapter zasobów

V 9.3.0 V 9.3.0 W produkcie IBM MQ 9.3.0produkt Jakarta Messaging 3.0 jest obsługiwany na potrzeby tworzenia nowych aplikacji. Produkt IBM MQ 9.3.0 nadal obsługuje produkt JMS 2.0 dla istniejących aplikacji. Oprócz adaptera zasobów, który obsługuje systemy Java EE i JMS 2.0, IBM MQ 9.3.0 udostępnia adapter zasobów, który obsługuje system Jakarta Messaging.

JM 3.0 **Adapter zasobów IBM MQ dla systemu Jakarta Messaging**

Produkt Jakarta Connectors Architecture udostępnia standardowy sposób łączenia aplikacji działających w środowisku Jakarta EE z systemem EIS (Enterprise Information System), takim jak IBM MQ lub Db2. Adapter zasobów IBM MQ dla systemu Jakarta Messaging implementuje interfejsy Jakarta Connectors 2.0.0 i zawiera IBM MQ classes for Jakarta Messaging. Umożliwia on aplikacjom Jakarta Messaging i komponentom bean sterowanym komunikatami (MDB) działającym na serwerze aplikacji uzyskiwanie dostępu do zasobów menedżera kolejek produktu IBM MQ . Adapter zasobów obsługuje zarówno domenę punkt z punktem, jak i domenę publikowania/subskrypcji.

JMS 2.0 **Adapter zasobów IBM MQ dla systemu JMS 2.0**

Java Platform, Enterprise Edition Connector Architecture (JCA) udostępnia standardowy sposób łączenia aplikacji działających w środowisku Java EE z systemem informacyjnym przedsiębiorstwa (Enterprise Information System-EIS), takim jak IBM MQ lub Db2. Adapter zasobów IBM MQ dla systemu JMS 2.0 implementuje interfejsy JCA 1.7 i zawiera IBM MQ classes for JMS. Umożliwia ona aplikacjom JMS i komponentom bean sterowanym komunikatami (MDB) działającym na serwerze aplikacji uzyskiwanie dostępu do zasobów menedżera kolejek produktu IBM MQ . Adapter zasobów obsługuje zarówno domenę punkt z punktem, jak i domenę publikowania/subskrypcji.

Adapter zasobów IBM MQ obsługuje dwa typy komunikacji między aplikacją i menedżerem kolejek:

**Komunikacja wychodząca**
Aplikacja uruchamia połączenie z menedżerem kolejek, a następnie wysyła komunikaty JMS do miejsc docelowych JMS i odbiera komunikaty JMS z miejsc docelowych JMS w sposób synchroniczny.

**Komunikacja przychodząca**
Komunikat JMS przychodzący do miejsca docelowego JMS jest dostarczany do komponentu MDB, który przetwarza komunikat asynchronicznie.

Adapter zasobów zawiera również IBM MQ classes for Java. Klasy są automatycznie dostępne dla aplikacji działających na serwerze aplikacji, na którym wdrożono adapter zasobów, i umożliwiają aplikacjom działającym na tym serwerze aplikacji używanie funkcji API IBM MQ classes for Java podczas uzyskiwania dostępu do zasobów menedżera kolejek produktu IBM MQ .

Używanie IBM MQ classes for Java w środowisku Java EE jest obsługiwane z ograniczeniami. Więcej informacji na temat tych ograniczeń zawiera sekcja “Uruchamianie aplikacji IBM MQ classes for Java w produkcie Java EE” na stronie 360.

## Która wersja adaptera zasobów ma być używana

V 9.3.0 V 9.3.0 Wersja używanego adaptera zasobów zależy od tego, czy jest on wdrażany na serwerze aplikacji, który obsługuje produkt Jakarta EE , czy Java EE:

V 9.3.0 JM 3.0 V 9.3.0 **Jakarta EE**
Od wersji IBM MQ 9.3.0obsługiwany jest produkt Jakarta Messaging 3.0 . Adapter zasobów IBM MQ dla systemu Jakarta Messaging musi być wdrożony na serwerze aplikacji, który obsługuje produkt Jakarta EE.

**Java EE 7**
Adapter zasobów IBM MQ 8.0 i nowszy obsługuje produkt JCA v1.7 i udostępnia obsługę języka JMS 2.0 . Ten adapter zasobów musi zostać wdrożony na serwerze aplikacji w wersji Java EE 7 lub nowszej (patrz sekcja “IBM MQ deklaracja obsługi adaptera zasobów” na stronie 452).

Adapter zasobów IBM MQ 8.0 lub nowszy można zainstalować na dowolnym serwerze aplikacji, który jest certyfikowany zgodnie ze specyfikacją Java Platform, Enterprise Edition 7 . Przy użyciu adaptera zasobów w wersji IBM MQ 8.0 lub nowszej aplikacja może nawiązać połączenie z menedżerem kolejek przy użyciu transportu BINDINGS lub CLIENT.

**Ważne:** Adapter zasobów w wersji IBM MQ 8.0 lub nowszej można wdrożyć tylko na serwerze aplikacji, który obsługuje produkt JMS 2.0.

## Używanie adaptera zasobów z produktem WebSphere Application Server traditional

W systemie IBM MQ 9.0adapter zasobów IBM MQ jest wstępnie zainstalowany w wersji WebSphere Application Server traditional 9.0 lub nowszej. Dlatego nie ma potrzeby instalowania nowego adaptera zasobów.

JM 3.0 WebSphere Application Server traditional obecnie nie obsługuje Jakarta EE. Patrz IBM MQ resource adapter statement of support (deklaracja obsługi adaptera zasobów).

**Uwaga:** Adapter zasobów w wersji IBM MQ 9.0 lub nowszej może łączyć się w trybie transportu CLIENT lub BINDINGS z dowolnym działającym menedżerem kolejek produktu IBM MQ .

## Używanie adaptera zasobów z produktem WebSphere Liberty

Aby nawiązać połączenie z produktem IBM MQ z serwera WebSphere Liberty, należy użyć adaptera zasobów IBM MQ . Ponieważ produkt Liberty nie zawiera adaptera zasobów IBM MQ , należy go uzyskać oddzielnie z serwisu Fix Central.

V 9.3.0 V 9.3.0 Wersja używanego adaptera zasobów zależy od tego, czy jest on wdrażany w wersji produktu Liberty obsługującej system Jakarta EE , czy Java EE.

Więcej informacji na temat pobierania i instalowania adaptera zasobów zawiera sekcja “Instalowanie adaptera zasobów w systemie Liberty” na stronie 460.

**Pojęcia pokrewne**

“Konfigurowanie adaptera zasobów na potrzeby komunikacji przychodzącej” na stronie 467
Aby skonfigurować komunikację przychodzącą, należy zdefiniować właściwości jednego lub większej liczby obiektów ActivationSpec .

“Konfigurowanie adaptera zasobów na potrzeby komunikacji wychodzącej” na stronie 487
Aby skonfigurować komunikację wychodzącą, należy zdefiniować właściwości obiektu ConnectionFactory i administrowanego obiektu docelowego.

“Korzystanie z IBM MQ classes for JMS/Jakarta Messaging” na stronie 84
IBM MQ classes for JMS i IBM MQ classes for Jakarta Messaging są dostawcami przesyłania komunikatów produktu Java dostarczanymi wraz z produktem IBM MQ. Oprócz implementowania interfejsów zdefiniowanych w specyfikacjach JMS i Jakarta Messaging dostawcy przesyłania komunikatów dodają dwa zestawy rozszerzeń do interfejsu API przesyłania komunikatów produktu Java .

“użycieIBM MQ classes for Java” na stronie 357
Użyj IBM MQ w środowisku Java . IBM MQ classes for Java Zezwalaj aplikacji Java na nawiązywanie połączenia z produktem IBM MQ jako klientem IBM MQ lub nawiązywanie połączenia bezpośrednio z menedżerem kolejek produktu IBM MQ .

**Odsyłacze pokrewne**

Konfigurowanie serwera aplikacji pod kątem używania najnowszego poziomu konserwacyjnego adaptera zasobów

Określanie problemu z adapterem zasobów IBM MQ

**WebSphere Application Server tematy**

Konserwowanie adaptera zasobów IBM MQ

Wdrażanie aplikacji JMS na serwerze Liberty w celu użycia dostawcy przesyłania komunikatów produktu IBM MQ

## IBM MQ deklaracja obsługi adaptera zasobów

Produkt Adapter zasobów IBM MQ , którego należy użyć do komunikacji między aplikacją i menedżerem kolejek, zależy od tego, czy używany jest interfejs API języka Jakarta Messaging 3.0 , czy interfejs API języka JMS 2.0 .

JMS 2.0 Produkt IBM MQ 8.0 lub nowszy jest dostarczany z adapterem zasobów, który implementuje specyfikację JMS 2.0 . Można go wdrożyć tylko na serwerze aplikacji, który jest zgodny z produktem Java Platform, Enterprise Edition 7 (Java EE 7) i dlatego obsługuje produkt JMS 2.0. Lista certyfikowanych serwerów aplikacji dla Java Platform, Enterprise Edition jest dostępna w serwisie WWW Oracle.

V 9.3.0 JM 3.0 V 9.3.0 W produkcie IBM MQ 9.3.0produkt Jakarta Messaging 3.0 jest obsługiwany na potrzeby tworzenia nowych aplikacji. Produkt IBM MQ 9.3.0 nadal obsługuje produkt JMS 2.0 dla istniejących aplikacji. Oprócz adaptera zasobów, który obsługuje systemy Java EE i JMS 2.0, IBM MQ 9.3.0 udostępnia adapter zasobów, który obsługuje system Jakarta Messaging. Nie jest obsługiwane używanie zarówno interfejsu API Jakarta Messaging 3.0 , jak i interfejsu API JMS 2.0 w tej samej aplikacji. Więcej informacji zawiera sekcja Używanie produktu IBM MQ classes for JMS.

### Wdrażanie w produkcie WebSphere Liberty

WebSphere Liberty 8.5.5 Fix Pack 6 i nowsze oraz WebSphere Application Server Liberty 9.0 i nowsze wersje są certyfikowanymi serwerami aplikacji Java EE 7 , więc adapter zasobów IBM MQ 9.0 może zostać na nich wdrożony.

V 9.3.0 V 9.3.0 Aby używać adaptera zasobów IBM MQ dla systemu Jakarta Messaging z produktem Liberty, należy użyć wersji systemu Liberty , która obsługuje produkt Jakarta EE.

Produkt WebSphere Liberty ma następujące funkcje umożliwiające pracę z adapterami zasobów:

- V 9.3.0 JM 3.0 V 9.3.0 Opcja messaging-3.0 umożliwia pracę z adapterami zasobów Jakarta Messaging 3.0 .
- Składnik wmqJmsClient-2.0 umożliwia pracę z adapterami zasobów produktu JMS 2.0 .
- Składnik wmqJmsClient-1.1 umożliwiający pracę z adapterami zasobów produktu JMS 1.1 .

**Ważne:**

- V 9.3.0 JM 3.0 V 9.3.0 Adapter zasobów IBM MQ dla systemu Jakarta Messaging musi być wdrożony w wersji produktu Liberty , która obsługuje Jakarta EE. Tego adaptera zasobów nie można używać z wersjami systemu Liberty , które obsługują starszą specyfikację Java EE , a nie Jakarta EE.
- JMS 2.0 Adapter zasobów produktu IBM MQ 8.0 lub nowszego, który obsługuje produkt JMS 2.0 , musi zostać wdrożony ze składnikiem wmqJmsClient-2.0 .

## Wdrażanie w produkcie WebSphere Application Server traditional

Produkt WebSphere Application Server traditional 9.0 jest dostarczany z zainstalowanym już adapterem zasobów IBM MQ 9.0 . Dlatego nie ma potrzeby instalowania nowego adaptera zasobów. Zainstalowany adapter zasobów może nawiązać połączenie w trybie transportu CLIENT lub BINDINGS z dowolnymi menedżerami kolejek działającymi w obsługiwanej wersji produktu IBM MQ. Więcej informacji na ten temat zawiera sekcja “Połączenia z menedżerami kolejek w wersji IBM MQ 8.0 lub nowszej” na stronie 453.

**Ważne:**

- Adaptera zasobów IBM MQ 9.0 nie można wdrożyć w wersjach produktu WebSphere Application Server traditional wcześniejszych niż IBM MQ 9.0, ponieważ te wersje nie są certyfikowane przez Java EE 7 .
- JM 3.0 WebSphere Application Server traditional obecnie nie obsługuje Jakarta EE.

Więcej informacji na temat wersji adaptera zasobów dostarczanych z produktem WebSphere Application Serverzawiera nota techniczna Która wersja adaptera zasobów (Resource Adapter-RA) produktu WebSphere MQ jest dostarczana z serwerem WebSphere Application Server?

## Używanie adaptera zasobów z innymi serwerami aplikacji

W przypadku wszystkich innych serwerów aplikacji zgodnych z produktem Java EE 7 lub Jakarta EE problemy występujące po pomyślnym zakończeniu testu weryfikującego instalację adaptera zasobów IBM MQ mogą zostać zgłoszone do IBM w celu zbadania danych śledzenia produktu IBM MQ i innych informacji diagnostycznych IBM MQ . Jeśli nie można pomyślnie uruchomić narzędzia do weryfikowania instalacji (IVT) adaptera zasobów IBM MQ , wszelkie napotkane problemy mogą być spowodowane niepoprawnym wdrożeniem lub niepoprawnymi definicjami zasobów, które są specyficzne dla serwera aplikacji, a problemy muszą zostać zbadane przy użyciu dokumentacji serwera aplikacji i organizacji obsługi dla tego serwera aplikacji.

## Java Środowisko wykonawcze

Środowisko wykonawcze Java (JRE), które jest używane do uruchamiania serwera aplikacji, musi być obsługiwane przez klienta IBM MQ 9.0 lub nowszego. Więcej informacji na ten tematzawiera sekcja Wymagania systemowe produktu IBM MQ. (Wybierz wersję i system operacyjny lub komponent, który chcesz wyświetlić, a następnie kliknij odsyłacz **Java** , który jest wyświetlany na karcie **Obsługiwane oprogramowanie** ).

## Połączenia z menedżerami kolejek w wersji IBM MQ 8.0 lub nowszej

Pełny zakres funkcji produktu JMS 2.0 jest dostępny podczas nawiązywania połączenia z menedżerem kolejek w wersji IBM MQ 8.0 lub nowszej przy użyciu adaptera zasobów, który został wdrożony na certyfikowanym serwerze aplikacji Java EE 7 . Aby można było korzystać z funkcji JMS 2.0 , adapter zasobów musi nawiązać połączenie z menedżerem kolejek przy użyciu trybu normalnego dostawcy

usługi przesyłania komunikatów produktu IBM MQ . Więcej informacji na ten temat zawiera sekcja Konfigurowanie właściwości JMS **PROVIDERVERSION**.

V 9.3.0 JM 3.0 V 9.3.0 Pełny zakres funkcji produktu Jakarta Messaging 3.0 jest dostępny podczas nawiązywania połączenia z menedżerem kolejek produktu IBM MQ 9.3 przy użyciu adaptera zasobów, który został wdrożony na certyfikowanym serwerze aplikacji Jakarta EE .

## Rozszerzenia produktu MQ

Specyfikacja JMS 2.0 wprowadza zmiany w sposobie działania niektórych zachowań. Ponieważ produkt IBM MQ 8.0 lub nowszy implementuje tę specyfikację, występują zmiany w zachowaniu między produktem IBM MQ 8.0 i nowszym oraz wcześniejszymi wersjami produktu. W systemie IBM MQ 8.0 lub nowszym IBM MQ classes for JMS zawiera obsługę Java właściwości systemowej `com.ibm.mq.jms.SupportMQExtensions` , która po ustawieniu wartości TRUEpowoduje, że te wersje pliku IBM MQ przywracają zachowanie IBM WebSphere MQ 7.5 lub wcześniejsze. Wartością domyślną tej właściwości jest FALSE.

Adapter zasobów w wersji IBM MQ 9.0 lub nowszej zawiera również właściwość adaptera zasobów o nazwie `supportMQExtensions` , która ma taki sam efekt i wartość domyślną jak właściwość systemowa produktu `com.ibm.mq.jms.SupportMQExtensions` Java . Ta właściwość adaptera zasobów jest domyślnie ustawiona na wartość false w pliku `ra.xml` .

Jeśli ustawiono zarówno właściwość adaptera zasobów, jak i właściwość systemową Java , właściwość systemowa ma pierwszeństwo.

Należy zauważyć, że w adapterze zasobów, który jest już wdrożony w produkcie WebSphere Application Server traditional 9.0, ta właściwość jest automatycznie ustawiana na wartość TRUE , aby ułatwić migrację.

Więcej informacji na ten temat zawiera “Właściwość SupportMQExtensions” na stronie 338.

## Zagadnienia ogólne

**Przeplatanie sesji nie jest obsługiwane**

Niektóre serwery aplikacji udostępniają możliwość nazywaną interakcjami sesji, w której ta sama sesja JMS może być używana w wielu transakcjach, chociaż jest ona rejestrowanych tylko w jednym momencie. Adapter zasobów IBM MQ nie obsługuje tej możliwości, co może prowadzić do następujących problemów:

```
Próba umieszczenia komunikatu w kolejce produktu MQ nie powiodła się z kodem przyczyny 2072
(MQRC_SYNCPOINT_NOT_AVAILABLE).
```

Wywołania funkcji xa_close () kończą się niepowodzeniem z kodem przyczyny -3 (XAER_PROTO), a w menedżerze kolejek systemu IBM MQ , do którego uzyskiwany jest dostęp z serwera aplikacji, generowany jest proces FDC o identyfikatorze sondy AT040010 . Informacje na temat wyłączania tej możliwości zawiera dokumentacja serwera aplikacji.

**Specyfikacja Java Transaction API (JTA) sposobu odtwarzania zasobów XA na potrzeby odtwarzania transakcji XA**

W sekcji 3.4.8 specyfikacji JTA nie zdefiniowano konkretnego mechanizmu, za pomocą którego ponownie tworzone są zasoby XA w celu przeprowadzenia odtwarzania transakcyjnego XA. W związku z tym odtwarzanie zasobów XA związanych z transakcją XA należy do poszczególnych menedżerów transakcji (a zatem do serwera aplikacji). W przypadku niektórych serwerów aplikacji adapter zasobów IBM MQ 9.0 może nie implementować mechanizmów specyficznych dla serwera aplikacji, które są używane do wykonywania odtwarzania transakcyjnego XA.

**Uzgadnianie połączeń w fabryce ManagedConnection**

Serwer aplikacji może wywołać metodę matchManagedConnections na instancji fabryki ManagedConnectionudostępnionej przez adapter zasobów IBM MQ . Klasa ManagedConnection jest zwracana tylko wtedy, gdy metoda znajdzie taki, który jest zgodny z argumentami **`javax.security.auth.Subject`** i **`javax.resource.spi.ConnectionRequestInfo`** przekazanymi do metody przez serwer aplikacji.

## Ograniczenia adaptera zasobów IBM MQ

Adapter zasobów IBM MQ jest obsługiwany na wszystkich platformach IBM MQ . Jeśli jednak używany jest adapter zasobów IBM MQ , niektóre funkcje produktu IBM MQ są niedostępne lub ograniczone.

Adapter zasobów IBM MQ ma następujące ograniczenia:

- Od wersji IBM MQ 8.0adapter zasobów jest adapterem Java Platform, Enterprise Edition 7 (Java EE 7) udostępniającym funkcję JMS 2.0 . W związku z tym adapter zasobów IBM MQ 8.0 lub nowszy musi być zainstalowany na certyfikowanym serwerze aplikacji Java EE 7 lub nowszym. Może on łączyć się w trybie transportu klienta lub powiązań z dowolnym menedżerem kolejek w usłudze.
- W przypadku uruchamiania na serwerze aplikacji WebSphere Liberty stabilny plik IBM MQ classes for Java nie jest obsługiwany. W przypadku innych serwerów aplikacji nie zaleca się używania produktu IBM MQ classes for Java . Szczegółowe informacje na temat uwag IBM MQ classes for Java dotyczących produktu Java EEzawiera IBM nota techniczna Using WebSphere MQ Java Interfaces in J2EE/JEE Environments (Używanie interfejsów produktu WebSphere MQ w środowiskach J2EE/JEE).
- W przypadku uruchamiania na serwerze aplikacji WebSphere Liberty w systemie z/OSnależy użyć składnika wmqJmsClient-2.0 . Ogólna obsługa JCA nie jest możliwa w przypadku systemu z/OS.
- Adapter zasobów IBM MQ nie obsługuje programów obsługi wyjścia kanału napisanych w językach innych niż Java.
- Gdy serwer aplikacji jest uruchomiony, wartość właściwości wymaganej sslFipsmusi być równa true dla wszystkich zasobów JCA lub false dla wszystkich zasobów JCA . Jest to wymaganie nawet wtedy, gdy zasoby JCA nie są używane współbieżnie. Jeśli właściwość sslFipsRequired ma różne wartości dla różnych zasobów JCA , IBM MQ zgłasza kod przyczyny MQRC_UNSUPPORTED_CIPHER_SUITE, nawet jeśli połączenie TLS nie jest używane.
- Nie można podać więcej niż jednego magazynu kluczy dla serwera aplikacji. Jeśli połączenia są nawiązywane z więcej niż jednym menedżerem kolejek, wszystkie połączenia muszą używać tego samego magazynu kluczy. To ograniczenie nie dotyczy systemu WebSphere Application Server.
- Jeśli tabela definicji kanału klienta (CCDT) jest używana z więcej niż jedną odpowiednią definicją kanału połączenia klienckiego, w przypadku niepowodzenia adapter zasobów może wybrać inną definicję kanału, a tym samym inny menedżer kolejek z tabeli CCDT, co może spowodować problemy z odtwarzaniem transakcji. Adapter zasobów nie podejmuje żadnych działań, aby zapobiec użyciu takiej konfiguracji, a jego zadaniem jest unikanie konfiguracji, które mogą powodować problemy podczas odtwarzania transakcji.
- Funkcja ponawiania połączenia nie jest obsługiwana w przypadku połączeń wychodzących działających w kontenerze Java EE (EJB/Servlet). Ponowienie połączenia nie jest w ogóle obsługiwane dla połączeń wychodzących JMS , gdy adapter jest używany w kontekście kontenera JEE , niezależnie od konfiguracji transakcji lub użycia bez transakcji.
- Ponowne uwierzytelnianie, zgodnie z definicją w sekcji 9.1.9 specyfikacji Java EE Connector Architecture w wersji 1.7 , połączeń JMS nie jest obsługiwane. Plik `ra.xml` w ramach adpatera zasobów IBM MQ musi mieć właściwość o nazwie **`reauthentication-support`** ustawioną na wartość `false`. Próba ponownego uwierzytelnienia połączenia JMS przez serwer aplikacji powoduje, że adapter zasobów IBM MQ zgłasza wyjątek javax.resource.spi.SecurityException z kodem komunikatu MQJCA1028 .

**Zadania pokrewne**

Określenie, że w czasie wykonywania na kliencie MQI będą używane tylko CipherSpecs z certyfikatem FIPS.

**Odsyłacze pokrewne**

Standardy FIPS (Federal Information Processing Standards) dla AIX, Linux, and Windows

## WebSphere Application Server i adapter zasobów IBM MQ

Adapter zasobów IBM MQ jest używany przez aplikacje, które wykonują przesyłanie komunikatów JMS z dostawcą przesyłania komunikatów IBM MQ w produkcie WebSphere Application Server.

**Ważne:** Nie należy używać adaptera zasobów IBM MQ z systemem WebSphere Application Server 6.0 lub WebSphere Application Server 6.1.

Produkt WebSphere Application Server traditional 9.0 zawiera wersję adaptera zasobów IBM MQ 9.0 . Adaptera zasobów IBM MQ 9.0 lub nowszego nie można wdrożyć we wcześniejszych wersjach produktu WebSphere Application Server, ponieważ te wersje nie mają certyfikatu Java EE 7 .

V 9.3.0 JM 3.0 V 9.3.0 WebSphere Application Server traditional obecnie nie obsługuje Jakarta EE. Patrz IBM MQ resource adapter statement of support (deklaracja obsługi adaptera zasobów).

Aby uzyskać dostęp do zasobów menedżera kolejek produktu IBM MQ z poziomu programu WebSphere Application Serverprzy użyciu aplikacji JMS , należy użyć dostawcy przesyłania komunikatów produktu IBM MQ w produkcie WebSphere Application Server. Dostawca przesyłania komunikatów IBM MQ zawiera wersję produktu IBM MQ classes for JMS. Więcej informacji na ten temat zawiera nota techniczna Która wersja produktu WebSphere MQ Resource Adapter (RA) jest dostarczana z serwerem WebSphere Application Server?.

**Ważne:** Nie należy dołączać do aplikacji żadnych plików JAR IBM MQ classes for JMS ani IBM MQ classes for Java . Może to spowodować wystąpienie wyjątków ClassCasti może być trudne do obsługi.

## Liberty i adapter zasobów IBM MQ

Adapter zasobów IBM MQ można zainstalować w produkcie WebSphere Liberty przy użyciu opcji Liberty . Używana opcja zależy od instalowanej wersji adaptera zasobów. Można również, z zastrzeżeniem pewnych ograniczeń, zainstalować adapter zasobów przy użyciu ogólnej obsługi Java Platform, Enterprise Edition Connector Architecture (Java EE JCA).

### Ogólne ograniczenia podczas instalowania adaptera zasobów w produkcie Liberty

Następujące ograniczenia mają zastosowanie do adaptera zasobów w przypadku korzystania ze składnika wmqJmsClient-1.1 lub wmqJmsClient-2.0 , a także w przypadku korzystania z ogólnej obsługi języka JCA :

- IBM MQ classes for Java nie są obsługiwane w systemie Liberty. Nie można ich używać z funkcją przesyłania komunikatów produktu IBM MQ Liberty ani z ogólną obsługą języka JCA . Więcej informacji na ten temat zawiera sekcja Używanie interfejsów produktu WebSphere MQ Java w środowiskach J2EE/JEE.
- Adapter zasobów IBM MQ ma typ transportu BINDINGS_THEN_CLIENT. Ten typ transportu nie jest obsługiwany w ramach funkcji przesyłania komunikatów produktu IBM MQ Liberty .
- W systemach wcześniejszych niż IBM MQ 9.0składnik Advanced Message Security (AMS) nie był uwzględniany w składniku przesyłania komunikatów produktu IBM MQ Liberty . Jednak produkt AMS jest obsługiwany z adapterem zasobów w wersji IBM MQ 9.0 lub nowszej.

**Uwaga:** W systemach IBM MQ w wersjach nowszych niż IBM MQ 9.0.0.6 i IBM MQ 9.1.0.1 należy użyć składnika transportSecurity-1.0 zamiast składnika ssl-1.0 .

Aby uzyskać więcej informacji, patrz:

Włączanie komunikacji SSL w produkcie Liberty
Wartości domyślne protokołu SSL w produkcie Liberty
Zabezpieczenia transportu 1.0

### Ograniczenia dotyczące korzystania z funkcji programu Liberty

W przypadku wartości od WebSphere Liberty 8.5.5 Fix Pack 2 do WebSphere Liberty 8.5.5 Fix Pack 5 włącznie dostępny był tylko składnik wmqJmsClient-1.1 i można było użyć tylko opcji JMS 1.1 . Produkt WebSphere Liberty 8.5.5 Fix Pack 6 dodał składnik wmqJmsClient-2.0 , aby można było użyć produktu JMS 2.0 .

V 9.3.0 JM 3.0 V 9.3.0 Począwszy od wersji IBM MQ 9.3.0obsługiwany jest system Jakarta Messaging 3.0 . Aby używać adaptera zasobów IBM MQ dla systemu Jakarta Messaging

z produktem Liberty, należy użyć wersji systemu Liberty , która obsługuje produkt Jakarta EE. Należy użyć adaptera zasobów dla systemu Jakarta Messaging z opcją Liberty generic messaging-3.0 .

Opcja, której należy użyć, zależy od używanej wersji adaptera zasobów:

- Adapter zasobów produktu IBM MQ 8.0 w wersji IBM MQ 8.0.0 Fix Pack 3 lub nowszej może być używany tylko ze składnikiem wmqJmsClient-2.0 .
- Adapter zasobów IBM MQ 9.0 może być używany tylko ze składnikiem wmqJmsClient-2.0 .
- V 9.3.0 JM 3.0 V 9.3.0 Opcja messaging-3.0 umożliwia pracę z adapterami zasobów systemu Jakarta Messaging 3.0 .

## Ograniczenia dotyczące korzystania z ogólnej obsługi JCA

Jeśli używana jest ogólna obsługa JCA , obowiązują następujące ograniczenia:

- Podczas korzystania z ogólnej obsługi JCA należy określić poziom JMS . Wartości JMS 2.0 i JCA 1.7 muszą być używane tylko z adapterami zasobów IBM MQ 8.0 w wersji IBM MQ 8.0.0 Fix Pack 3 lub nowszej.
- Nie można uruchomić adaptera zasobów IBM MQ w systemie z/OS przy użyciu ogólnej obsługi języka JCA . Aby uruchomić adapter zasobów IBM MQ w systemie z/OS, należy go uruchomić ze składnikiem wmqJmsClient-1.1 lub wmqJmsClient-2.0 .
- Położenie adaptera zasobów jest określane przy użyciu następującego elementu XML:

JM 3.0
```
<resourceAdapter id="mqJms" location="${server.config.dir}/
wmq.jakarta.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

JMS 2.0
```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

**Ważne:** Wartość znacznika ID może być dowolną wartością EXCEPT dla `wmqJms`. Jeśli jako identyfikator zostanie użyty identyfikator `wmqJms` , produkt Liberty nie będzie w stanie poprawnie załadować adaptera zasobów. Jest to spowodowane tym, że `wmqJms` jest identyfikatorem używanym wewnętrznie do odwoływania się do konkretnej funkcji produktu IBM MQ. W rzeczywistości tworzony jest wyjątek NullPointer.

W poniższych przykładach przedstawiono niektóre fragmenty kodu z pliku `server.xml` podczas uruchamiania komendy JMS 2.0:

```
<!-- Enable features -->
  <featureManager>
    <feature>servlet-3.1</feature>
    <feature>jndi-1.0</feature>
    <feature>jca-1.7</feature>
    <feature>jms-2.0</feature>
  </featureManager>
```

**Wskazówka:** Należy zwrócić uwagę na użycie składników jca-1.7 i jms-2.0 oraz na brak składnika wmqJmsClient-2.0 .

```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

**Wskazówka:** Należy zwrócić uwagę na użycie parametru `mqJms` dla identyfikatora, który jest preferowany. Nie należy używać opcji `wmqJms`.

```
<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>
```

**Wskazówka:** Należy zwrócić uwagę na odwołanie classloaderProviderdo adaptera zasobów za pośrednictwem identyfikatora `mqJms`. Jest to zezwolenie na ładowanie klas specyficznych dla języka IBM MQ.

## Ograniczenia dotyczące śledzenia przy użyciu ogólnej obsługi JCA

Śledzenie i rejestrowanie nie są zintegrowane z systemem śledzenia Liberty . Zamiast tego należy włączyć śledzenie adaptera zasobów IBM MQ za pomocą właściwości systemowych Java lub pliku konfiguracyjnego IBM MQ classes for JMS , zgodnie z opisem w sekcji Śledzenie aplikacji IBM MQ classes for JMS. Szczegółowe informacje na temat ustawiania właściwości systemu Java w pliku Libertyzawiera dokumentacja Dokumentacja produktu WebSphere Liberty.

Na przykład, aby włączyć śledzenie adaptera zasobów IBM MQ w produkcie Liberty 19.0.0.9, dodaj wpis do pliku Liberty `jvm.options`:

1. Utwórz plik tekstowy o nazwie `jvm.options`.
2. Wstaw do tego pliku następujące opcje maszyny JVM, aby włączyć śledzenie (po jednej w każdym wierszu):

   ```
   -Dcom.ibm.msg.client.commonservices.trace.status=ON
   -Dcom.ibm.msg.client.commonservices.trace.outputName=C:\Trace\MQRA-WLP_%PID%.trc
   ```

3. Aby zastosować te ustawienia na pojedynczym serwerze, zapisz plik `jvm.options` w katalogu:

   ```
   ${server.config.dir}/jvm.options
   ```

   Aby zastosować te zmiany do wszystkich Liberty, zapisz plik `jvm.options` w katalogu:

   ```
   ${wlp.install.dir}/etc/jvm.options
   ```

   Będzie to miało wpływ na wszystkie maszyny JVM, które nie mają lokalnie zdefiniowanego pliku `jvm.options` .
4. Zrestartuj serwer, aby włączyć zmiany.

Powoduje to zapisanie danych śledzenia w pliku śledzenia o nazwie `MQRA-WLP_<process identifier>.trc` w katalogu `<path_to_trace_to>`.

## Pełna obsługa interfejsu XA produktu Liberty z tabelami definicji kanału klienta

Począwszy od wersji WebSphere Liberty 18.0.0.2 , w produkcie IBM MQ 9.2.0 lub nowszym, można używać grup menedżerów kolejek w tabeli definicji kanału klienta (CCDT) w połączeniu z transakcjami XA. Oznacza to, że można teraz korzystać z dystrybucji obciążenia i dostępności, udostępnianych przez grupy menedżerów kolejek, przy zachowaniu integralności transakcji.

W przypadku wystąpienia błędów połączenia z menedżerem kolejek menedżer kolejek musi ponownie stać się dostępny, aby można było rozstrzygnąć transakcję. Odtwarzanie transakcji jest zarządzane przez program Libertyi może być konieczne skonfigurowanie menedżera transakcji w taki sposób, aby dozwolony był odpowiedni okres czasu na ponowne udostępnienie menedżerów kolejek. Więcej informacji na ten temat zawiera sekcja Menedżer transakcji (transakcja) w dokumentacji produktu WebSphere Liberty .

Jest to opcja po stronie klienta, która wymaga adaptera zasobów produktu IBM MQ 9.2.0 lub nowszego, a nie menedżera kolejek w wersji IBM MQ 9.2.0 lub nowszej.

# Instalowanie adaptera zasobów IBM MQ

Adapter zasobów IBM MQ jest dostarczany jako plik archiwum zasobów (RAR). Zainstaluj plik RAR na serwerze aplikacji. Konieczne może być dodanie katalogów do ścieżki systemowej.

## O tym zadaniu

Adapter zasobów IBM MQ jest dostarczany w postaci pliku archiwum zasobów (RAR):

- V 9.3.0 JM 3.0 V 9.3.0 W systemie Jakarta Messaging 3.0plik ten nosi nazwę `wmq.jakarta.jmsra.rar`. Plik RAR zawiera implementację języka IBM MQ classes for Jakarta Messaging i IBM MQ interfejsów Jakarta Connectors Architecture (JCA).
- JMS 2.0 W systemie JMS 2.0ten plik ma nazwę `wmq.jmsra.rar`. Plik RAR zawiera implementację języka IBM MQ classes for JMS i języka IBM MQ interfejsów Java EE Connector Architecture (JCA).

Jeśli adapter zasobów jest instalowany w ramach instalacji produktu IBM MQ , plik RAR jest instalowany razem z plikiem IBM MQ classes for JMS w katalogu Tabela 61 na stronie 459.

*Tabela 61. Katalog IBM MQ zawierający plik RAR dla każdej platformy*

| **Platforma** | **Katalog** |
|---|---|
| AIX and Linux | *MQ_INSTALLATION_PATH*/java/lib/jca |
| IBM i | /QIBM/ProdData/mqm/java/lib/jca |
| Windows | *MQ_INSTALLATION_PATH*\java\lib\jca |
| z/OS | *MQ_INSTALLATION_PATH*/java/lib/jca |

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Adapter zasobów IBM MQ musi być używany do nawiązywania połączenia z produktem IBM MQ z serwera aplikacji. W zależności od używanego serwera aplikacji adapter zasobów może być wstępnie zainstalowany lub może być konieczne jego samodzielne zainstalowanie.

*Tabela 62. Instalowanie adaptera zasobów na serwerze aplikacji*

| **Serwer aplikacji** | **Wstępnie zainstalowane lub konieczne jest zainstalowanie?** |
|---|---|
| WebSphere Application Server traditional 9.0 | Adapter zasobów IBM MQ 9.0 jest wstępnie zainstalowany w katalogu WebSphere Application Server traditional 9.0. Dlatego nie ma potrzeby instalowania nowego adaptera zasobów w produkcie WebSphere Application Server traditional 9.0. |
| WebSphere Liberty | WebSphere Liberty nie zawiera adaptera zasobów IBM MQ , dlatego należy go uzyskać oddzielnie z serwisu Fix Central. |
| Inny serwer aplikacji Java EE lub Jakarta EE | Adapter zasobów należy uzyskać niezależnie od produktu Fix Central, tak jak w przypadku produktu WebSphere Liberty. |

## Procedura

- Jeśli nawiązywane jest połączenie z produktem IBM MQ z serwisu WebSphere Libertylub innego serwera aplikacji Java EE lub Jakarta EE , pobierz i zainstaluj adapter zasobów IBM MQ zgodnie z opisem w sekcji “Instalowanie adaptera zasobów w systemie Liberty” na stronie 460.
- Linux AIX
  W przypadku połączeń powiązań w systemach AIX and Linux należy upewnić się, że katalog zawierający biblioteki JNI ( Java Native Interface) znajduje się w ścieżce systemowej.

Położenie tego katalogu, który zawiera również biblioteki IBM MQ classes for JMS , można znaleźć w sekcji “Konfigurowanie bibliotek JNI ( Java Native Interface)” na stronie 100.

Windows W systemie Windowsten katalog jest automatycznie dodawany do ścieżki systemowej podczas instalowania produktu IBM MQ classes for JMS.

**Wskazówka:** Zamiast ustawiania ścieżki systemowej adapter zasobów IBM MQ ma właściwość o nazwie nativeLibrary, która może być użyta do określenia położenia biblioteki JNI. Na przykład w pliku WebSphere Liberty zostanie on skonfigurowany w sposób przedstawiony w poniższym przykładzie:

```
<wmqJmsClient nativeLibraryPath="/opt/mqm/java/lib64"/>
```

Transakcje są obsługiwane zarówno w trybie klienta, jak i w trybie powiązań.

### *Instalowanie adaptera zasobów w systemie Liberty*

Aby połączyć się z serwerem IBM MQ z serwera WebSphere Liberty, innego serwera Java EE lub serwera aplikacji Jakarta EE , należy użyć adaptera zasobów IBM MQ . Ponieważ produkt Liberty nie zawiera adaptera zasobów IBM MQ , należy go uzyskać oddzielnie z serwisu Fix Central.

## Zanim rozpoczniesz

**Uwaga:** Informacje zawarte w tym temacie nie dotyczą systemu WebSphere Application Server traditional 9.0. Adapter zasobów IBM MQ 9.0 jest wstępnie zainstalowany w katalogu WebSphere Application Server traditional 9.0. Dlatego w tym przypadku nie ma potrzeby instalowania nowego adaptera zasobów.

Przed rozpoczęciem tego zadania należy się upewnić, że na komputerze jest zainstalowane środowisko Java runtime environment (JRE) i że środowisko JRE zostało dodane do ścieżki systemowej.

Instalator Java , który jest używany w tym procesie instalacji, nie wymaga uruchomienia przez użytkownika root ani przez żadnego konkretnego użytkownika. Jedynym wymaganiem jest, aby użytkownik, dla którego jest on uruchamiany, miał dostęp do zapisu w katalogu, w którym mają być zapisywane pliki.

W przypadku produktu Liberty w wersjach do WebSphere Liberty 8.5.5 Fix Pack 1, jeśli komponent EJB jest wdrażany wyłącznie przy użyciu konfiguracji w produkcie `ejb-jar.xml`, wersja produktu WebSphere Application Server , z której korzysta profil Liberty , musi mieć zastosowaną poprawkę APAR PM89890 . Ta metoda konfiguracji jest używana dla programu weryfikującego instalację adaptera zasobów, dlatego ta poprawka APAR jest wymagana do uruchomienia programu IVT.

V 9.3.0 JM 3.0 V 9.3.0 Począwszy od wersji IBM MQ 9.3.0obsługiwany jest system Jakarta Messaging 3.0 . Aby używać adaptera zasobów IBM MQ dla systemu Jakarta Messaging z produktem Liberty, należy użyć wersji systemu Liberty , która obsługuje produkt Jakarta EE. Na przykład można użyć Liberty ogólnej funkcji messaging-3.0 .

## O tym zadaniu

Plik JAR adaptera zasobów, który można pobrać z serwisu Fix Central , jest plikiem wykonywalnym. Po uruchomieniu tego pliku wykonywalnego wyświetlana jest umowa licencyjna IBM MQ , która musi zostać zaakceptowana. Zostanie wyświetlone zapytanie o katalog, w którym ma zostać zainstalowany adapter zasobów IBM MQ . Plik RAR adaptera zasobów i program testu sprawdzającego instalację (IVT) są następnie instalowane w tym katalogu. Można zaakceptować wartość domyślną lub określić inny katalog, który może być katalogiem adapterów zasobów serwera aplikacji lub dowolnym innym katalogiem w systemie. Katalog jest tworzony w ramach instalacji, jeśli nie istnieje.

Przed IBM MQ 9.0nazwa pliku, który ma zostać pobrany, miała format *V.R.M.F*`-WS-MQ-Java-InstallRA.jar`, na przykład `8.0.0.6-WS-MQ-Java-InstallRA.jar`. W systemie IBM MQ 9.0nazwa pliku ma format *V.R.M.F*`-IBM-MQ-Java-InstallRA.jar`, na przykład `9.0.0.0-IBM-MQ-Java-InstallRA.jar`.

Po pobraniu i zainstalowaniu adaptera zasobów można go skonfigurować w produkcie WebSphere Liberty.

## Procedura

1. Pobierz adapter zasobów IBM MQ z serwisu Fix Central.
   a) Kliknij następujący odsyłacz: IBM MQ Resource Adapter(Adapter zasobów).
   b) Znajdź adapter zasobów dla używanej wersji produktu IBM MQ na wyświetlonej liście dostępnych poprawek.
      Na przykład:

      ```
      release level: 9.1.4.0-IBM-MQ-Java-InstallRA
      Continuous Delivery Release: 9.1.4 IBM MQ Resource Adapter for use with Application
      Servers
      ```

      Następnie kliknij nazwę pliku adaptera zasobów i postępuj zgodnie z procedurą pobierania.
2. Uruchom instalację, wprowadzając następującą komendę z katalogu, do którego pobrano plik.
   W systemie IBM MQ 9.0komenda ma następujący format:

   ```
   java -jar V.R.M.F-IBM-MQ-Java-InstallRA.jar
   ```

   gdzie *V.R.M.F* jest numerem wersji, wydania, modyfikacji i pakietu poprawek, a *V.R.M.F*-IBM-MQ-Java-InstallRA.jar jest nazwą pliku, który został pobrany z serwisu Fix Central.
   Na przykład, aby zainstalować adapter zasobów IBM MQ dla wersji IBM MQ 9.1.4 , należy użyć następującej komendy:

   ```
   java -jar 9.1.4.0-IBM-MQ-Java-InstallRA.jar
   ```

   **Uwaga:** Aby przeprowadzić tę instalację, na komputerze musi być zainstalowane środowisko JRE i dodane do ścieżki systemowej.

   Po wprowadzeniu komendy wyświetlane są następujące informacje:

   ```
   Przed użyciem, wyodrębnieniem lub zainstalowaniem produktu IBM MQ 9.1należy zaakceptować
   1. IBM International License Agreement for Evaluation of
   Programy 2. IBM International Program License Agreement i dodatkowe
   informacje licencyjne. Przeczytaj uważnie poniższe umowy licencyjne.

   Umowę licencyjną można wyświetlić oddzielnie za pomocą
   --viewLicenseopcja umowy licencyjnej.
   Naciśnij klawisz Enter, aby wyświetlić teraz warunki licencji, lub 'x', aby pominąć.
   ```

3. Przejrzyj i zaakceptuj warunki licencji:
   a) Aby wyświetlić licencję, naciśnij klawisz Enter.
      Alternatywnie naciśnięcie klawisza x powoduje pominięcie wyświetlania licencji.
      Po wyświetleniu licencji lub natychmiast po wybraniu opcji x zostanie wyświetlony następujący komunikat informujący o możliwości wyświetlenia dodatkowych warunków licencji:

      ```
      Dodatkowe informacje licencyjne można wyświetlać oddzielnie przy użyciu
      --viewLicenseOpcja informacji.
      Naciśnij klawisz Enter, aby wyświetlić teraz dodatkowe informacje licencyjne, lub 'x', aby
      pominąć.
      ```

   b) Aby wyświetlić dodatkowe warunki licencji, naciśnij klawisz Enter.
      Alternatywnie naciśnięcie klawisza x powoduje pominięcie wyświetlania dodatkowych warunków licencji.
      Po wyświetleniu dodatkowych warunków licencji lub natychmiast po wybraniu opcji x zostanie wyświetlony następujący komunikat z prośbą o zaakceptowanie umowy licencyjnej:

      ```
      Wybierając opcję "Zgadzam się" poniżej, zgadzasz się na warunki
      Umowa licencyjna i warunki inne niżIBM (jeśli mają zastosowanie). Jeśli ten parametr nie
      zostanie podany
      zgadzam się, wybierz opcję "Nie zgadzam się".
      ```

```
Wybierz opcję [ 1] Zgadzam się lub [ 2] Nie zgadzam się:
```

c) Aby zaakceptować umowę licencyjną i kontynuować wybieranie katalogu instalacyjnego, wybierz 1.

Jeśli zostanie wybrana wartość 2, instalacja zostanie natychmiast zakończona.

Jeśli wybrano 1, zostanie wyświetlony następujący komunikat z prośbą o wybranie docelowego katalogu instalacyjnego:

```
Podaj katalog dla plików produktu lub pozostaw to pole puste, aby zaakceptować wartość
domyślną.
Domyślnym katalogiem docelowym jest H: \Liberty\WMQ
Katalog docelowy dla plików produktu?
```

4. Podaj katalog instalacyjny adaptera zasobów:

   - Aby zainstalować adapter zasobów w położeniu domyślnym, naciśnij klawisz Enter bez określania wartości.
   - Aby zainstalować adapter zasobów w innym położeniu niż domyślne, podaj nazwę katalogu, w którym ma zostać zainstalowany adapter zasobów, a następnie naciśnij klawisz Enter.

   Po zainstalowaniu plików w wybranym położeniu zostanie wyświetlony komunikat z potwierdzeniem, jak pokazano w poniższym przykładzie:

   ```
   Wyodrębnianie plików do H: \Liberty\WMQ \wmq
   Successfully extracted all product files.
   ```

   Podczas instalacji w wybranym katalogu instalacyjnym zostanie utworzony nowy katalog o nazwie `wmq` , a w katalogu `wmq` zostaną zainstalowane następujące pliki:

   - Program testowy weryfikacji instalacji, `wmq.jakarta.jmsra.ivt` (Jakarta Messaging 3.0) lub `wmq.jmsra.ivt` (JMS 2.0).
   - Plik IBM MQ RAR,`wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0 lub `wmq.jmsra.rar` (JMS 2.0).

5. JMS 2.0

   Opcjonalne: Skonfiguruj adapter zasobów Java EE 7 (JMS 2.0) w pliku WebSphere Liberty Profile.

   Poniżej przedstawiono kroki, które należy wykonać w celu skonfigurowania adaptera zasobów w produkcie Liberty . Więcej informacji na ten temat zawiera dokumentacja produktuWebSphere Application Server.

   a) Dodaj składnik wmqJmsClient-2.0 do pliku `server.xml` , aby umożliwić pracę z adapterem zasobów IBM MQ .

   Więcej informacji na ten temat zawiera “Która wersja adaptera zasobów ma być używana” na stronie 451.

   b) Dodaj odwołanie do zainstalowanego pliku `wmq.jmsra.rar` (JMS 2.0).

   Przykładowa konfiguracja do obsługi serwletów i komponentów MDB w produkcie JNDI może wyglądać następująco:

   ```
    <featureManager>
       <feature>wmqJmsClient-2.0</feature>
       <feature>servlet-3.0</feature>
       <feature>jmsMdb-3.1</feature>
       <feature>jndi-1.0</feature>
       </featureManager>

   <variable name="wmqJmsClient.rar.location"
         value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
   ```

6. JM 3.0

   Opcjonalne: Skonfiguruj adapter zasobów Jakarta EE 9 (Jakarta Messaging 3.0) w pliku WebSphere Liberty Profile.

   Poniżej przedstawiono kroki, które należy wykonać w celu skonfigurowania adaptera zasobów w produkcie Liberty . Więcej informacji na ten temat zawiera dokumentacja produktuWebSphere Application Server.

a) Dodaj składnik wmqJmsClient-3.0 do pliku `server.xml` , aby umożliwić pracę z adapterem zasobów IBM MQ .

   Więcej informacji na ten temat zawiera “Która wersja adaptera zasobów ma być używana” na stronie 451.

b) Dodaj odwołanie do zainstalowanego pliku `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0).

Przykładowa konfiguracja do obsługi serwletów i komponentów MDB w produkcie JNDI może wyglądać następująco:

```
 <featureManager>
    <feature>wmqJmsClient-3.0</feature>
    <feature>servlet-3.0</feature>
    <feature>jmsMdb-3.1</feature>
    <feature>jndi-1.0</feature>
    </featureManager>

<variable name="wmqJmsClient.rar.location"
      value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
```

**Uwaga:** Jeśli używany jest produkt Open Liberty, a nie produkt WebSphere Liberty Profile, konieczne będzie użycie ogólnej funkcji obsługi adaptera zasobówmessagingClient-3.0zamiastwmqJmsClient-3.0i innych aspektów konfiguracji. Więcej informacji na ten temat zawiera dokumentacja produktu Open Liberty .

## Konfigurowanie adaptera zasobów IBM MQ

Aby skonfigurować adapter zasobów IBM MQ , należy zdefiniować różne zasoby Java Platform, Enterprise Edition Connector Architecture (JCA) i opcjonalnie właściwości systemowe. Należy również skonfigurować adapter zasobów w celu uruchomienia programu testu sprawdzającego instalację (installation verification test-IVT). Jest to ważne, ponieważ usługa systemu IBM może wymagać uruchomienia tego programu w celu wskazania, że serwer aplikacji inny niżIBM został poprawnie skonfigurowany.

### Zanim rozpoczniesz

W tej czynności przyjęto założenie, że użytkownik jest już zaznajomiony z JMS i IBM MQ classes for JMS. Wiele właściwości używanych do konfigurowania adaptera zasobów IBM MQ jest równoważnych właściwościom obiektów IBM MQ classes for JMS i mają taką samą funkcję.

### O tym zadaniu

Każdy serwer aplikacji udostępnia własny zestaw interfejsów administracyjnych. Niektóre serwery aplikacji udostępniają graficzne interfejsy użytkownika służące do definiowania zasobów JCA , ale inne wymagają, aby administrator zapisał plany wdrażania XML. Dlatego też informacje na temat konfigurowania adaptera zasobów IBM MQ dla każdego serwera aplikacji wykraczają poza zakres niniejszej dokumentacji.

Dlatego poniższe kroki koncentrują się tylko na tym, co należy skonfigurować. Informacje na temat konfigurowania adaptera zasobów JCA można znaleźć w dokumentacji dostarczonej z serwerem aplikacji.

### Procedura

Zdefiniuj zasoby JCA w następujących kategoriach:

- Zdefiniuj właściwości obiektu ResourceAdapter .

  Te właściwości, które reprezentują właściwości globalne adaptera zasobów, takie jak poziom śledzenia diagnostycznego, są opisane w sekcji “Konfiguracja właściwości obiektu ResourceAdapter” na stronie 465.

- Zdefiniuj właściwości obiektu ActivationSpec .

  Te właściwości określają sposób aktywowania komponentu MDB na potrzeby komunikacji przychodzącej. Więcej informacji na ten temat zawiera “Konfigurowanie adaptera zasobów na potrzeby komunikacji przychodzącej” na stronie 467.

- Zdefiniuj właściwości obiektu ConnectionFactory .

  Serwer aplikacji używa tych właściwości do utworzenia obiektu JMS ConnectionFactory na potrzeby komunikacji wychodzącej. Więcej informacji na ten temat zawiera sekcja "Konfigurowanie adaptera zasobów na potrzeby komunikacji wychodzącej" na stronie 487.
- Zdefiniuj właściwości administrowanego obiektu docelowego.

  Serwer aplikacji używa tych właściwości do utworzenia obiektu kolejki JMS lub obiektu tematu JMS dla komunikacji wychodzącej. Więcej informacji na ten tematzawiera sekcja "Konfigurowanie adaptera zasobów na potrzeby komunikacji wychodzącej" na stronie 487.
- Opcjonalne: Zdefiniuj plan wdrożenia dla adaptera zasobów.

  Plik RAR adaptera zasobów IBM MQ zawiera plik o nazwie `META-INF/ra.xml`, który zawiera deskryptor wdrażania dla adaptera zasobów. Ten deskryptor wdrażania jest definiowany przez schemat XML w pliku https://xmlns.jcp.org/xml/ns/javaee/connector_1_7.xsd i zawiera informacje o adapterze zasobów i udostępnianych przez niego usługach. Serwer aplikacji może również wymagać planu wdrożenia dla adaptera zasobów. Ten plan wdrożenia jest specyficzny dla serwera aplikacji.

Określ wymagane właściwości systemowe maszyny JVM:

- Jeśli używany jest protokół TLS (Transport Layer Security), określ położenia pliku kluczy i pliku zaufanych certyfikatów jako właściwości systemowe maszyny JVM, tak jak w poniższym przykładzie:

  ```
  java ... -Djavax.net.ssl.keyStore=
  key_store_location
           -Djavax.net.ssl.trustStore=trust_store_location
           -Djavax.net.ssl.keyStorePassword=key_store_password
  ```

  Te właściwości nie mogą być właściwościami obiektu ActivationSpec lub ConnectionFactory i nie można określić więcej niż jednego magazynu kluczy dla serwera aplikacji. Właściwości mają zastosowanie do całej maszyny JVM i mogą mieć wpływ na serwer aplikacji, jeśli inne aplikacje działające na serwerze aplikacji korzystają z połączeń TLS. Serwer aplikacji może również zresetować te właściwości do różnych wartości. Więcej informacji na temat używania protokołu TLS z produktem IBM MQ classes for JMSzawiera sekcja "Używanie protokołu TLS z produktem IBM MQ classes for JMS" na stronie 262.
- Opcjonalne: Jeśli jest to wymagane, skonfiguruj adapter zasobów w taki sposób, aby komunikaty ostrzegawcze były rejestrowane w dzienniku wyjścia standardowego serwera aplikacji.

  Dzienniki adaptera zasobów, ostrzeżenia i komunikaty o błędach korzystają z tego samego mechanizmu, co IBM MQ classes for JMS. Więcej informacji na ten temat zawiera sekcja Rejestrowanie błędów dla produktu IBM MQ classes for JMS. Oznacza to, że domyślnie komunikaty są kierowane do pliku o nazwie `mqjms.log`. Aby skonfigurować adapter zasobów w celu dodatkowego rejestrowania komunikatów ostrzegawczych w dzienniku wyjścia standardowego serwera aplikacji, należy ustawić następującą właściwość systemową maszyny JVM dla serwera aplikacji:

  ```
  -Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
  ```

  Jest to ta sama właściwość, która jest używana do sterowania śledzeniem dla IBM MQ classes for JMS. Podobnie jak w przypadku IBM MQ classes for JMS, można użyć właściwości systemowej wskazującej plik `jms.config` (patrz sekcja "Plik konfiguracyjny IBM MQ classes for JMS/Jakarta Messaging" na stronie 102). Informacje na temat ustawiania właściwości systemowej maszyny JVM zawiera dokumentacja serwera aplikacji.

Skonfiguruj adapter zasobów, aby uruchomić test weryfikujący instalację

- Skonfiguruj adapter zasobów, aby uruchomić program testu sprawdzającego instalację (installation verification test-IVT) dostarczony z adapterem zasobów IBM MQ .

  Informacje na temat tego, co należy skonfigurować w celu uruchomienia programu IVT, zawiera sekcja "Weryfikowanie instalacji adaptera zasobów" na stronie 509.

  Jest to ważne, ponieważ usługa systemu IBM może wymagać uruchomienia tego programu w celu wskazania, że serwer aplikacji inny niżIBM został poprawnie skonfigurowany.

  **Ważne:** Przed uruchomieniem programu należy skonfigurować adapter zasobów.

### *Konfiguracja właściwości obiektu ResourceAdapter*

Obiekt ResourceAdapter hermetyzuje właściwości globalne adaptera zasobów IBM MQ , takie jak poziom śledzenia diagnostycznego. Aby zdefiniować te właściwości, należy użyć narzędzi adaptera zasobów zgodnie z opisem w dokumentacji dostarczonej z serwerem aplikacji.

Obiekt ResourceAdapter ma dwa zestawy właściwości:

- Właściwości powiązane ze śledzeniem diagnostycznym
- Właściwości powiązane z pulą połączeń zarządzaną przez adapter zasobów

Sposób definiowania tych właściwości zależy od interfejsów administracyjnych udostępnianych przez serwer aplikacji. Jeśli używana jest baza danych WebSphere Application Server traditional, należy zapoznać się z sekcją “WebSphere Application Server traditional Konfiguracja” na stronie 466 lub sekcją “WebSphere Liberty Konfiguracja” na stronie 467, jeśli używana jest baza danych WebSphere Liberty. W przypadku innych serwerów aplikacji należy zapoznać się z dokumentacją danego serwera aplikacji.

Więcej informacji na temat definiowania właściwości powiązanych ze śledzeniem diagnostycznym zawiera sekcja Śledzenie adaptera zasobów IBM MQ .

Adapter zasobów zarządza wewnętrzną pulą połączeń JMS używaną do dostarczania komunikatów do komponentów MDB. Tabela 63 na stronie 465 zawiera listę właściwości obiektu ResourceAdapter powiązanych z pulą połączeń.

*Tabela 63. Właściwości obiektu ResourceAdapter powiązane z pulą połączeń*

| Nazwa właściwości | Typ | Wartość domyślna | Opis |
|---|---|---|---|
| maxConnections | Łańcuch | 50 | Maksymalna liczba połączeń z menedżerem kolejek produktu IBM MQ i maksymalna liczba wdrożonych komponentów MDB. |
| connectionConcurrency | Łańcuch | 1 | Maksymalna liczba komponentów MDB, które współużytkują połączenie JMS . Współużytkowanie połączeń nie jest możliwe i ta właściwość zawsze ma wartość 1. |
| Liczba ponowień połączenia reconnectionRetry | Łańcuch | 5 | Maksymalna liczba prób podejmowanych przez adapter zasobów w celu ponownego nawiązania połączenia z menedżerem kolejek produktu IBM MQ w przypadku niepowodzenia połączenia. |
| Odstęp czasu między ponownymi próbami połączenia reconnectionRetry | Łańcuch | 300 000 | Czas (w milisekundach), przez który adapter zasobów oczekuje przed podjęciem próby ponownego nawiązania połączenia z menedżerem kolejek produktu IBM MQ . |
| Liczba startupRetry | Łańcuch | 0 | Domyślna liczba prób nawiązania połączenia z komponentem MDB podczas uruchamiania, jeśli menedżer kolejek nie jest uruchomiony podczas uruchamiania serwera aplikacji. |
| Odstęp czasu startupRetry | Łańcuch | 30 000 | Domyślny czas uśpienia między próbami nawiązania połączenia podczas uruchamiania (w milisekundach). |
| supportMQExtensions | Łańcuch | Fałsz | Przywraca zachowanie programu IBM MQ JMS do zachowania sprzed operacjiJMS 2.0 . Więcej informacji na ten temat zawiera sekcja “Właściwość SupportMQExtensions” na stronie 338. |

| *Tabela 63. Właściwości obiektu ResourceAdapter powiązane z pulą połączeń (kontynuacja)* | | | |
|---|---|---|---|
| **Nazwa właściwości** | **Typ** | **Wartość domyślna** | **Opis** |
| nativeLibraryŚcieżka | Łańcuch | <puste> | Ścieżka, która ma być używana do ładowania biblioteki JNI produktu IBM MQ w celu umożliwienia połączeń w trybie powiązań.<br>Windows W systemie Windows ścieżka systemowa musi również zawierać położenie zgodnej instalacji produktu IBM MQ . |

Gdy komponent MDB jest wdrażany na serwerze aplikacji, tworzone jest nowe połączenie JMS i uruchamiana jest konwersacja z menedżerem kolejek, pod warunkiem, że nie została przekroczona maksymalna liczba połączeń określona we właściwości maxConnection . Maksymalna liczba komponentów MDB jest zatem równa maksymalnej liczbie połączeń. Jeśli liczba wdrożonych komponentów MDB osiągnie tę wartość maksymalną, każda próba wdrożenia innego komponentu MDB zakończy się niepowodzeniem. Jeśli komponent MDB jest zatrzymany, jego połączenie może być używane przez inny komponent MDB.

Jeśli ma zostać wdrożonych wiele komponentów MDB, należy zwiększyć wartość właściwości maxConnections .

Właściwości odstępu czasu reconnectionRetryi reconnectionRetryzarządzają zachowaniem adaptera zasobów w przypadku niepowodzenia połączenia z menedżerem kolejek produktu IBM MQ (na przykład z powodu awarii sieci). Jeśli połączenie nie powiedzie się, adapter zasobów zawiesza dostarczanie komunikatów do wszystkich komponentów MDB dostarczanych przez to połączenie na okres określony przez właściwość reconnectionRetry. Następnie adapter zasobów podejmuje próbę ponownego nawiązania połączenia z menedżerem kolejek. Jeśli próba nie powiedzie się, adapter zasobów będzie podejmował dalsze próby ponownego nawiązania połączenia w odstępach czasu określonych przez właściwość przedziału czasu reconnectionRetry, aż do osiągnięcia limitu określonego przez właściwość licznika reconnectionRetry. Jeśli wszystkie próby zakończą się niepowodzeniem, dostarczanie zostanie zatrzymane na stałe do momentu ręcznego zrestartowania komponentów MDB.

Ogólnie rzecz biorąc, obiekt ResourceAdapter nie wymaga administrowania. Aby na przykład włączyć śledzenie diagnostyczne w systemach AIX and Linux , można ustawić następujące właściwości:

```
traceEnabled:    true
traceLevel:      10
```

Te właściwości nie mają wpływu, jeśli adapter zasobów nie został uruchomiony, co ma miejsce na przykład wtedy, gdy aplikacje używające zasobów IBM MQ działają tylko w kontenerze klienta. W takiej sytuacji można ustawić właściwości śledzenia diagnostycznego jako właściwości systemowe wirtualnej maszyny języka Java (JVM). Właściwości można ustawić za pomocą opcji -D komendy **java** , jak w poniższym przykładzie:

```
java ... -DtraceEnabled=true -DtraceLevel=6
```

Nie trzeba definiować wszystkich właściwości obiektu ResourceAdapter . Wszystkie nieokreślone właściwości przyjmują wartości domyślne. W środowisku zarządzanym lepiej nie mieszać dwóch sposobów określania właściwości. W przypadku ich mieszania właściwości systemowe maszyny JVM mają pierwszeństwo przed właściwościami obiektu ResourceAdapter .

## WebSphere Application Server traditional Konfiguracja

Te same właściwości są dostępne dla adaptera zasobów w pliku WebSphere Application Server traditional, ale należy je ustawić na panelu właściwości adaptera zasobów (patrz sekcja Ustawienia dostawcy JMS w dokumentacji produktu WebSphere Application Server traditional ). Śledzenie jest

sterowane przez sekcję diagnostyczną konfiguracji WebSphere Application Server traditional . Więcej informacji na ten temat zawiera sekcja Praca z dostawcami diagnostycznymi w dokumentacji produktu WebSphere Application Server traditional .

### WebSphere Liberty Konfiguracja

Adapter zasobów jest konfigurowany przy użyciu elementów XML w pliku `server.xml` , jak pokazano w poniższym przykładzie:

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"/>
```

Śledzenie można włączyć, dodając następujący element XML:

```
<logging traceSpecification="JMSApi=all:WAS.j2c=all:"/>
```

#### *Konfigurowanie adaptera zasobów na potrzeby komunikacji przychodzącej*

Aby skonfigurować komunikację przychodzącą, należy zdefiniować właściwości jednego lub większej liczby obiektów ActivationSpec .

Właściwości obiektu ActivationSpec określają sposób, w jaki komponent bean sterowany komunikatami (MDB) odbiera komunikaty JMS z kolejki IBM MQ . Zachowanie transakcyjne komponentu MDB jest zdefiniowane w jego deskryptorze wdrażania.

Obiekt ActivationSpec ma dwa zestawy właściwości:

- Właściwości używane do tworzenia połączenia produktu JMS z menedżerem kolejek produktu IBM MQ
- Właściwości, które są używane do tworzenia konsumenta połączenia JMS dostarczającego komunikaty asynchronicznie po ich odebraniu w określonej kolejce.

Sposób definiowania właściwości obiektu ActivationSpec zależy od interfejsów administracyjnych udostępnianych przez serwer aplikacji.

### Właściwości używane do tworzenia połączenia JMS z menedżerem kolejek produktu IBM MQ

Wszystkie właściwości w pliku Tabela 64 na stronie 468 są opcjonalne.

*Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| applicationName | Łańcuch | • Nazwa klasy wywołującej, jeśli jest dostępna, nie może być dłuższa niż 28 znaków. Jeśli nie jest dostępny, zostanie użyty łańcuch `WebSphere MQ Client for Java`. | Nazwa, pod którą aplikacja jest rejestrowana w menedżerze kolejek. Ta nazwa aplikacji jest wyświetlana w komendzie **DISPLAY CONN MQSC/PCF** (gdzie pole jest nazywane **APPLTAG** ). lub na ekranie IBM MQ Explorer **Połączenia aplikacji** (gdzie pole ma nazwę **App name** ). |
| brokerCCDurSubQueue [1] | Łańcuch | • **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE**<br>• Nazwa kolejki | Nazwa kolejki, z której konsument połączenia odbiera komunikaty subskrypcji trwałej |
| brokerCCSubbrokerCCSub [1] | Łańcuch | • **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE**<br>• Nazwa kolejki | Nazwa kolejki, z której konsument połączenia odbiera komunikaty nietrwałej subskrypcji |
| brokerControlbrokerControl [1] | Łańcuch | • **SYSTEM.BROKER.CONTROL.QUEUE**<br>• Nazwa kolejki | Nazwa kolejki sterującej brokera |
| brokerQueuekolejki brokera [1] | Łańcuch | • **"" (pusty łańcuch)**<br>• Nazwa menedżera kolejek | Nazwa menedżera kolejek, w którym działa broker |
| brokerSubbrokerSub [1] | Łańcuch | • **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE**<br>• Nazwa kolejki | Nazwa kolejki, z której konsument komunikatów nietrwałych odbiera komunikaty |
| brokerVersion [1] | Łańcuch | • **unspecified** (nieokreślony)-po przeprowadzeniu migracji brokera z wersji V6 do wersji V7należy ustawić tę właściwość, aby nagłówki RFH2 nie były już używane. Po migracji ta właściwość nie jest już istotna.<br>• **V1** -aby użyć brokera IBM MQ publish/subscribe broker.This jest wartością domyślną, jeśli właściwość TRANSPORT ma wartość BIND lub CLIENT.<br>• **V2** -Aby użyć brokera IBM Integration Bus w trybie rodzimym. Ta wartość jest wartością domyślną, jeśli parametr TRANSPORT ma wartość DIRECT lub DIRECTHTTP. | Wersja używanego brokera |

| *Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)* | | | |
|---|---|---|---|
| **Nazwa właściwości** | **Typ** | **Poprawne wartości (wartość domyślna pogrubiona)** | **Opis** |
| ccdtURL | Łańcuch | • **Null**<br>• Adres URL (URL) | URL , który identyfikuje nazwę i położenie pliku zawierającego tabelę definicji kanału klienta (CCDT) oraz określa sposób dostępu do pliku. |
| CCSID | Łańcuch | • **819**<br>• Identyfikator kodowanego zestawu znaków obsługiwany przez wirtualną maszynę języka Java (JVM) | Identyfikator kodowanego zestawu znaków dla połączenia |
| kanał | Łańcuch | • **SYSTEM.DEF.SVRCONN**<br>• Nazwa kanału MQI | Nazwa kanału MQI, który ma być używany |
| cleanupInterval [1] | int | • **3 600 000**<br>• Dodatnia liczba całkowita | Odstęp czasu (w milisekundach) między uruchomieniami w tle programu narzędziowego procedury czyszczącej publikowania/subskrypcji |
| cleanupLevel [1] | Łańcuch | • **Bezpieczne**<br>• Brak<br>• silny<br>• Wymuszenie<br>• NIEDUR | Poziom procedury czyszczącej dla składnicy subskrypcji opartej na brokerze |
| clientID | Łańcuch | • **Null**<br>• Identyfikator klienta | Identyfikator klienta dla połączenia |
| cloneSupport | Łańcuch | • **DISABLED** -w danym momencie może działać tylko jedna instancja trwałego subskrybenta tematów.<br>• ENABLED-dwie lub więcej instancji tego samego trwałego subskrybenta tematów może działać równocześnie, ale każda instancja musi działać w oddzielnej wirtualnej maszynie języka Java (JVM). | Określa, czy dwie lub więcej instancji tego samego stałego subskrybenta tematów może działać równocześnie. |

| *Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)* | | | |
|---|---|---|---|
| **Nazwa właściwości** | **Typ** | **Poprawne wartości (wartość domyślna pogrubiona)** | **Opis** |
| Wyszukiwanie connectionFactory | Łańcuch | • **Null**<br>• Nazwa JNDI obiektu ConnectionFactory . | Jeśli ta właściwość jest ustawiona, specyfikacja ActivationSpec wyszukuje obiekt JMS ConnectionFactory o określonej nazwie JNDI w przestrzeni nazw JNDI serwera aplikacji, a następnie używa właściwości tego obiektu do utworzenia połączenia produktu JMS z menedżerem kolejek produktu IBM MQ z jednym wyjątkiem. Jedyną właściwością ActivationSpec , która będzie używana podczas tworzenia połączenia JMS , jest clientID. Aby uzyskać więcej informacji, zapoznaj się z sekcją: “Właściwości ActivationSpec connectionFactoryLookup i destinationLookup” na stronie 483. |

*Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| Lista connectionName | Łańcuch | • **host lokalny (1414)**<br>• Łańcuch składający się z elementów oddzielonych przecinkami, w którym każdy element ma następujący format:<br>`HOSTNAME(PORT)`<br>gdzie *NAZWAHOSTA* jest nazwą DNS lub adresem IP. | Lista nazw połączeń TCP/IP używanych dla komunikacji przychodzącej.<br>Jeśli ta opcja jest określona, program **connectionNameList** zastępuje właściwości **hostname** i **port** .<br>Ta właściwość jest używana do ponownego nawiązywania połączenia z menedżerami kolejek z wieloma instancjami.<br>Format **connectionNameList** jest podobny do formatu **localAddress**, ale nie można go z nim mylić. Parametr **localAddress** określa parametry komunikacji lokalnej, natomiast parametr **connectionNameList** określa sposób nawiązania połączenia ze zdalnym menedżerem kolejek. |
| V 9.3.0<br>V 9.3.0<br>dynamicallyBalanced [4] | Boolowski | • **Fałsz**<br>• Prawda | Określa, czy dla tego komponentu MDB można zażądać odbierania komunikatów z innego menedżera kolejek w ramach równoważenia aplikacji w jednolitym klastrze. |
| FAILIFQUIESCE | Boolowski | • **Prawda**<br>• Fałsz | Określa, czy wywołania niektórych metod nie powiodą się, jeśli menedżer kolejek jest w stanie wyciszania |
| headerCompression | Łańcuch | • **Brak**<br>• SYSTEM-kompresja nagłówka komunikatu RLE jest wykonywana | Lista technik, których można użyć do kompresji danych nagłówka w połączeniu |

*Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| hostName | Łańcuch | • **localhost**<br>• Nazwa hosta<br>• Adres IP | Nazwa hosta lub adres IP systemu, w którym znajduje się menedżer kolejek.<br>Właściwości **hostname** i **port** są zastępowane przez właściwość **connectionNameList** , gdy jest ona określona. |
| localAddress | Łańcuch | • **Null**<br>• Łańcuch w formacie:<br>`[ host_name ][(low_port [, high_port ])]`<br>gdzie *nazwa_hosta* jest nazwą hosta lub adresem IP, *low_port* i *high_port* są numerami portów TCP, a nawiasy kwadratowe oznaczają komponent opcjonalny. | W przypadku połączenia z menedżerem kolejek ta właściwość określa jedną lub obie następujące elementy:<br>• Lokalny interfejs sieciowy, który ma być używany<br>• Port lokalny lub zakres portów lokalnych, które mają być używane<br>Format **localAddress** jest podobny do formatu **connectionNameList**, ale nie można go z nim mylić. Parametr **localAddress** określa parametry komunikacji lokalnej, natomiast parametr **connectionNameList** określa sposób nawiązania połączenia ze zdalnym menedżerem kolejek. |
| messageCompression | Łańcuch | • **Brak**<br>• Lista zawierająca jedną lub więcej następujących wartości rozdzielonych znakami odstępu:<br>RLE<br>ZLIBFAST<br>ZLIBHIGH | Lista technik, których można użyć do kompresowania danych komunikatu w połączeniu |
| messageRetention [1] | Boolowski | • **true** (prawda)-w kolejce wejściowej pozostają niepotrzebne komunikaty.<br>• false-Nieżądane komunikaty są przetwarzane zgodnie z ich opcjami dyspozycji | Określa, czy konsument połączenia przechowuje niepożądane komunikaty w kolejce wejściowej. |

| *Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)* | | | |
|---|---|---|---|
| **Nazwa właściwości** | **Typ** | **Poprawne wartości (wartość domyślna pogrubiona)** | **Opis** |
| messageSelection [1] | Łańcuch | • **client**<br>• BROKER | Określa, czy wybór komunikatów jest dokonywany przez produkt IBM MQ classes for JMS , czy przez broker. Wybór komunikatu przez broker nie jest obsługiwany, jeśli właściwość brokerVersion ma wartość 1. |
| Hasło | Łańcuch | • **Null**<br>• Hasło | Domyślne hasło używane podczas tworzenia połączenia z menedżerem kolejek |
| pollingInterval [1] | int | • **5000**<br>• Dowolna dodatnia liczba całkowita | Jeśli kolejka żadnego procesu nasłuchującego w ramach sesji nie zawiera odpowiedniego komunikatu, jest to maksymalny odstęp czasu (w milisekundach) między kolejnymi próbami pobrania komunikatu z kolejki przez każdy z procesów nasłuchujących komunikatów. Jeśli często zdarza się, że dla żadnego z obiektów nasłuchiwania komunikatów w sesji nie jest dostępny odpowiedni komunikat, należy rozważyć zwiększenie wartości tej właściwości. Ta właściwość ma zastosowanie tylko wtedy, gdy właściwość TRANSPORT ma wartość BIND lub CLIENT. |
| Port | int | • **1414**<br>• Numer portu TCP | Port, na którym nasłuchuje menedżer kolejek.<br>Właściwości **hostname** i **port** są zastępowane przez właściwość **connectionNameList** , gdy jest ona określona. |

*Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| providerVersion | string (łańcuch) | • **nieokreślona**<br>• Łańcuch w jednym z następujących formatów<br>– V.R.M.F<br>– V.R.M<br>– V.R<br>– V<br>gdzie V, R, M i F są wartościami całkowitymi większymi lub równymi zeru. | Wersja, wydanie, poziom modyfikacji i pakiet poprawek menedżera kolejek, z którym ma zostać połączony komponent MDB. |
| queueManager | Łańcuch | • **"" (pusty łańcuch)**<br>• Nazwa menedżera kolejek | Nazwa menedżera kolejek, z którym ma zostać nawiązane połączenie |
| receiveExit [3] | Łańcuch | • **Null**<br>• Łańcuch składający się z jednego lub większej liczby elementów oddzielonych przecinkami, gdzie każdy element jest pełną nazwą klasy implementującej interfejs IBM MQ classes for Java ( MQReceiveExit ). | Identyfikuje program obsługi wyjścia odbierania kanału lub sekwencję programów obsługi wyjścia odbierania, które mają być uruchamiane po sobie |
| Inicjowanie receiveExit | Łańcuch | • **Null**<br>• Łańcuch składający się z jednego lub kilku elementów danych użytkownika oddzielonych przecinkami | Dane użytkownika przekazywane do programów obsługi wyjścia odbierania kanału podczas ich wywołania |

| *Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)* | | | |
|---|---|---|---|
| **Nazwa właściwości** | **Typ** | **Poprawne wartości (wartość domyślna pogrubiona)** | **Opis** |
| rescanInterval [1] | int | • **5000**<br>• Dowolna dodatnia liczba całkowita | Jeśli konsument komunikatów w domenie typu punkt z punktem używa selektora komunikatów do wybrania komunikatów, które mają być odbierane, program IBM MQ classes for JMS wyszukuje odpowiednie komunikaty w kolejce IBM MQ w kolejności określonej przez atrybut **MsgDeliverySequence** kolejki. When IBM MQ classes for JMS finds a suitable message and delivers it to the consumer, IBM MQ classes for JMS resumes the search for the next suitable message from its current position in the queue. Produkt IBM MQ classes for JMS kontynuuje wyszukiwanie w kolejce w ten sposób, dopóki nie osiągnie końca kolejki lub dopóki nie upłynie odstęp czasu (w milisekundach) określony przez wartość tej właściwości. W każdym przypadku program IBM MQ classes for JMS powraca do początku kolejki, aby kontynuować wyszukiwanie i rozpoczyna się nowy przedział czasu. |
| securityExit [3] | Łańcuch | • **Null**<br>• Pełna nazwa klasy implementującej interfejs IBM MQ classes for Java , MQSecurityExit | Identyfikuje program obsługi wyjścia zabezpieczeń kanału |
| securityExit-inicjowanie | Łańcuch | • **Null**<br>• Łańcuch danych użytkownika | Dane użytkownika przekazywane do programu obsługi wyjścia zabezpieczeń kanału podczas jego wywołania |

*Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| sendExit [3] | Łańcuch | • **Null**<br>• Łańcuch składający się z jednego lub większej liczby elementów oddzielonych przecinkami, gdzie każdy element jest pełną nazwą klasy implementującej interfejs IBM MQ classes for Java ( MQSendExit ). | Identyfikuje program obsługi wyjścia wysyłania kanału lub sekwencję programów obsługi wyjścia wysyłania, które mają być uruchamiane po sobie |
| SENDEXITINIT | Łańcuch | • **Null**<br>• Łańcuch składający się z jednego lub kilku elementów danych użytkownika oddzielonych przecinkami | Dane użytkownika przekazywane do programów obsługi wyjścia wysyłania kanału podczas ich wywołania |
| SHARECONVALLOWED | Boolowski | • NIE-Połączenie klienta nie może współużytkować swojego gniazda.<br>• **YES** -połączenie klienta może współużytkować swoje gniazdo. | Określa, czy połączenie klienta może współużytkować gniazdo z innymi połączeniami JMS najwyższego poziomu z tego samego procesu do tego samego menedżera kolejek, jeśli definicje kanału są zgodne. |
| sparseSubscriptions [1] | Boolowski | • **false** -subskrypcje otrzymują często zgodne komunikaty.<br>• true-subskrypcje otrzymują rzadko zgodne komunikaty. Ta wartość wymaga, aby kolejka subskrypcji mogła zostać otwarta do przeglądania. | Steruje strategią pobierania komunikatów obiektu TopicSubscriber |
| Magazyny sslCert | Łańcuch | • **Null**<br>• Łańcuch zawierający jeden lub więcej adresów URL LDAP rozdzielonych odstępami. Każdy URL LDAP ma następujący format:<br>`ldap://host_name [: port ]`<br>gdzie *nazwa_hosta* jest nazwą hosta lub adresem IP, *port* jest numerem portu TCP, a nawiasy kwadratowe oznaczają komponent opcjonalny. | Serwery LDAP (Lightweight Directory Access Protocol), które przechowują listy odwołań certyfikatów (CRL) do użycia w połączeniu TLS |
| SSLCIPHERSUITE | Łańcuch | • **Null**<br>• Nazwa CipherSuite | Zestaw algorytmów szyfrowania CipherSuite , który ma być używany na potrzeby połączenia TLS |

*Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| sslFipsWymagane [2] | Boolowski | • **Fałsz**<br>• Prawda | Określa, czy połączenie TLS musi używać pakietu CipherSuite , który jest obsługiwany przez dostawcę JSSE FIPS produktu IBM Java (IBMJSSEFIPS). |
| SSLPEERNAME | Łańcuch | • **Null**<br>• Szablon nazw wyróżniających | W przypadku połączenia TLS: szablon, który jest używany do sprawdzania nazwy wyróżniającej w certyfikacie cyfrowym udostępnianym przez menedżer kolejek. |
| SSLRESETCOUNT | int | • **0**<br>• Liczba całkowita z zakresu od 0 do 999 999 999 | Łączna liczba bajtów wysłanych i odebranych przez połączenie TLS przed ponownym negocjowaniem kluczy tajnych używanych przez protokół TLS |
| Fabryka sslSocket | Łańcuch | Łańcuch reprezentujący pełną nazwę klasy udostępniającej implementację interfejsu javax.net.ssl.SSLSocketFactory . Opcjonalnie z dołączonym argumentem, który ma zostać przekazany do metody konstruktora, ujętym w nawiasy. | Wszystkie połączenia nawiązane w zasięgu obiektu administrowanego używają gniazd uzyskanych z tej implementacji interfejsu SSLSocketFactory . |
| statusRefreshPrzedział czasu [1] | int | • **60000**<br>• Dowolna dodatnia liczba całkowita | Odstęp czasu (w milisekundach) między operacjami odświeżania długotrwałych transakcji, które wykrywają, kiedy subskrybent traci połączenie z menedżerem kolejek. Ta właściwość ma zastosowanie tylko wtedy, gdy właściwość **subscriptionStore** ma wartość QUEUE. |
| subscriptionStore [1] | Łańcuch | • **BROKER**<br>• MIGRATE<br>• QUEUE | Określa miejsce, w którym program IBM MQ classes for JMS zapisuje dane trwałe dotyczące aktywnych subskrypcji. |

*Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| transportType | Łańcuch | • **client**<br>• POWIĄZANIA<br>• BINDINGS_THEN_CLIENT (POWIĄZANIA, NASTĘPNIE KLIENT) | Określa, czy połączenie z menedżerem kolejek używa trybu klienta, czy trybu powiązań. Jeśli zostanie podana wartość `BINDINGS_THEN_CLIENT` , adapter zasobów najpierw próbuje nawiązać połączenie w trybie powiązań. Jeśli ta próba nawiązania połączenia nie powiedzie się, adapter zasobów podejmie próbę nawiązania połączenia w trybie klienta.<br>z/OS Jeśli specyfikacja aktywowania działająca w systemie WebSphere Application Server for z/OS została skonfigurowana pod kątem używania trybu transportowego `BINDINGS_THEN_CLIENT` , a wcześniej nawiązane połączenie zostało zerwane, wszystkie próby ponownego nawiązania połączenia przez specyfikację aktywowania najpierw podejmą próbę użycia trybu transportowego `BINDINGS` . Jeśli próba nawiązania połączenia w trybie transportu `BINDINGS` (Powiązania) nie powiedzie się, specyfikacja aktywowania próbuje następnie nawiązać połączenie w trybie transportu `CLIENT` . |
| Nazwa użytkownika | Łańcuch | • **Null**<br>• Nazwa użytkownika | Domyślna nazwa użytkownika, która ma być używana podczas tworzenia połączenia z menedżerem kolejek |

| *Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)* | | | |
|---|---|---|---|
| **Nazwa właściwości** | **Typ** | **Poprawne wartości (wartość domyślna pogrubiona)** | **Opis** |
| wildcardFormat | Łańcuch | • CHAR-rozpoznaje tylko znaki wieloznaczne używane w brokerze w wersji 1<br>• **TOPIC** -rozpoznaje tylko znaki wieloznaczne poziomu tematu używane w wersji 2 brokera. | Która wersja składni znaku wieloznacznego ma być używana |

**Uwagi:**

1. Ta właściwość może być używana z wersją 70 produktu IBM MQ classes for JMS.
2. Ważne informacje na temat używania wymaganej właściwości sslFipszawiera sekcja "Ograniczenia adaptera zasobów IBM MQ" na stronie 455.
3. Informacje na temat konfigurowania adaptera zasobów w taki sposób, aby mógł on znaleźć wyjście, zawiera sekcja "Konfigurowanie produktu IBM MQ classes for JMS do korzystania z wyjść kanałów" na stronie 290.
4. V 9.3.0 Właściwość dynamicallyBalanced nie jest obsługiwana w połączeniu z obsługą transakcji XA. Jeśli parametr dynamicallyBalanced ma wartość "true", to komponent MDB musi być skonfigurowany do wyłączania transakcji XA.

## Właściwości używane do tworzenia konsumenta połączenia JMS

**Uwaga:** Wartości **destination** i **destinationType** muszą być zdefiniowane jawnie. Wszystkie pozostałe właściwości w pliku Tabela 65 na stronie 479 są opcjonalne.

| *Tabela 65. Właściwości obiektu ActivationSpec , które są używane do tworzenia konsumenta połączenia JMS* | | | |
|---|---|---|---|
| **Nazwa właściwości** | **Typ** | **Poprawne wartości (wartość domyślna pogrubiona)** | **Opis** |
| miejsce docelowe | Łańcuch | Nazwa miejsca docelowego | Miejsce docelowe, z którego mają być odbierane komunikaty. Właściwość **useJNDI** określa sposób interpretowania wartości tej właściwości. |
| destinationLookup | Łańcuch | • **Null**<br>• Nazwa JNDI obiektu docelowego | Jeśli ta właściwość jest ustawiona, specyfikacja ActivationSpec wyszukuje obiekt docelowy JMS o określonej nazwie JNDI w przestrzeni nazw JNDI serwera aplikacji, a następnie używa właściwości tego obiektu do utworzenia konsumenta połączenia JMS , preferując inne właściwości określone w specyfikacji ActivationSpec. Więcej informacji na ten tematzawiera sekcja "Właściwości ActivationSpec connectionFactoryLookup i destinationLookup" na stronie 483. |

*Tabela 65. Właściwości obiektu ActivationSpec , które są używane do tworzenia konsumenta połączenia JMS (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| destinationType | Łańcuch | • V 9.3.0 jakarta.jms.Queue (Jakarta Messaging 3.0)<br>• V 9.3.0 jakarta.jms.Topic (Jakarta Messaging 3.0)<br>• javax.jms.Queue (JMS 2.0)<br>• javax.jms.Topic (JMS 2.0) | Typ miejsca docelowego, kolejki lub tematu |
| maxMessages | int | • **1**<br>• Dodatnia liczba całkowita | Maksymalna liczba komunikatów, które można jednocześnie przypisać do sesji serwera. Jeśli specyfikacja aktywowania dostarcza komunikaty do komponentu MDB w transakcji XA, niezależnie od ustawienia tej właściwości używana jest wartość 1. |
| maxPoolGłębokość | int | • **10**<br>• Dodatnia liczba całkowita | Maksymalna liczba sesji serwera w puli sesji serwera używanych przez konsumenta połączenia |
| messageSelector | Łańcuch | • **Null**<br>• Wyrażenie selektora komunikatów SQL92 | Wyrażenie selektora komunikatów określające, które komunikaty mają zostać dostarczone |
| nonASFTimeout | int | • **0**<br>• Dodatnia liczba całkowita | Wartość dodatnia wskazuje, że używana jest dostawa bez ASF. Wartością jest czas (w milisekundach), przez który żądanie pobrania oczekuje na komunikaty, które nie zostały jeszcze wysłane (wywołanie pobrania z oczekiwaniem). Wartość domyślna, 0, wskazuje, że używane jest dostarczanie ASF.<br>Ten parametr jest poprawny, jeśli:<br>• Aplikacja działa w systemie WebSphere Application Server 7.0 lub nowszym.<br>• Aplikacja działa w produkcie WebSphere Liberty przy użyciu odpowiedniego poziomu składnika klienta wmqJms. Więcej informacji na ten temat zawiera sekcja “Liberty i adapter zasobów IBM MQ” na stronie 456. |

*Tabela 65. Właściwości obiektu ActivationSpec , które są używane do tworzenia konsumenta połączenia JMS (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| Włączona opcja nonASFRollback | Boolowski | • **false** (fałsz)-komunikat jest pobierany nawet wtedy, gdy działanie komponentu MDB zakończy się niepowodzeniem.<br>• true-Niepowodzenie w obrębie komponentu MDB powoduje wycofanie komunikatu do kolejki. | Określa, czy dostarczanie komunikatów znajduje się w punkcie synchronizacji IBM MQ , jeśli komponent MDB nie jest transakcją. Ignorowany, jeśli komponent MDB jest transakcją lub jeśli parametr **nonASFTimeout** jest ustawiony na wartość `0`. |
| poolTimeout | int | • **300000**<br>• Dodatnia liczba całkowita | Czas (w milisekundach), przez który nieużywana sesja serwera jest utrzymywana w puli sesji serwera przed zamknięciem z powodu nieaktywności |
| READAHEADALLOWED | int | • **DESTINATION** -określa, czy odczyt z wyprzedzeniem jest dozwolony, odwołując się do definicji kolejki lub tematu.<br>• DISABLED-odczyt z wyprzedzeniem jest niedozwolony.<br>• ENABLED-odczyt z wyprzedzeniem jest dozwolony.<br>• QUEUE-określ, czy odczyt z wyprzedzeniem jest dozwolony, odwołując się do definicji kolejki.<br>• TOPIC-Określenie, czy odczyt z wyprzedzeniem jest dozwolony, poprzez odwołanie się do definicji tematu. | Określa, czy wątek przeglądania specyfikacji aktywowania może używać odczytu z wyprzedzeniem do przeglądania wielu komunikatów z miejsca docelowego w buforze wewnętrznym przed przekazaniem do sesji serwera w celu zniszczenia.<br>**Uwaga:** Włączenie odczytu z wyprzedzeniem może spowodować zwiększenie liczby komunikatów JMSCC0108 lub zmniejszenie wydajności, jeśli szybkość przetwarzania MDB nie może nadążyć za szybkością przeglądania komunikatów z miejsca docelowego. |
| readAheadClosePolicy | int | • **ALL** -wszystkie komunikaty w wewnętrznym buforze odczytu z wyprzedzeniem są dostarczane do komponentu MDB przed jego zatrzymaniem.<br>• CURRENT-kończy się tylko bieżące wywołanie komponentu MDB, potencjalnie pozostawiając komunikaty w wewnętrznym buforze odczytu z wyprzedzeniem, które są następnie usuwane. | Co się dzieje z komunikatami w wewnętrznym buforze odczytu z wyprzedzeniem, gdy komponent MDB zostanie zatrzymany przez administratora. |

*Tabela 65. Właściwości obiektu ActivationSpec , które są używane do tworzenia konsumenta połączenia JMS (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| receiveCCSID | int | • **0** -użyj maszyny JVM `Charset.defaultCharset`<br>• 1208- UTF-8<br>• Obsługiwany identyfikator kodowanego zestawu znaków | Właściwość miejsca docelowego, która ustawia docelowy identyfikator CCSID dla konwersji komunikatów menedżera kolejek. Wartość jest ignorowana, chyba że parametr **`receiveConversion`** ma wartość `QMGR`. |
| receiveConversion | Łańcuch | • **KOMUNIKAT KLIENTA**<br>• QMGR | Właściwość miejsca docelowego, która określa, czy konwersja danych będzie wykonywana przez menedżer kolejek. |
| sharedSubscription | Boolowski | • **False** -komponent MDB nie powinien otwierać subskrypcji jako subskrypcji współużytkowanej.<br>• True (prawda)-komponent MDB powinien otworzyć subskrypcję jako subskrypcję współużytkowaną (przy użyciu reguł, które JMS 2.0 implikuje, patrz specyfikacja JMS 2.0 w serwisie Java.net). | Steruje sposobem, w jaki komponent MDB jest sterowany z subskrypcji współużytkowanej. Więcej informacji na temat używania tej właściwości zawiera sekcja "Przykłady definiowania właściwości sharedSubscription" na stronie 486. |
| startTimeout | int | • **10 000**<br>• Dodatnia liczba całkowita | Czas (w milisekundach), w którym dostarczanie komunikatu do komponentu MDB musi zostać rozpoczęte po zaplanowaniu pracy nad dostarczeniem komunikatu. Jeśli ten czas upłynie, komunikat zostanie wycofany do kolejki. |
| subscriptionDurability | Łańcuch | • **NonDurable** -Nietrwała subskrypcja jest używana do dostarczania komunikatów do komponentu MDB subskrybującego temat.<br>• Trwała-trwała subskrypcja jest używana do dostarczania komunikatów do komponentu MDB subskrybującego temat. | Określa, czy do dostarczania komunikatów do komponentu MDB subskrybującego temat używana jest trwała, czy nietrwała subskrypcja. |
| subscriptionName | Łańcuch | • **"" (pusty łańcuch)**<br>• Nazwa subskrypcji | Nazwa trwałej subskrypcji |

*Tabela 65. Właściwości obiektu ActivationSpec , które są używane do tworzenia konsumenta połączenia JMS (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| useJNDI | Boolowski | • **false** -właściwość o nazwie destination jest interpretowana jako nazwa kolejki lub tematu produktu IBM MQ .<br>• true-właściwość o nazwie destination jest interpretowana jako nazwa jednego z następujących obiektów w przestrzeni nazw JNDI serwera aplikacji:<br>– V 9.3.0 V 9.3.0 jakarta.jms.Queue (Jakarta Messaging 3.0)<br>– V 9.3.0 V 9.3.0 jakarta.jms.Topic (Jakarta Messaging 3.0)<br>– javax.jms.Queue (JMS 2.0)<br>– javax.jms.Topic (JMS 2.0) | Deprecated Określa sposób interpretowania wartości właściwości o nazwie destination.<br>**Uwaga:** Ta właściwość jest nieaktualna w wersji IBM MQ 9.0. Zamiast niej należy użyć właściwości destinationLookup . |

## Konflikty właściwości i zależności

Obiekt ActivationSpec może mieć właściwości powodujące konflikt. Na przykład można określić właściwości TLS dla połączenia w trybie powiązań. W takim przypadku zachowanie jest określane przez typ transportu i domenę przesyłania komunikatów, która jest typu punkt z punktem lub publikowanie/subskrypcja, zgodnie z właściwością **destinationType** . Wszystkie właściwości, które nie mają zastosowania do określonego typu transportu lub domeny przesyłania komunikatów, są ignorowane.

Jeśli zostanie zdefiniowana właściwość, która wymaga zdefiniowania innych właściwości, ale nie zostaną one zdefiniowane, obiekt ActivationSpec zgłosi wyjątek InvalidPropertypodczas wywoływania metody validate () podczas wdrażania komponentu MDB. Wyjątek jest zgłaszany administratorowi serwera aplikacji w sposób zależny od serwera aplikacji. Jeśli na przykład właściwość subscriptionDurability zostanie ustawiona na wartość Durable, co oznacza, że mają być używane trwałe subskrypcje, należy również zdefiniować właściwość **subscriptionName** .

Jeśli zdefiniowane są zarówno właściwości **ccdtURL** , jak i **channel** , zgłaszany jest wyjątek InvalidProperty. Jednak w przypadku zdefiniowania tylko właściwości **ccdtURL** , pozostawiając właściwość o nazwie **channel** z wartością domyślną SYSTEM.DEF.SVRCONN, nie jest zgłaszany żaden wyjątek, a tabela definicji kanału klienta identyfikowana przez właściwość **ccdtURL** jest używana do uruchamiania połączenia JMS .

## Właściwości ActivationSpec connectionFactoryLookup i destinationLookup

W specyfikacji JMS 2.0 wprowadzono dwie nowe właściwości ActivationSpec . Właściwości connectionFactoryLookup i destinationLookup mogą mieć nazwę JNDI obiektu administrowanego, która ma być używana zamiast innych właściwości ActivationSpec .

Jeśli na przykład fabryka połączeń jest zdefiniowana w pliku JNDI , a nazwa JNDI tego obiektu jest określona we właściwości wyszukiwania connectionFactorydla specyfikacji aktywowania, wszystkie

właściwości fabryki połączeń zdefiniowane w pliku JNDI są używane zamiast właściwości w pliku Tabela 64 na stronie 468.

Jeśli miejsce docelowe jest zdefiniowane w pliku JNDI , a nazwa JNDI jest ustawiona we właściwości destinationLookup specyfikacji ActivationSpec, to wartości, które są używane zamiast wartości w pliku Tabela 65 na stronie 479, są używane. Więcej informacji na temat używania tych dwóch właściwości zawiera sekcja “Właściwości ActivationSpec connectionFactoryLookup i destinationLookup” na stronie 483.

Te dwie właściwości mogą być używane do określania JNDI nazw obiektów ConnectionFactory i obiektów Destination, które są preferowane względem właściwości specyfikacji ActivationSpec zdefiniowanych w Tabela 64 na stronie 468 i Tabela 65 na stronie 479.

Należy zwrócić uwagę na następujące punkty, które szczegółowo opisują sposób działania tych właściwości.

**Wyszukiwanie connectionFactory**
: Element ConnectionFactory wyszukiwany w produkcie JNDI jest używany jako źródło właściwości wymienionych w sekcji Tabela 64 na stronie 468. Obiekt ConnectionFactory nie jest używany do tworzenia połączeń JMS . Tylko właściwości obiektu są odpytywane. Te właściwości z fabryki ConnectionFactory przesłaniają wszystkie właściwości zdefiniowane w specyfikacji ActivationSpec. Istnieje od tego jeden wyjątek. Jeśli właściwość ActivationSpec ma ustawioną właściwość **ClientID** , wartość tej właściwości przesłania wartość określoną w polu ConnectionFactory. Dzieje się tak dlatego, że w typowym scenariuszu używana jest jedna fabryka połączeń ConnectionFactory z wieloma obiektami ActivationSpecs. Upraszcza to administrowanie. Jednak specyfikacja JMS 2.0 określa, że każde JMS połączenie utworzone z ConnectionFactory powinno mieć unikalną wartość **ClientID**. Z tego powodu opcja ActivationSpecs musi mieć możliwość nadpisania dowolnej wartości ustawionej w obiekcie ConnectionFactory. Jeśli w specyfikacji ActivationSpecnie jest ustawiona żadna wartość **ClientID** , używana jest dowolna wartość z fabryki połączeń.

**destinationLookup**
: Właściwości **Destination** i **UseJndi** są zdefiniowane w specyfikacji aktywowania ActivationSpec. Jeśli opcja **UseJndi** ma wartość `true`, tekst określony we właściwości miejsca docelowego jest traktowany jako nazwa JNDI , a obiekt docelowy o tej nazwie JNDI jest wyszukiwany w pliku JNDI.

  Właściwość destinationLookup zachowuje się dokładnie tak samo. Jeśli została ustawiona, obiekt docelowy o nazwie JNDI określonej przez właściwość jest wyszukiwany w pliku JNDI. Ta właściwość ma pierwszeństwo przed właściwością **useJNDI** .

  Deprecated Właściwość useJNDI jest nieaktualna w wersji IBM MQ 9.0 , ponieważ właściwość **destinationLookup** jest specyfikacją JMS 2.0 lub późniejszym odpowiednikiem wykonania tej samej funkcji.

## Właściwości ActivationSpec bez odpowiedników w produkcie IBM MQ classes for JMS

Większość właściwości obiektu ActivationSpec są równoważne właściwościom obiektów IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging albo parametrom metod IBM MQ classes for JMS IBM MQ classes for Jakarta Messaging . Jednak trzy właściwości strojenia i jedna właściwość łatwości używania nie mają odpowiedników w IBM MQ classes for JMS lub IBM MQ classes for Jakarta Messaging:

**startTimeout**
: Czas (w milisekundach), przez który menedżer pracy serwera aplikacji oczekuje na udostępnienie zasobów po zaplanowaniu przez adapter zasobów obiektu pracy w celu dostarczenia komunikatu do komponentu MDB. Jeśli ten czas upłynie przed rozpoczęciem dostarczania komunikatu, nastąpi przekroczenie limitu czasu obiektu roboczego, komunikat zostanie wycofany do kolejki, a adapter zasobów będzie mógł podjąć ponowną próbę dostarczenia komunikatu. Ostrzeżenie jest zapisywane w danych śledzenia diagnostycznego, jeśli jest włączone, ale nie ma wpływu na proces dostarczania komunikatów. Można oczekiwać, że ten warunek wystąpi tylko wtedy, gdy serwer aplikacji jest bardzo obciążony. Jeśli warunek występuje regularnie, należy rozważyć zwiększenie wartości tej właściwości, aby zapewnić menedżerowi pracy więcej czasu na zaplanowanie dostarczania komunikatów.

**maxPoolGłębokość**
Maksymalna liczba sesji serwera w puli sesji serwera używanych przez konsumenta połączenia. Po utworzeniu sesji serwera rozpoczyna ona konwersację z menedżerem kolejek. Konsument połączenia używa sesji serwera do dostarczenia komunikatu do komponentu MDB. Większa głębokość puli pozwala na współbieżne dostarczanie większej liczby komunikatów w sytuacjach dużego wolumenu, ale zużywa więcej zasobów serwera aplikacji. Jeśli ma zostać wdrożonych wiele komponentów MDB, należy rozważyć zmniejszenie głębokości puli w celu utrzymania obciążenia serwera aplikacji na poziomie umożliwiającym zarządzanie. Każdy konsument połączenia używa własnej puli sesji serwera, dlatego ta właściwość nie definiuje łącznej liczby sesji serwera dostępnych dla wszystkich konsumentów połączenia.

**poolTimeout**
Czas (w milisekundach), przez który nieużywana sesja serwera jest otwarta w puli sesji serwera przed zamknięciem z powodu braku aktywności. Przejściowy wzrost obciążenia komunikatami powoduje utworzenie dodatkowych sesji serwera w celu rozdzielenia obciążenia, ale po powrocie do normalnego obciążenia komunikatami dodatkowe sesje serwera pozostają w puli i nie są używane.

Za każdym razem, gdy używana jest sesja serwera, jest ona oznaczana znacznikiem czasu. Okresowo wątek scavenger sprawdza, czy każda sesja serwera była używana w okresie określonym przez tę właściwość. Jeśli sesja serwera nie została użyta, zostanie zamknięta i usunięta z puli sesji serwera. Sesja serwera może nie zostać zamknięta natychmiast po upływie podanego okresu. Ta właściwość reprezentuje minimalny okres braku aktywności przed usunięciem.

**useJNDI**
Opis tej właściwości zawiera sekcja Tabela 65 na stronie 479.

## Wdrażanie komponentu MDB

Aby wdrożyć komponent MDB, należy najpierw zdefiniować właściwości obiektu ActivationSpec , określając właściwości wymagane przez komponent MDB. W poniższym przykładzie przedstawiono typowy zestaw właściwości, które można jawnie zdefiniować:

V 9.3.0 JM 3.0 V 9.3.0

```
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
```

Serwer aplikacji używa tych właściwości do utworzenia obiektu ActivationSpec , który jest następnie powiązany z komponentem MDB. Właściwości obiektu ActivationSpec określają sposób dostarczania komunikatów do komponentu MDB. Wdrożenie komponentu MDB kończy się niepowodzeniem, jeśli komponent MDB wymaga transakcji rozproszonych, ale adapter zasobów nie obsługuje transakcji rozproszonych. Informacje na temat instalowania adaptera zasobów w celu obsługi transakcji rozproszonych zawiera sekcja “Instalowanie adaptera zasobów IBM MQ” na stronie 458.

Jeśli więcej niż jeden komponent MDB odbiera komunikaty z tego samego miejsca docelowego, komunikat wysłany w domenie typu punkt z punktem jest odbierany tylko przez jeden komponent MDB, nawet jeśli inne komponenty MDB są uprawnione do odbierania komunikatu. W szczególności, jeśli dwie bazy danych MDB używają różnych selektorów komunikatów, a komunikat przychodzący jest zgodny

z obydwoma selektorami komunikatów, tylko jedna z nich odbiera komunikat. Komponent MDB wybrany do odebrania komunikatu jest niezdefiniowany i nie można polegać na konkretnym komunikacie odebrany przez komponent MDB. Komunikaty wysyłane w domenie publikowania/subskrypcji są odbierane przez wszystkie zakwalifikowane komponenty MDB.

W pewnych okolicznościach komunikat dostarczony do komponentu MDB może zostać wycofany do kolejki IBM MQ . Taka sytuacja może mieć miejsce na przykład wtedy, gdy komunikat jest dostarczany w ramach jednostki pracy, która jest następnie wycofywana. Wycofany komunikat jest ponownie dostarczany, ale niepoprawnie sformatowany komunikat może wielokrotnie spowodować niepowodzenie komponentu MDB i dlatego nie można go dostarczyć. Taka wiadomość jest nazywana wiadomością nieprzetwarzaną. Produkt IBM MQ można skonfigurować w taki sposób, aby produkt IBM MQ classes for JMS automatycznie przekazywał komunikat nieprzetwarzalny do innej kolejki w celu dalszego badania lub usunięcia komunikatu.

Szczegółowe informacje na temat obsługi komunikatów nieprzetwarzalnych zawiera sekcja “Obsługa komunikatów nieprzetwarzalnych w produkcie IBM MQ classes for JMS” na stronie 240.

**Pojęcia pokrewne**

Określenie, że w czasie wykonywania na kliencie MQI będą używane tylko CipherSpecs z certyfikatem FIPS.

Standardy FIPS (Federal Information Processing Standards) dla AIX, Linux, and Windows

**Zadania pokrewne**

Konfigurowanie zasobów JMS na serwerze WebSphere Application Server

*Przykłady definiowania właściwości sharedSubscription*

Właściwość sharedSubscription specyfikacji aktywowania można zdefiniować w pliku WebSphere Liberty `server.xml` . Inną możliwością jest zdefiniowanie właściwości w komponencie bean sterowanym komunikatami (MDB) przy użyciu adnotacji.

## Przykład: definiowanie w pliku Liberty `server.xml`

W pliku WebSphere Liberty `server.xml` należy zdefiniować specyfikację aktywowania, jak pokazano w poniższym przykładzie. W tym przykładzie tworzona jest trwała subskrypcja współużytkowana menedżera kolejek na hoście lokalnym/porcie 1490.

```
<jmsActivationSpec id="SubApp/SubscribingEJB/SubscribingMDB" authDataRef="JMSConnectionAlias">
<properties.wmqJms hostName="localhost" port="1490" maxPoolDepth="5"
subscriptionName="MySubName"
subscriptionDurability="DURABLE" sharedSubscription="true"/>
</jmsActivationSpec>
```

## Przykład: definiowanie w obrębie komponentu MDB

Można również zdefiniować właściwość sharedSubscription w obrębie komponentu MDB przy użyciu adnotacji, jak to pokazano w poniższym przykładzie:

```
@ActioncationConfigProperty(propertyName ="sharedSubscription",
propertyValue = "true")
```

W poniższym przykładzie przedstawiono fragment kodu komponentu MDB, który używa metody adnotacji:

V 9.3.0 JM 3.0 V 9.3.0

```
/**
 * 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
   }

}
```

**Pojęcia pokrewne**

Subskrybenci i subskrypcje

Trwałość subskrypcji

“Subskrypcje sklonowane i współużytkowane” na stronie 336

W produkcie IBM MQ 8.0 lub nowszym istnieją dwie metody nadawania wielu konsumentom dostępu do tej samej subskrypcji. Te dwie metody są realizowane za pomocą sklonowanych subskrypcji lub za pomocą subskrypcji współużytkowanych.

### *Konfigurowanie adaptera zasobów na potrzeby komunikacji wychodzącej*

Aby skonfigurować komunikację wychodzącą, należy zdefiniować właściwości obiektu ConnectionFactory i administrowanego obiektu docelowego.

## Przykład użycia komunikacji wychodzącej

W przypadku korzystania z komunikacji wychodzącej aplikacja działająca na serwerze aplikacji uruchamia połączenie z menedżerem kolejek, a następnie wysyła komunikaty do swoich kolejek i odbiera komunikaty ze swoich kolejek w sposób synchroniczny. Na przykład następująca metoda serwletu, doGet(), używa komunikacji wychodzącej:

V 9.3.0 JM 3.0 V 9.3.0

```
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();
}
```

Gdy serwlet odbiera żądanie HTTP GET, pobiera obiekt ConnectionFactory i obiekt Queue z przestrzeni nazw JNDI oraz używa tych obiektów do wysłania komunikatu do kolejki IBM MQ . Następnie serwlet odbiera wysłany przez niego komunikat.

## Zasoby wymagane do komunikacji wychodzącej

Aby skonfigurować komunikację wychodzącą, należy zdefiniować zasoby Java EE Connector Architecture (JCA) w następujących kategoriach:

- Właściwości obiektu ConnectionFactoryużywanego przez serwer aplikacji do tworzenia obiektu JMS ConnectionFactory .
- Właściwości administrowanego obiektu docelowego, którego serwer aplikacji używa do utworzenia obiektu kolejki JMS lub obiektu tematu JMS .

Sposób definiowania tych właściwości zależy od interfejsów administracyjnych udostępnianych przez serwer aplikacji. ConnectionFactory, Queue i Topic utworzone przez serwer aplikacji są powiązane z przestrzenią nazw JNDI, z której mogą być pobierane przez aplikację.

Zwykle dla każdego menedżera kolejek, z którym może być konieczne nawiązanie połączenia przez aplikacje, definiowany jest jeden obiekt ConnectionFactory . Należy zdefiniować jeden obiekt kolejki dla każdej kolejki, do której aplikacje mogą potrzebować dostępu w domenie typu punkt z punktem. Dla każdego tematu, który może być publikowany lub subskrybowany przez aplikacje, należy zdefiniować jeden obiekt tematu. Obiekt ConnectionFactory może być niezależny od domeny. Alternatywnie może to być obiekt specyficzny dla domeny, obiekt fabryki QueueConnectiondla domeny typu punkt z punktem lub obiekt fabryki TopicConnectiondla domeny publikowania/subskrypcji.

**Wskazówka:** W produkcie JMS 2.0fabryka połączeń może być używana do tworzenia zarówno połączeń, jak i kontekstów. W związku z tym możliwe jest powiązanie puli połączeń z fabryką połączeń, która zawiera zarówno połączenia, jak i konteksty. Zaleca się, aby fabryka połączeń była używana tylko do tworzenia połączeń lub kontekstów. Dzięki temu pula połączeń dla tej fabryki połączeń będzie zawierać tylko obiekty pojedynczego typu, co spowoduje, że pula będzie bardziej wydajna.

## Właściwości obiektu ConnectionFactory

Tabela 66 na stronie 489 zawiera listę właściwości obiektu ConnectionFactory . Serwer aplikacji używa tych właściwości do utworzenia obiektu JMS ConnectionFactory .

*Tabela 66. Właściwości obiektu ConnectionFactory*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| applicationName | Łańcuch | • Nazwa klasy wywołującej, jeśli jest dostępna, nie może być dłuższa niż 28 znaków. Jeśli nie jest dostępny, zostanie użyty łańcuch `WebSphere MQ Client for Java` . | Nazwa, pod którą aplikacja jest rejestrowana w menedżerze kolejek. Ta nazwa aplikacji jest wyświetlana przez komendę **`DISPLAY CONN MQSC/PCF`** (gdzie pole ma nazwę **`APPLTAG`**) lub na ekranie **Połączenia aplikacji** w Eksploratorze IBM MQ (gdzie pole ma nazwę **App name**). |
| Kolejka brokerCCSub | Łańcuch | • **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE**<br>• Nazwa kolejki | Nazwa kolejki, z której konsument połączenia odbiera komunikaty nietrwałej subskrypcji. |
| Kolejka brokerControl | Łańcuch | • **SYSTEM.BROKER.CONTROL.QUEUE**<br>• Nazwa kolejki | Nazwa kolejki sterującej brokera. |
| Kolejka brokerPub | Łańcuch | • **SYSTEM.BROKER.DEFAULT.STREAM**<br>• Nazwa kolejki | Nazwa kolejki, do której wysyłane są opublikowane komunikaty (kolejka strumienia). |
| Menedżer brokerQueue | Łańcuch | • **"" (pusty łańcuch)**<br>• Nazwa menedżera kolejek | Nazwa menedżera kolejek, w którym działa broker. |

*Tabela 66. Właściwości obiektu ConnectionFactory (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| Kolejka brokerSub | Łańcuch | • **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE**<br>• Nazwa kolejki | Nazwa kolejki, z której konsument nietrwałych komunikatów odbiera komunikaty.<br>Więcej informacji na ten temat zawiera opis właściwości BROKERSUBQ . |
| brokerVersion | Łańcuch | • **unspecified** (nieokreślony)-po przeprowadzeniu migracji brokera z wersji V6 do wersji V7należy ustawić tę właściwość, aby nagłówki RFH2 nie były już używane. Po migracji ta właściwość nie jest już istotna.<br>• **V1** -aby użyć brokera publikowania/subskrypcji produktu IBM MQ . Ta wartość jest wartością domyślną, jeśli dla parametru TRANSPORT ustawiono wartość BIND lub CLIENT.<br>• **V2** -Aby użyć brokera IBM Integration Bus w trybie rodzimym. Ta wartość jest wartością domyślną, jeśli parametr TRANSPORT ma wartość DIRECT lub DIRECTHTTP. | Wersja używanego brokera. |
| ccdtURL | Łańcuch | • **Null**<br>• Adres URL (URL) | URL identyfikujący nazwę i położenie pliku zawierającego tabelę definicji kanału klienta (CCDT) oraz określający sposób dostępu do pliku. |
| CCSID | Łańcuch | • **819**<br>• Identyfikator kodowanego zestawu znaków obsługiwany przez wirtualną maszynę języka Java (JVM) | Identyfikator kodowanego zestawu znaków dla połączenia. |
| kanał | Łańcuch | • **SYSTEM.DEF.SVRCONN**<br>• Nazwa kanału MQI | Nazwa kanału MQI, który ma być używany. |
| cleanupInterval | int | • **3 600 000**<br>• Dodatnia liczba całkowita | Odstęp czasu (w milisekundach) między uruchomieniami w tle programu narzędziowego do czyszczenia publikowania/subskrypcji. |
| cleanupLevel | Łańcuch | • **Bezpieczne**<br>• Brak<br>• silny<br>• Wymuszenie<br>• NIEDUR | Poziom procedury czyszczącej dla składnicy subskrypcji opartej na brokerze. |

*Tabela 66. Właściwości obiektu ConnectionFactory (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| clientID | Łańcuch | • **Null**<br>• Identyfikator klienta | Identyfikator klienta dla połączenia. |
| cloneSupport | Łańcuch | • **DISABLED** -w danym momencie może działać tylko jedna instancja trwałego subskrybenta tematów.<br>• ENABLED-dwie lub więcej instancji tego samego trwałego subskrybenta tematów może działać równocześnie, ale każda instancja musi działać w oddzielnej wirtualnej maszynie języka Java (JVM). | Określa, czy dwie lub więcej instancji tego samego trwałego subskrybenta tematu może działać równocześnie. |
| Lista connectionName | Łańcuch | • **host lokalny (1414)**<br>• Łańcuch składający się z elementów oddzielonych przecinkami, w którym każdy element ma następujący format:<br>`HOSTNAME(PORT)`<br>gdzie *NAZWAHOSTA* jest nazwą DNS lub adresem IP. | Lista nazw połączeń TCP/IP używanych do komunikacji wychodzącej.<br>**connectionNameList** zastępuje właściwości **hostname** i **port** .<br>Ta właściwość jest używana do ponownego nawiązywania połączenia z menedżerami kolejek z wieloma instancjami.<br>Format **connectionNameList** jest podobny do formatu **localAddress**, ale nie można go z nim mylić. Parametr **localAddress** określa parametry komunikacji lokalnej, natomiast parametr **connectionNameList** określa sposób nawiązania połączenia ze zdalnym menedżerem kolejek. |
| FAILIFQUIESCE | Boolowski | • **Prawda**<br>• Fałsz | Określa, czy wywołania niektórych metod nie powiodą się, jeśli menedżer kolejek jest w stanie wyciszania. |
| headerCompression | Łańcuch | • **Brak**<br>• Kompresja nagłówka komunikatu SYSTEM-RLE jest wykonywana. | Lista technik, których można użyć do kompresji danych nagłówka w połączeniu. |
| hostName | Łańcuch | • **localhost**<br>• Nazwa hosta<br>• Adres IP | Nazwa hosta lub adres IP systemu, w którym znajduje się menedżer kolejek.<br>Właściwości **hostname** i **port** są zastępowane przez właściwość **connectionNameList** , gdy jest ona określona. |

*Tabela 66. Właściwości obiektu ConnectionFactory (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| localAddress | Łańcuch | • **Null**<br>• Łańcuch w formacie:<br>`[ host_name ][(low_port [, high_port ])]`<br>gdzie *nazwa_hosta* jest nazwą hosta lub adresem IP, *low_port* i *high_port* są numerami portów TCP, a nawiasy kwadratowe oznaczają komponent opcjonalny. | W przypadku połączenia z menedżerem kolejek ta właściwość określa jedną lub obie z następujących wartości:<br>• Lokalny interfejs sieciowy, który ma być używany<br>• Port lokalny lub zakres portów lokalnych, które mają być używane<br>Format **localAddress** jest podobny do formatu **connectionNameList**, ale nie można go z nim mylić. Parametr **localAddress** określa parametry komunikacji lokalnej, natomiast parametr **connectionNameList** określa sposób nawiązania połączenia ze zdalnym menedżerem kolejek. |
| messageCompression | Łańcuch | • **Brak**<br>• Lista zawierająca jedną lub więcej następujących wartości rozdzielonych znakami odstępu:<br>RLE<br>ZLIBFAST<br>ZLIBHIGH | Lista technik, których można użyć do kompresji danych komunikatu w połączeniu. |
| messageSelection | Łańcuch | • **client**<br>• BROKER | Określa, czy wybór komunikatów jest dokonywany przez produkt IBM MQ classes for JMS , czy przez broker. Wybór komunikatów przez broker nie jest obsługiwany, jeśli **brokerVersion** ma wartość 1. |
| Hasło | Łańcuch | • **Null**<br>• Hasło | Hasło domyślne używane podczas tworzenia połączenia z menedżerem kolejek. |

*Tabela 66. Właściwości obiektu ConnectionFactory (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| pollingInterval | int | • **5000**<br>• Dowolna dodatnia liczba całkowita | Jeśli kolejka żadnego procesu nasłuchującego w ramach sesji nie zawiera odpowiedniego komunikatu, jest to maksymalny odstęp czasu (w milisekundach) między kolejnymi próbami pobrania komunikatu z kolejki przez każdy z procesów nasłuchujących komunikatów. Jeśli często zdarza się, że dla żadnego z obiektów nasłuchiwania komunikatów w sesji nie jest dostępny odpowiedni komunikat, należy rozważyć zwiększenie wartości tej właściwości. Ta właściwość ma zastosowanie tylko wtedy, gdy właściwość **TRANSPORT** ma wartość `BIND` lub `CLIENT`. |
| Port | int | • **1414**<br>• Numer portu TCP | Port, na którym nasłuchuje menedżer kolejek.<br>Właściwości **hostname** i **port** są zastępowane przez właściwość **connectionNameList** , gdy jest ona określona. |
| providerVersion | string (łańcuch) | • **nieokreślona**<br>• Łańcuch w jednym z następujących formatów<br>– V.R.M.F<br>– V.R.M<br>– V.R<br>– V<br>gdzie V, R, M i F są wartościami całkowitymi większymi lub równymi zeru. | Wersja, wydanie, poziom modyfikacji i pakiet poprawek menedżera kolejek, z którym aplikacja ma nawiązać połączenie. |
| Odstęp czasu pubAck | int | • **25**<br>• Dodatnia liczba całkowita | Liczba komunikatów opublikowanych przez publikator, zanim produkt IBM MQ classes for JMS zażąda potwierdzenia od brokera. |
| queueManager | Łańcuch | • **"" (pusty łańcuch)**<br>• Nazwa menedżera kolejek | Nazwa menedżera kolejek, z którym ma zostać nawiązane połączenie. |

*Tabela 66. Właściwości obiektu ConnectionFactory (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| receiveExit [3] | Łańcuch | • **Null**<br>• Łańcuch składający się z jednego lub większej liczby elementów oddzielonych przecinkami, gdzie każdy element jest pełną nazwą klasy implementującej interfejs IBM MQ classes for Java ( MQReceiveExit ). | Identyfikuje program obsługi wyjścia odbierania kanału lub sekwencję programów obsługi wyjścia odbierania, które mają być uruchamiane po sobie. |
| Inicjowanie receiveExit | Łańcuch | • **Null**<br>• Łańcuch składający się z jednego lub kilku elementów danych użytkownika oddzielonych przecinkami | Dane użytkownika przekazywane do programów obsługi wyjścia odbierania kanału podczas ich wywołania. |
| rescanInterval | int | • **5000**<br>• Dowolna dodatnia liczba całkowita | Jeśli konsument komunikatów w domenie typu punkt z punktem używa selektora komunikatów do wybrania komunikatów, które mają być odbierane, program IBM MQ classes for JMS wyszukuje odpowiednie komunikaty w kolejce IBM MQ w kolejności określonej przez atrybut **MsgDeliverySequence** kolejki. When IBM MQ classes for JMS finds a suitable message and delivers it to the consumer, IBM MQ classes for JMS resumes the search for the next suitable message from its current position in the queue. Produkt IBM MQ classes for JMS kontynuuje wyszukiwanie w kolejce w ten sposób, dopóki nie osiągnie końca kolejki lub dopóki nie upłynie odstęp czasu (w milisekundach) określony przez wartość tej właściwości. W każdym przypadku program IBM MQ classes for JMS powraca do początku kolejki, aby kontynuować wyszukiwanie i rozpoczyna się nowy przedział czasu. |
| securityExit [3] | Łańcuch | • **Null**<br>• Pełna nazwa klasy implementującej interfejs IBM MQ classes for Java , MQSecurityExit | Identyfikuje program obsługi wyjścia zabezpieczeń kanału. |
| securityExit-inicjowanie | Łańcuch | • **Null**<br>• Łańcuch danych użytkownika | Dane użytkownika przekazywane do programu obsługi wyjścia zabezpieczeń kanału podczas jego wywołania. |

*Tabela 66. Właściwości obiektu ConnectionFactory (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| SENDCHECKCOUNT | int | • **0**<br>• Dowolna dodatnia liczba całkowita | Liczba wywołań wysyłania dozwolonych między sprawdzeniami pod kątem błędów asynchronicznego umieszczania w ramach pojedynczej sesji JMS bez transakcji. |
| sendExit [3] | Łańcuch | • **Null**<br>• Łańcuch składający się z jednego lub większej liczby elementów oddzielonych przecinkami, gdzie każdy element jest pełną nazwą klasy implementującej interfejs IBM MQ classes for Java ( MQSendExit ). | Identyfikuje program obsługi wyjścia wysyłania kanału lub sekwencję programów obsługi wyjścia wysyłania, które mają być uruchamiane po sobie. |
| SENDEXITINIT | Łańcuch | • **Null**<br>• Łańcuch składający się z jednego lub kilku elementów danych użytkownika oddzielonych przecinkami | Dane użytkownika przekazywane do programów obsługi wyjścia wysyłania kanału podczas ich wywołania. |
| SHARECONVALLOWED | Boolowski | • NIE-Połączenie klienta nie może współużytkować swojego gniazda.<br>• **YES** -połączenie klienta może współużytkować swoje gniazdo. | Określa, czy połączenie klienta może współużytkować gniazdo z innymi połączeniami JMS najwyższego poziomu z tego samego procesu do tego samego menedżera kolejek, jeśli definicje kanału są zgodne. |
| sparseSubscriptions | Boolowski | • **false** -subskrypcje otrzymują często zgodne komunikaty.<br>• true-subskrypcje otrzymują rzadko zgodne komunikaty. Ta wartość wymaga, aby kolejka subskrypcji mogła zostać otwarta do przeglądania. | Steruje strategią pobierania komunikatów obiektu TopicSubscriber . |
| Magazyny sslCert | Łańcuch | • **Null**<br>• Łańcuch zawierający jeden lub więcej adresów URL LDAP rozdzielonych odstępami. Każdy URL LDAP ma następujący format:<br>`ldap://host_name [: port ]`<br>gdzie *nazwa_hosta* jest nazwą hosta lub adresem IP, *port* jest numerem portu TCP, a nawiasy kwadratowe oznaczają komponent opcjonalny. | Serwery LDAP (Lightweight Directory Access Protocol), które przechowują listy odwołań certyfikatów (CRL) do użycia w połączeniu TLS. |
| SSLCIPHERSUITE | Łańcuch | • **Null**<br>• Nazwa CipherSuite | CipherSuite , który ma być używany dla połączenia TLS. |

*Tabela 66. Właściwości obiektu ConnectionFactory (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| sslFipsWymagane [2] | Boolowski | • **Fałsz**<br>• Prawda | Określa, czy połączenie TLS musi używać pakietu CipherSuite obsługiwanego przez dostawcę IBM Java JSSE FIPS (IBMJSSEFIPS). |
| SSLPEERNAME | Łańcuch | • **Null**<br>• Szablon nazw wyróżniających | W przypadku połączenia TLS jest to szablon używany do sprawdzania nazwy wyróżniającej w certyfikacie cyfrowym udostępnianym przez menedżer kolejek. |
| SSLRESETCOUNT | int | • **0**<br>• Liczba całkowita z zakresu od 0 do 999 999 999 | Łączna liczba bajtów wysłanych i odebranych przez połączenie TLS przed ponownym negocjowaniem kluczy tajnych używanych przez protokół TLS. |
| Fabryka sslSocket | Łańcuch | Łańcuch reprezentujący pełną nazwę klasy udostępniającej implementację interfejsu javax.net.ssl.SSLSocketFactory , opcjonalnie zawierający argument, który ma zostać przekazany do metody konstruktora, ujęty w nawiasy. | Wszystkie połączenia nawiązane w zasięgu administrowanego obiektu docelowego używają gniazd uzyskanych z tej implementacji interfejsu SSLSocketFactory . |
| Przedział czasu statusRefresh | int | • **60000**<br>• Dowolna dodatnia liczba całkowita | Odstęp czasu (w milisekundach) między operacjami odświeżania długotrwałych transakcji, które wykrywają, kiedy subskrybent traci połączenie z menedżerem kolejek. Ta właściwość ma zastosowanie tylko wtedy, gdy właściwość **SUBSTORE** ma wartość QUEUE. |
| subscriptionStore | Łańcuch | • **BROKER**<br>• MIGRATE<br>• QUEUE | Określa miejsce, w którym program IBM MQ classes for JMS zapisuje dane trwałe dotyczące aktywnych subskrypcji. |

| *Tabela 66. Właściwości obiektu ConnectionFactory (kontynuacja)* | | | |
|---|---|---|---|
| **Nazwa właściwości** | **Typ** | **Poprawne wartości (wartość domyślna pogrubiona)** | **Opis** |
| targetClientZgodne | Boolowski | • **Prawda**<br>• Fałsz | Określa, czy komunikat odpowiedzi wysyłany do kolejki identyfikowanej przez pole nagłówka JMSReplyTo komunikatu przychodzącego ma nagłówek MQRFH2 tylko wtedy, gdy komunikat przychodzący ma nagłówek MQRFH2 .<br><br>Tę właściwość można również skonfigurować dla specyfikacji aktywowania. Więcej informacji na ten temat zawiera “Konfigurowanie właściwości uzgadniania targetClientdla specyfikacji aktywowania” na stronie 507. |

*Tabela 66. Właściwości obiektu ConnectionFactory (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| temporaryModel | Łańcuch | • **SYSTEM.DEFAULT.MODEL.QUEUE**<br>• SYSTEM.JMS.TEMPQ.MODEL<br>• Dowolny łańcuch | Nazwa kolejki modelowej, na podstawie której są tworzone tymczasowe kolejki produktu JMS .<br>Użyj SYSTEM.DEFAULT.MODEL.QUEUE , jeśli spełnione są oba poniższe warunki:<br>• Aplikacja używa kolejki tymczasowej, która będzie akceptować komunikaty nietrwałe.<br>• Tylko jedna aplikacja utworzy w menedżerze kolejek kolejkę tymczasową, na którą wskazuje ConnectionFactory w danym momencie. Należy zauważyć, że SYSTEM.DEFAULT.MODEL.QUEUE może być jednocześnie otwarta tylko przez jedną aplikację.<br>Użyj SYSTEM.JMS.TEMPQ.MODEL w następujących sytuacjach:<br>• Gdy aplikacja używa kolejki tymczasowej, która będzie akceptować komunikaty trwałe.<br>• Jeśli wiele aplikacji może nawiązać połączenie z menedżerem kolejek, który wskazuje fabryka połączeń ConnectionFactory , a te aplikacje muszą jednocześnie tworzyć kolejki tymczasowe.<br>Zdefiniuj nową kolejkę modelową z atrybutem **DEFPSIST** ustawionym na YESi atrybutem **DEFSOPT** ustawionym na SHARED w następującej sytuacji:<br>• Jeśli aplikacja używa kolejki tymczasowej, która będzie akceptować nietrwałe komunikaty, a wiele aplikacji połączy się z menedżerem kolejek, do którego wskazuje fabryka połączeń ConnectionFactory , a te aplikacje będą musiały jednocześnie tworzyć kolejki tymczasowe.<br>Podczas tworzenia nowej kolejki modelowej należy ustawić właściwość **temporaryModel** na nazwę nowej kolejki modelowej. |

*Tabela 66. Właściwości obiektu ConnectionFactory (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| tempQPrefix | Łańcuch | • **"" (pusty łańcuch)**<br>• Przedrostek, którego można użyć do utworzenia nazwy kolejki dynamicznej IBM MQ . Reguły tworzenia przedrostka są takie same, jak reguły tworzenia treści pola **DynamicQName** w deskryptorze obiektu IBM MQ o strukturze MQOD, ale ostatni niepusty znak musi być znakiem gwiazdki (*). Jeśli wartość właściwości jest pustym łańcuchem, produkt IBM MQ classes for JMS używa wartości AMQ.* podczas tworzenia kolejki dynamicznej. | Przedrostek używany do tworzenia nazwy kolejki dynamicznej IBM MQ . |
| TEMPTOPICPREFIX | Łańcuch | Dowolny niepusty łańcuch składający się tylko z poprawnych znaków dla łańcucha tematu IBM MQ | Podczas tworzenia tematów tymczasowych program JMS generuje łańcuch tematu w postaci "TEMP/*TEMPTOPICPREFIX*/*unikalny_identyfikator* " lub, jeśli ta właściwość ma wartość domyślną, tylko "TEMP/*unikalny_identyfikator*". Określenie niepustego parametru **TEMPTOPICPREFIX** umożliwia zdefiniowanie konkretnych kolejek modelowych na potrzeby tworzenia kolejek zarządzanych dla subskrybentów tematów tymczasowych utworzonych w ramach tego połączenia. |

*Tabela 66. Właściwości obiektu ConnectionFactory (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| transportType | Łańcuch | • **client**<br>• POWIĄZANIA<br>• BINDINGS_THEN_CLIENT (POWIĄZANIA, NASTĘPNIE KLIENT) | Określa, czy połączenie z menedżerem kolejek używa trybu klienta, czy trybu powiązań. Jeśli zostanie podana wartość `BINDINGS_THEN_CLIENT` , adapter zasobów najpierw próbuje nawiązać połączenie w trybie powiązań. Jeśli ta próba nawiązania połączenia nie powiedzie się, adapter zasobów podejmie próbę nawiązania połączenia w trybie klienta.<br>z/OS Jeśli specyfikacja aktywowania działająca w systemie WebSphere Application Server for z/OS została skonfigurowana pod kątem używania trybu transportowego `BINDINGS_THEN_CLIENT` , a wcześniej nawiązane połączenie zostało zerwane, wszystkie próby ponownego nawiązania połączenia przez specyfikację aktywowania najpierw podejmą próbę użycia trybu transportowego `BINDINGS` . Jeśli próba nawiązania połączenia w trybie transportu `BINDINGS` (Powiązania) nie powiedzie się, specyfikacja aktywowania próbuje następnie nawiązać połączenie w trybie transportu `CLIENT` . |
| Nazwa użytkownika | Łańcuch | • **Null**<br>• Nazwa użytkownika | Domyślna nazwa użytkownika, która ma być używana podczas tworzenia połączenia z menedżerem kolejek. |
| wildcardFormat | int | • CHAR-rozpoznaje tylko znaki wieloznaczne używane w brokerze w wersji 1<br>• TOPIC-rozpoznaje tylko znaki wieloznaczne na poziomie tematu, używane w brokerze w wersji 2 | Która wersja składni znaku wieloznacznego ma być używana. |

**Uwagi:**

1. Ważne informacje na temat używania wymaganej właściwości sslFipszawiera sekcja “Ograniczenia adaptera zasobów IBM MQ” na stronie 455.
2. Informacje na temat konfigurowania adaptera zasobów w taki sposób, aby mógł on znaleźć wyjście, zawiera sekcja “Konfigurowanie produktu IBM MQ classes for JMS do korzystania z wyjść kanałów” na stronie 290.

W poniższym przykładzie przedstawiono typowy zestaw właściwości obiektu ConnectionFactory :

```
channel:        SYSTEM.DEF.SVRCONN
hostName:       192.168.0.42
port:           1414
queueManager:   ExampleQM
transportType:  CLIENT
```

## Właściwości administrowanego obiektu docelowego

Serwer aplikacji używa właściwości administrowanego obiektu docelowego do utworzenia obiektu kolejki JMS lub obiektu tematu JMS .

Tabela 67 na stronie 501 zawiera listę właściwości wspólnych dla obiektu kolejki i obiektu tematu.

*Tabela 67. Właściwości wspólne dla obiektu kolejki i obiektu tematu*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| CCSID | Łańcuch | • **1208**<br>• Identyfikator kodowanego zestawu znaków obsługiwany przez wirtualną maszynę języka Java (JVM) | Identyfikator kodowanego zestawu znaków dla miejsca docelowego. |
| kodowanie | Łańcuch | • **native**<br>• Łańcuch składający się z trzech znaków:<br>– Pierwszy znak określa reprezentację binarnych liczb całkowitych:<br>- *N* oznacza normalne kodowanie.<br>- *R* oznacza kodowanie odwrotne.<br>– Drugi znak określa reprezentację upakowanych dziesiętnych liczb całkowitych:<br>- *N* oznacza normalne kodowanie.<br>- *R* oznacza kodowanie odwrotne.<br>– Trzeci znak określa reprezentację liczb zmiennopozycyjnych:<br>- *N* oznacza standardowe kodowanie IEEE.<br>- *R* oznacza odwrotne kodowanie IEEE.<br>- *3* oznacza kodowanie zSeries .<br>NATIVE jest odpowiednikiem łańcucha NNN. | Reprezentacja binarnych liczb całkowitych, upakowanych liczb dziesiętnych i liczb zmiennopozycyjnych dla miejsca docelowego. |
| Koniec ważności | Łańcuch | • **APP** -czas utraty ważności komunikatu jest określany przez producenta komunikatów.<br>• UNLIM-Komunikat nigdy nie traci ważności.<br>• 0-komunikat nigdy nie traci ważności.<br>• Dodatnia liczba całkowita reprezentująca czas utraty ważności komunikatu w milisekundach. | Czas utraty ważności komunikatu wysłanego do miejsca docelowego. |

*Tabela 67. Właściwości wspólne dla obiektu kolejki i obiektu tematu (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| FAILIFQUIESCE | Łańcuch | • **Prawda**<br>• Fałsz | Określa, czy próba uzyskania dostępu do miejsca docelowego nie powiedzie się, jeśli menedżer kolejek jest w stanie wyciszania. |

*Tabela 67. Właściwości wspólne dla obiektu kolejki i obiektu tematu (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| Styl messageBody | Łańcuch | • **Nieokreślone**<br>• JMS<br>• MQ | W kolejkach i tematach w systemie JMS można ustawić właściwość **`messageBodyStyle`** : UNSPECIFIED (wartość domyślna)<br>• Podczas wysyłania produkt IBM MQ classes for JMS generuje i dołącza nagłówek MQRFH2 w zależności od wartości właściwości WMQ_TARGET_CLIENT.<br>• Podczas odbierania produkt IBM MQ classes for JMS ustawia właściwości komunikatu JMS zgodnie z wartościami w MQRFH2, jeśli są obecne. Nagłówek MQRFH2 nie jest prezentowany jako część treści komunikatu produktu JMS .<br>JMS<br>• Podczas wysyłania produkt IBM MQ classes for JMS automatycznie generuje nagłówek MQRFH2 i dołącza nagłówek do komunikatu IBM MQ .<br>• Podczas odbierania produkt IBM MQ classes for JMS ustawia właściwości komunikatu JMS zgodnie z wartościami w MQRFH2, jeśli są obecne. Nagłówek MQRFH2 nie jest prezentowany jako część treści komunikatu produktu JMS .<br>MQ<br>• Podczas wysyłania IBM MQ classes for JMS nie generuj nagłówka MQRFH2.<br>• Podczas odbierania produkt IBM MQ classes for JMS przedstawia MQRFH2 jako część treści komunikatu JMS . |

*Tabela 67. Właściwości wspólne dla obiektu kolejki i obiektu tematu (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| trwałość | Łańcuch | • **APP** -trwałość komunikatu jest określana przez producenta komunikatów.<br>• QDEF-trwałość komunikatu jest określana przez atrybut **DefPersistence** kolejki IBM MQ .<br>• PERS-komunikat jest trwały.<br>• NON-Komunikat jest nietrwały.<br>• HIGH-Trwałość komunikatu jest określana przez atrybut **NonPersistentMessageClass** kolejki IBM MQ zgodnie z wyjaśnieniem w sekcji “Trwałe komunikaty JMS” na stronie 260. | Trwałość komunikatu wysłanego do miejsca docelowego. |
| priorytet | Łańcuch | • **APP** -priorytet komunikatu jest określany przez producenta komunikatu.<br>• QDEF-priorytet komunikatu jest określany przez atrybut **DefPriority** kolejki IBM MQ .<br>• Liczba całkowita z zakresu od 0, najniższy priorytet, do 9, najwyższy priorytet. | Priorytet komunikatu wysyłanego do miejsca docelowego. |
| PUTASYNCALLOWED | Łańcuch | • QUEUE-określa, czy asynchroniczne operacje umieszczania są dozwolone, odwołując się do definicji kolejki.<br>• TOPIC-Określenie, czy asynchroniczne operacje umieszczania są dozwolone, poprzez odwołanie się do definicji tematu.<br>• DESTINATION-określa, czy asynchroniczne operacje umieszczania są dozwolone, odwołując się do definicji kolejki lub tematu.<br>• DISABLED-asynchroniczne operacje umieszczania nie są dozwolone.<br>• ENABLED-asynchroniczne operacje put są dozwolone. | Określa, czy producenci komunikatów mogą używać asynchronicznych operacji umieszczania w celu wysyłania komunikatów do tego miejsca docelowego. |

*Tabela 67. Właściwości wspólne dla obiektu kolejki i obiektu tematu (kontynuacja)*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| READAHEADALLOWED | int | • **DESTINATION** -określa, czy odczyt z wyprzedzeniem jest dozwolony, odwołując się do definicji kolejki lub tematu.<br>• DISABLED-odczyt z wyprzedzeniem jest niedozwolony.<br>• ENABLED-odczyt z wyprzedzeniem jest dozwolony.<br>• QUEUE-określ, czy odczyt z wyprzedzeniem jest dozwolony, odwołując się do definicji kolejki.<br>• TOPIC-Określenie, czy odczyt z wyprzedzeniem jest dozwolony, poprzez odwołanie się do definicji tematu. | Określa, czy konsumenty komunikatów i przeglądarki kolejek mogą używać odczytu z wyprzedzeniem do pobierania nietrwałych komunikatów z miejsca docelowego do buforu wewnętrznego przed ich odebraniem. |
| receiveCCSID | int | • **0** -użyj maszyny JVM `Charset.defaultCharset`<br>• 1208- UTF-8<br>• Obsługiwany identyfikator kodowanego zestawu znaków | Właściwość miejsca docelowego, która ustawia docelowy identyfikator CCSID dla konwersji komunikatów menedżera kolejek. Wartość jest ignorowana, chyba że parametr **`receiveConversion`** ma wartość `QMGR`. |
| receiveConversion | Łańcuch | • **KOMUNIKAT KLIENTA**<br>• QMGR | Właściwość miejsca docelowego, która określa, czy konwersja danych będzie wykonywana przez menedżer kolejek. |
| targetClient | Łańcuch | • **JMS** -celem komunikatu jest aplikacja JMS .<br>• MQ -celem komunikatu jest aplikacja inna niżJMS IBM MQ . | Określa, czy celem komunikatu wysyłanego do miejsca docelowego jest aplikacja JMS . Komunikat z miejscem docelowym, który jest aplikacją JMS , zawiera nagłówek MQRFH2 . |

Tabela 68 na stronie 505 zawiera listę właściwości, które są specyficzne dla obiektu kolejki.

*Tabela 68. Właściwości specyficzne dla obiektu kolejki*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| baseQueueManagerName | Łańcuch | • **"" (pusty łańcuch)**<br>• Nazwa menedżera kolejek | Nazwa menedżera kolejek, który jest właścicielem bazowej kolejki produktu IBM MQ . |
| Nazwa baseQueue | Łańcuch | • **"" (pusty łańcuch)**<br>• Nazwa kolejki | Nazwa bazowej kolejki produktu IBM MQ . |

Tabela 69 na stronie 506 zawiera listę właściwości, które są specyficzne dla obiektu Topic.

*Tabela 69. Właściwości specyficzne dla obiektu Topic*

| Nazwa właściwości | Typ | Poprawne wartości (wartość domyślna pogrubiona) | Opis |
|---|---|---|---|
| baseTopicNazwa | Łańcuch | • **"" (pusty łańcuch)**<br>• Nazwa tematu | Nazwa tematu bazowego. |
| brokerCCDurSubQueue > | Łańcuch | • **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE**<br>• Nazwa kolejki | Nazwa kolejki, z której konsument połączenia odbiera komunikaty subskrypcji trwałej. |
| brokerDurSubQueue | Łańcuch | • **SYSTEM.JMS.D.SUBSCRIBER.QUEUE**<br>• Nazwa kolejki | Nazwa kolejki, z której trwały subskrybent tematu odbiera komunikaty. Więcej informacji na ten temat zawiera opis właściwości **BROKEDURRSUBQ** w dokumentacji IBM MQ Explorer . |
| Kolejka brokerPub | Łańcuch | • **Nie ustawiono**<br>• Nazwa kolejki | Nazwa kolejki, do której wysyłane są opublikowane komunikaty (kolejka strumienia). Wartość tej właściwości nadpisuje wartość właściwości **brokerPubQueue** obiektu ConnectionFactory . Jeśli jednak wartość tej właściwości nie zostanie ustawiona, zamiast niej zostanie użyta wartość właściwości **brokerPubQueue** obiektu ConnectionFactory . |
| brokerPubQueueManager | Łańcuch | • **"" (pusty łańcuch)**<br>• Nazwa menedżera kolejek | Nazwa menedżera kolejek będącego właścicielem kolejki, do której są wysyłane komunikaty opublikowane w temacie. |
| brokerVersion | Łańcuch | • **Nie ustawiono**<br>• 1<br>• 2 | Wersja używanego brokera. Wartość tej właściwości nadpisuje wartość właściwości **brokerVersion** obiektu ConnectionFactory . Jeśli jednak wartość tej właściwości nie zostanie ustawiona, zamiast niej zostanie użyta wartość właściwości **brokerVersion** obiektu ConnectionFactory . |

W poniższym przykładzie przedstawiono zestaw właściwości obiektu Queue:

```
expiry:              UNLIM
persistence:         QDEF
```

```
baseQueueManagerName: ExampleQM
baseQueueName:        SYSTEM.JMS.TEMPQ.MODEL
```

W poniższym przykładzie przedstawiono zestaw właściwości obiektu Topic:

```
expiry:               UNLIM
persistence:          NON
baseTopicName:        myTestTopic
```

**Zadania pokrewne**

Określenie, że w czasie wykonywania na kliencie MQI będą używane tylko CipherSpecs z certyfikatem FIPS.

Konfigurowanie zasobów JMS na serwerze WebSphere Application Server

**Odsyłacze pokrewne**

Standardy FIPS (Federal Information Processing Standards) dla AIX, Linux, and Windows

## *Konfigurowanie właściwości uzgadniania targetClientdla specyfikacji aktywowania*

Istnieje możliwość skonfigurowania właściwości **targetClientMatching** dla specyfikacji aktywowania w taki sposób, aby nagłówek MQRFH2 był dołączany do komunikatów odpowiedzi, gdy komunikaty żądań nie zawierają nagłówka MQRFH2 . Oznacza to, że wszystkie właściwości komunikatu definiowane przez aplikację w komunikacie odpowiedzi są dołączane podczas wysyłania komunikatu.

## O tym zadaniu

Jeśli aplikacja komponentu bean sterowanego komunikatami (MDB) konsumuje komunikaty, które nie zawierają nagłówka MQRFH2 , za pośrednictwem specyfikacji aktywowania adaptera zasobów JCA IBM MQ , a następnie wysyła komunikaty odpowiedzi do miejsca docelowego JMS utworzonego na podstawie pola JMSReplyTo komunikatu żądania, komunikaty odpowiedzi muszą zawierać nagłówek MQRFH2 , nawet jeśli komunikaty żądania nie są, w przeciwnym razie wszystkie właściwości komunikatu zdefiniowane przez aplikację w komunikacie odpowiedzi zostaną utracone.

Właściwość **targetClientMatching** określa, czy komunikat odpowiedzi wysyłany do kolejki identyfikowanej przez pole nagłówka JMSReplyTo komunikatu przychodzącego ma nagłówek MQRFH2 tylko wtedy, gdy komunikat przychodzący ma nagłówek MQRFH2 . Tę właściwość można skonfigurować dla specyfikacji aktywowania zarówno w produkcie WebSphere Application Server traditional , jak i w produkcie WebSphere Liberty.

Jeśli właściwość **targetClientMatching** zostanie ustawiona na wartość `false`, nagłówek MQRFH2 może zostać dołączony do komunikatu odpowiedzi wysyłanego do miejsca docelowego JMS utworzonego z nagłówka JMSReplyTo komunikatu żądania przychodzącego, który nie zawiera nagłówka MQRFH2. Jest to spowodowane tym, że właściwość **targetClient** w miejscu docelowym JMS jest ustawiona na wartość `0`, co oznacza, że komunikaty zawierają nagłówek MQRFH2 . Obecność nagłówka MQRFH2 w komunikacie wychodzącym pozwala na przechowywanie zdefiniowanych przez użytkownika właściwości komunikatu wysyłanego do kolejki IBM MQ .

Jeśli właściwość **targetClientMatching** jest ustawiona na wartość `true` , a komunikat żądania nie zawiera nagłówka MQRFH2 , nagłówek MQRFH2 nie jest dołączany do komunikatu odpowiedzi.

## Procedura

- W produkcie WebSphere Application Server traditionalużyj konsoli administracyjnej, aby zdefiniować właściwość **targetClientMatching** jako właściwość niestandardową w specyfikacji aktywowania IBM MQ :
  a) Na panelu nawigacyjnym kliknij opcję **Zasoby-> JMS-> Specyfikacje aktywowania**.
  b) Wybierz nazwę specyfikacji aktywowania, która ma zostać wyświetlona lub zmieniona.
  c) Kliknij opcję **Właściwości niestandardowe-> Nowa** , a następnie wprowadź szczegóły nowej właściwości niestandardowej.

Ustaw nazwę właściwości na `targetClientMatching`, typ na `java.lang.Boolean` i wartość na `false`.

- W pliku WebSphere Libertyokreśl właściwość **`targetClientMatching`** w definicji specyfikacji aktywowania w pliku `server.xml`.

  Na przykład:

```
<jmsActivationSpec id="SimpleMDBApplication/SimpleEchoMDB/SimpleEchoMDB">
<properties.wmqJms destinationRef="MDBRequestQ"
queueManager="MY_QMGR" transportType="BINDINGS" targetClientMatching="false"/>
<authData password="********" user="tom"/>
</jmsActivationSpec>
```

**Pojęcia pokrewne**

“Tworzenie miejsc docelowych w aplikacji JMS” na stronie 227
Zamiast pobierać miejsca docelowe jako obiekty administrowane z przestrzeni nazw JNDI ( Java Naming and Directory Interface), aplikacja JMS może używać sesji do dynamicznego tworzenia miejsc docelowych w czasie wykonywania. Aplikacja może używać identyfikatora URI (uniform resource identifier) do identyfikowania kolejki IBM MQ lub tematu oraz opcjonalnie do określania jednej lub większej liczby właściwości obiektu kolejki lub tematu.

“Konfigurowanie adaptera zasobów na potrzeby komunikacji wychodzącej” na stronie 487
Aby skonfigurować komunikację wychodzącą, należy zdefiniować właściwości obiektu ConnectionFactory i administrowanego obiektu docelowego.

## *IBM MQ wstrzymanie komponentu bean sterowanego komunikatami w WebSphere Liberty*

Właściwość **`maxSequentialDeliveryFailures`** specyfikacji aktywowania definiuje maksymalną liczbę sekwencyjnych niepowodzeń dostarczenia komunikatów do instancji komponentu bean sterowanego komunikatami (MDB), która jest tolerowana przez adapter zasobów przed wstrzymaniem komponentu MDB.

### Zanim rozpoczniesz

Należy pamiętać o zestawie zdarzeń, które mogą spowodować wstrzymanie komponentu MDB w produkcie WebSphere Liberty. Adapter zasobów traktuje jedną z poniższych informacji jako niepowodzenie dostarczenia komunikatu:

- Niesprawdzany wyjątek zgłaszany przez metodę **`onMessage`** komponentu MDB.
- `JMSException` występujący podczas przetwarzania adaptera zasobów przed dostarczeniem komunikatu do komponentu MDB.
- `JMSException` występujący podczas przetwarzania adaptera zasobów, po dostarczeniu komunikatu do komponentu MDB.
- Transakcja XA lub transakcja lokalna używana do pobierania wycofanego komunikatu.
- Na serwerze aplikacji nie ma dostępnego wątku, który dostarczył komunikat do komponentu MDB.

### O tym zadaniu

Wartością domyślną właściwości **`maxSequentialDeliveryFailures`** jest *-1*, co oznacza, że komponent MDB nigdy nie jest wstrzymywany. Każda inna wartość ujemna jest traktowana tak samo jak *-1*. Wartość:

- *0* oznacza, że komponent MDB wstrzymuje się przy pierwszym błędzie
- *1* oznacza, że komponent MDB wstrzymuje się z dwoma sekwencyjnymi błędami
- *2* oznacza, że komponent MDB wstrzymuje się z trzema sekwencyjnymi błędami itd.

Tę właściwość można skonfigurować dla specyfikacji aktywowania tylko w produkcie WebSphere Libertyoraz wtedy, gdy poziom produktu Liberty to 18.0.0.4lub nowszy.

**Ostrzeżenie:** Jeśli atrybut zostanie ustawiony na wartość inną niż domyślna w dowolnym środowisku serwera aplikacji innym niż Liberty, wartość ta zostanie zignorowana i w dzienniku zostanie zapisany komunikat ostrzegawczy.

Ponadto można zainstalować adapter zasobów IBM MQ w katalogu WebSphere Liberty jako ogólny adapter zasobów. Spowoduje to wyłączenie wszystkich możliwości integracji produktów IBM MQ i WebSphere Application Server oraz uniemożliwi adapterowi zasobów wykrycie, że jest on uruchomiony w produkcie Liberty. Dlatego ustawienie parametru **maxSequentialDeliveryFailures** na wartość większą lub równą *0* nie jest obsługiwane i powoduje wyświetlenie w dzienniku komunikatu ostrzegawczego.

## Procedura

- W pliku WebSphere Libertyokreśl właściwość **maxSequentialDeliveryFailures** w definicji specyfikacji aktywowania w pliku `server.xml`.

  Na przykład:

```
<jmsActivationSpec>
     <properties.wmqJms destinationRef="jndi/MDBQ"
                        transportType="BINDINGS"
                        queueManager="MQ21"
                        maxSequentialDeliveryFailures="1"/>
</jmsActivationSpec>
```

**Pojęcia pokrewne**

“Konfigurowanie adaptera zasobów na potrzeby komunikacji wychodzącej” na stronie 487
Aby skonfigurować komunikację wychodzącą, należy zdefiniować właściwości obiektu ConnectionFactory i administrowanego obiektu docelowego.

# Weryfikowanie instalacji adaptera zasobów

Program testu sprawdzającego instalację (IVT) dla adaptera zasobów IBM MQ jest dostarczany w postaci pliku EAR. Aby użyć programu, należy go wdrożyć i zdefiniować niektóre obiekty jako zasoby JCA .

## O tym zadaniu

Program testu weryfikującego instalację (IVT) jest dostarczany jako plik archiwum korporacyjnego (EAR) o nazwie `wmq.jakarta.jmsra.ivt.ear` (Jakarta Messaging 3.0) lub `wmq.jmsra.ivt.ear` (JMS 2.0). Ten plik jest instalowany razem z plikiem IBM MQ classes for JMS w tym samym katalogu co plik RAR adaptera zasobów IBM MQ , `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) lub `wmq.jmsra.rar` (JMS 2.0). Więcej informacji na temat miejsca, w którym te pliki są zainstalowane, zawiera sekcja “Instalowanie adaptera zasobów IBM MQ” na stronie 458.

Program IVT należy wdrożyć na serwerze aplikacji. Program IVT zawiera serwlet i komponent MDB, które testują, czy komunikat może zostać wysłany do kolejki IBM MQ i odebrany z tej kolejki. Za pomocą programu IVT można sprawdzić, czy adapter zasobów IBM MQ został poprawnie skonfigurowany do obsługi transakcji rozproszonych. Jeśli adapter zasobów IBM MQ jest wdrażany na serwerze aplikacji innym niżIBM , usługa IBM może poprosić o zademonstrowanie działania narzędzia do weryfikowania instalacji (IVT) w celu sprawdzenia, czy serwer aplikacji jest poprawnie skonfigurowany.

Przed uruchomieniem programu IVT należy zdefiniować obiekt ConnectionFactory , obiekt Queue i ewentualnie obiekt Activation Specification jako zasoby systemu JCA oraz upewnić się, że serwer aplikacji tworzy obiekty JMS na podstawie tych definicji i wiąże je w przestrzeni nazw systemu JNDI . Można wybrać właściwości obiektów, które mają być zgodne z ustawieniami hosta i portu własnego QueueManager, ale poniżej przedstawiono prosty przykład właściwości:

```
ConnectionFactory object:
channel:              SYSTEM.DEF.SVRCONN
hostName:             localhost
port:                 1550
queueManager:         QM1
transportType:        CLIENT
```

```
Queue object:
baseQueueManagerName: QM1
baseQueueName:        TEST.QUEUE
```

Mechanizm używany do definiowania obiektów ConnectionFactory, Queue i Activation Specification różni się w zależności od serwera aplikacji. Na przykład, aby ustawić te właściwości w pliku WebSphere Liberty, dodaj następujące wpisy do pliku `server.xml` serwera aplikacji:

```
JM 3.0 <!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jakarta.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

```
JMS 2.0 <!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

Domyślnie program IVT oczekuje powiązania obiektu ConnectionFactory w przestrzeni nazw JNDI o nazwie jms/ivt/IVTCF i obiektu Queue o nazwie jms/ivt/IVTQueue. Można użyć różnych nazw, ale w przeciwnym razie należy wprowadzić nazwy obiektów na stronie początkowej programu IVT i odpowiednio zmodyfikować plik EAR.

Po wdrożeniu programu IVT, gdy serwer aplikacji utworzy obiekty JMS i powiąże je w przestrzeni nazw JNDI , można uruchomić program IVT, wykonując następujące kroki.

## Procedura

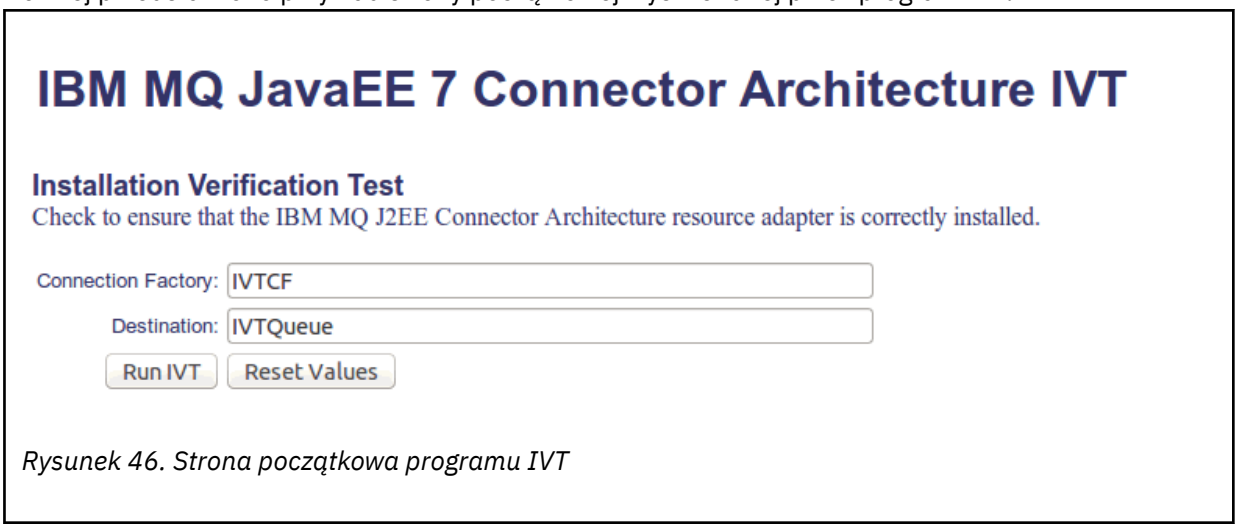
1. Uruchom program IVT, wprowadzając w przeglądarce WWW adres URL w następującym formacie:

   ```
   http://app_server_host: port/WMQ_IVT/
   ```

gdzie *host_serwera_aplikacji* jest adresem IP lub nazwą hosta systemu, na którym działa serwer aplikacji, a *port* jest numerem portu TCP, na którym nasłuchuje serwer aplikacji. Oto przykład:

```
http://localhost:9080/WMQ_IVT/
```

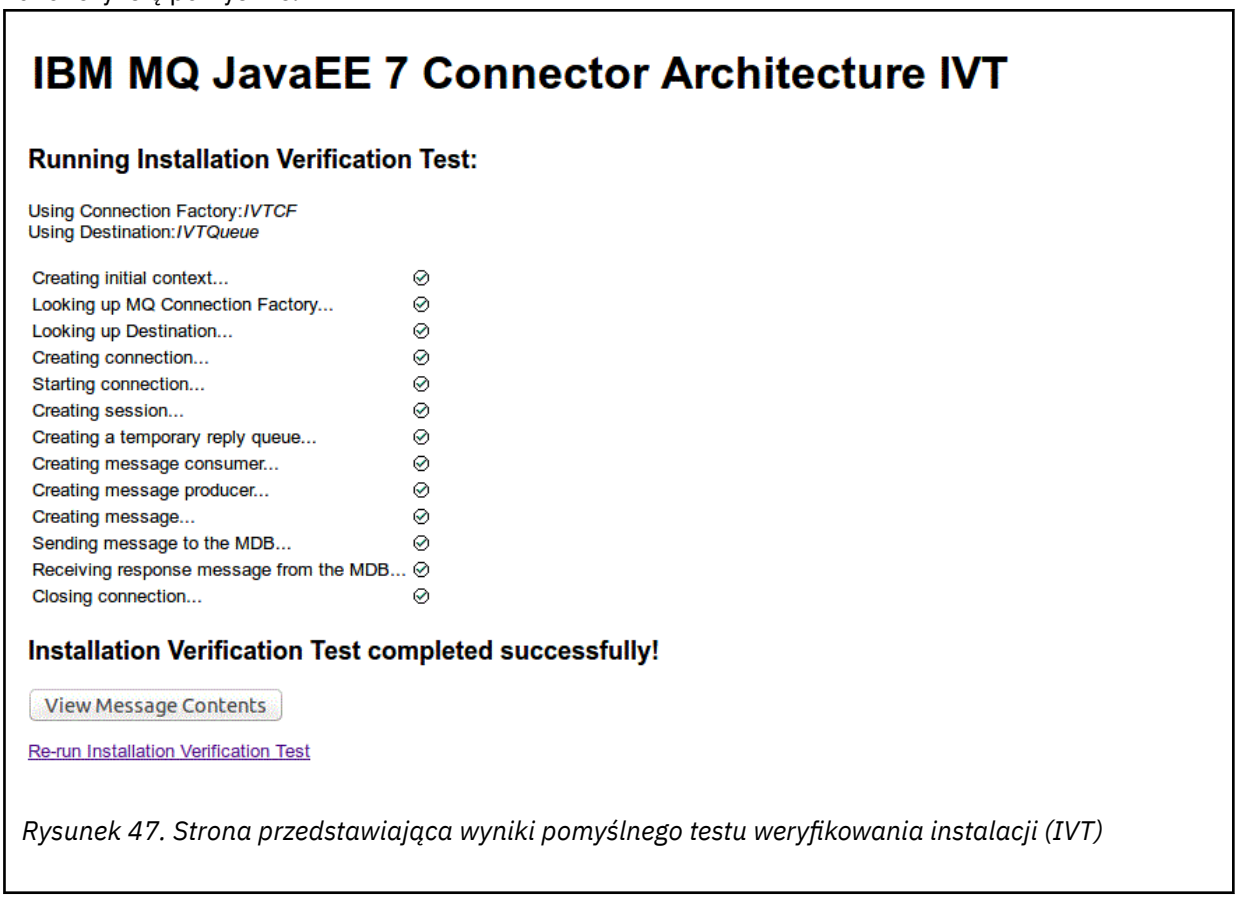
Poniżej przedstawiono przykład strony początkowej wyświetlanej przez program IVT.




*Rysunek 46. Strona początkowa programu IVT*

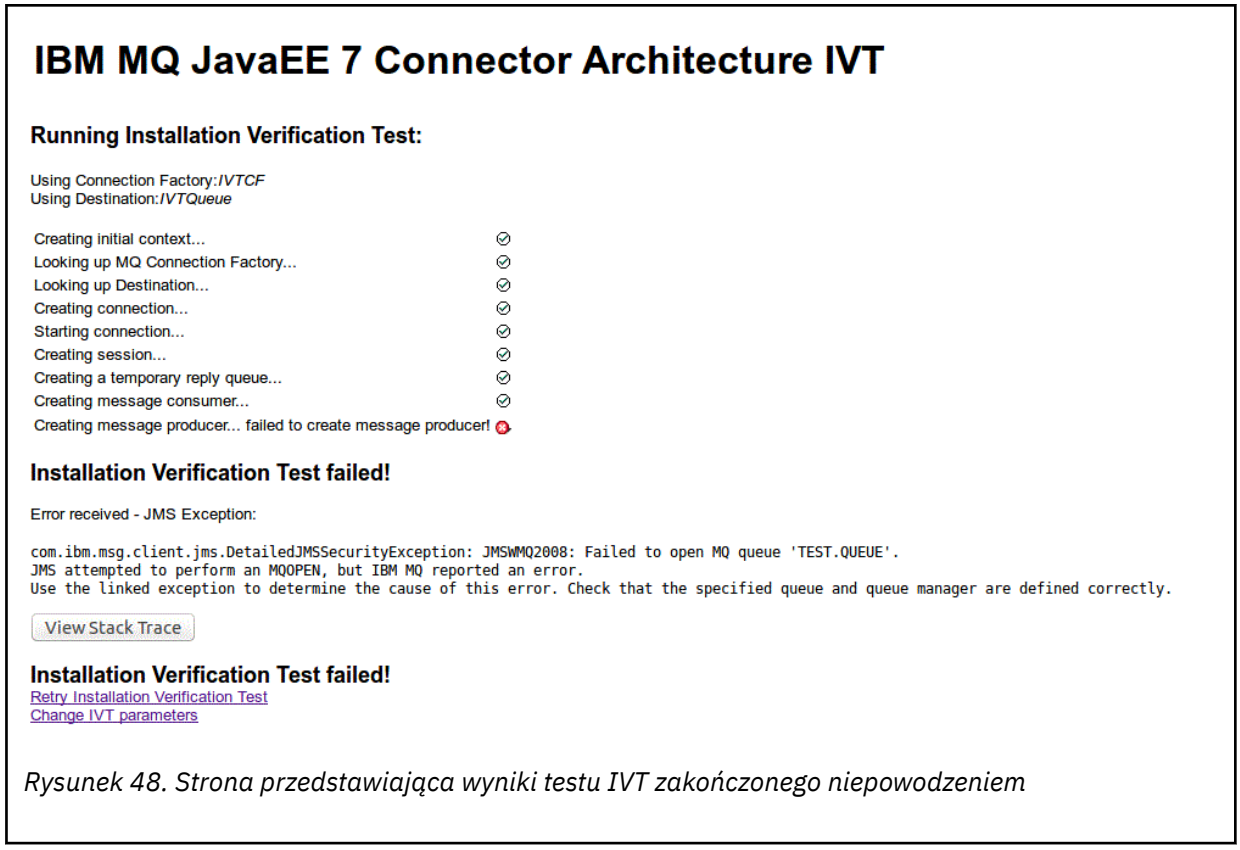
2. Aby uruchomić test, kliknij opcję **Uruchom narzędzie do weryfikowania instalacji (IVT)**.

   Poniżej przedstawiono przykład strony, która jest wyświetlana, jeśli test weryfikujący instalację zakończył się pomyślnie.




*Rysunek 47. Strona przedstawiająca wyniki pomyślnego testu weryfikowania instalacji (IVT)*

Poniżej przedstawiono przykład strony, która jest wyświetlana w przypadku niepowodzenia narzędzia do weryfikowania instalacji (IVT). Aby uzyskać więcej informacji na temat przyczyny niepowodzenia, kliknij opcję **Wyświetl stos wywołań**.




*Rysunek 48. Strona przedstawiająca wyniki testu IVT zakończonego niepowodzeniem*

## Windows Instalowanie i testowanie adaptera zasobów na serwerze GlassFish

Aby zainstalować adapter zasobów IBM MQ na serwerze GlassFish w systemie operacyjnym Windows , należy najpierw utworzyć i uruchomić domenę. Następnie można wdrożyć i skonfigurować adapter zasobów oraz wdrożyć i uruchomić aplikację testu sprawdzającego instalację (IVT).

### Zanim rozpoczniesz

- Te instrukcje dotyczą serwera GlassFish w wersji 4.
- Ta wersja serwera GlassFish Server nie obsługuje języka Jakarta EE.

### O tym zadaniu

W tej czynności przyjęto założenie, że użytkownik korzysta z działającego serwera aplikacji GlassFish Server oraz że zna standardowe czynności administracyjne związane z tym serwerem. W tej czynności przyjęto również założenie, że w systemie lokalnym jest zainstalowana baza danych IBM MQ i że użytkownik zna standardowe czynności administracyjne.

**Uwaga:** Aby wykonać następujące czynności, należy mieć działającą instalację produktu IBM MQ ze skonfigurowanymi następującymi obiektami:

- Menedżer kolejek o nazwie QM, który jest uruchomiony na porcie 1414 i używa kanału SYSTEM.DEF.SVRCONN, który łączy się przy użyciu transportu klienta.
- Kolejka o nazwie Q1.

### Procedura

1. Uruchom program powłoki GlassFish Server **asadmin** .

a) Otwórz wiersz komend systemu Windows i przejdź do katalogu *GlassFish*/bin , gdzie *GlassFish* jest katalogiem, w którym zainstalowano serwer GlassFish Server w wersji 4.

b) Wprowadź komendę **asadmin** w wierszu komend.

Komenda **asadmin** otwiera w wierszu komend program powłoki, który umożliwia utworzenie nowej domeny.

W systemie jest uruchomiony serwer GlassFish Server w wersji 4.

2. Utwórz, a następnie uruchom domenę.

a) Użyj komendy **create-domain** , podając port i nazwę domeny, aby utworzyć nową domenę. Wprowadź następującą komendę w wierszu komend:

```
create-domain --adminport port domain_name
```

gdzie *port* jest numerem portu, a *nazwa_domeny* jest nazwą, która ma być używana przez domenę.

**Uwaga:** Komenda **create-domain** ma wiele opcjonalnych parametrów, które są z nią powiązane. Jednak w przypadku tego zadania wymagany jest tylko parametr `-- adminport` . Więcej informacji na ten temat zawiera dokumentacja produktu GlassFish Server, wersja 4.

Jeśli podany port jest używany, zostanie wyświetlony następujący komunikat:

`Port dla nazwa_domeny port jest używany`

Jeśli podana nazwa domeny jest używana, zostanie wyświetlony komunikat informujący o tym, że podana nazwa jest już używana, a także lista wszystkich nazw domen, które są aktualnie niedostępne.

b) Po wyświetleniu zachęty do wprowadzenia nazwy użytkownika i hasła wprowadź referencje, które mają być używane do logowania się do serwera aplikacji za pośrednictwem przeglądarki WWW.

Jeśli komenda zakończy się pomyślnie, w wierszu komend zostanie wyświetlony komunikat z podsumowaniem tworzenia domeny, w tym komunikat `Command create-domain executed successfully.`

Pomyślnie utworzono domenę.

c) Uruchom domenę, wprowadzając następującą komendę w wierszu komend:

```
start-domain domain_name
```

gdzie *nazwa_domeny* jest poprzednio podaną nazwą domeny.

3. Użyj przeglądarki WWW, aby uzyskać dostęp do serwera aplikacji GlassFish .

a) Na pasku adresu przeglądarki WWW wprowadź następującą komendę:

```
localhost:port
```

gdzie *port* jest portem określonym wcześniej podczas tworzenia domeny.

Zostanie wyświetlona konsola GlassFish .

b) Po załadowaniu konsoli GlassFish i wyświetleniu pytania o nazwę użytkownika i hasło, wprowadź referencje podane w kroku 2b.

4. Prześlij adapter zasobów do serwera GlassFish Server 4.

a) Na pasku narzędzi **Wspólne zadania** wybierz pozycję menu **Aplikacje** , aby wyświetlić stronę **Aplikacje** .

b) Kliknij przycisk **Wdróż** , aby otworzyć stronę **Wdrażanie aplikacji lub modułów** .

c) Kliknij przycisk **Przeglądaj** , a następnie przejdź do położenia pliku `wmq.jmsra.rar` . Wybierz plik, a następnie kliknij przycisk **OK**.

5. Utwórz pulę połączeń.

a) Na pasku narzędzi, w sekcji **Zasoby**, wybierz pozycję menu **Konektory** .

b) Następnie wybierz pozycję menu **Pule połączeń konektora** , aby otworzyć stronę **Pule połączeń konektora** .

c) Kliknij opcję **Nowa** , aby otworzyć stronę **Nowa pula połączeń konektora (krok 1 z 2)** .

d) Na stronie **Nowa pula połączeń konektora (krok 1 z 2)** wprowadź nazwę puli jako jms/ivt/IVTCF-Connection-Pool w polu **Pool Name** (Nazwa puli).

e) W polu **Adapter zasobów** wybierz opcję `wmq.jmsra`.

f) W polu **Connection Definition** (Definicja połączenia) wpisz `javax.jms.ConnectionFactory`.

g) Wybierz opcję **Dalej**, a następnie kliknij przycisk **Zakończ**.

6. Utwórz zasoby konektora.

a) Na pasku narzędzi w menu **Konektory** wybierz opcję **Zasób konektora** , aby otworzyć stronę **Zasoby konektora** .

b) Wybierz opcję **Nowy**, aby otworzyć stronę **Nowy zasób konektora** .

c) W polu **Nazwa JNDI** wprowadź wartość `IVTCF`.

d) W polu **Pool Name** (Nazwa puli) wpisz `jms/ivt/IVTCF-Connection-Pool`.

e) Pozostaw wszystkie pozostałe pola puste.

f) Dla każdej z następujących par właściwość/wartość kliknij opcję **Dodaj właściwość**i wprowadź nazwę właściwości oraz wartość, jak pokazano w poniższym przykładzie:

- nazwa: host; wartość: localhost
- nazwa: port; wartość 1414
- nazwa: kanał; wartość: SYSTEM.DEF.SVRCONN DEF.SVRCONN
- nazwa: queueManager; wartość: QM
- nazwa: transportType; wartość: KLIENT

**Uwaga:** Upewnij się, że używane są poprawne wartości dla własnych ustawień konfiguracyjnych, które mogą być inne niż przedstawione w tym przykładzie.

g) Na pasku narzędzi w sekcji **Konektory**wybierz pozycję menu **Zasoby obiektów administracyjnych** , aby otworzyć stronę **Zasoby obiektów administracyjnych** .

h) Na stronie **Zasoby obiektów administracyjnych** kliknij opcję **Nowy** , aby otworzyć stronę **Nowy zasób obiektów administracyjnych** .

i) W polu **Nazwa JNDI** wprowadź wartość `IVTQueue`.

j) W polu **Resource Adapter** (Adapter zasobów) wpisz `wmq.jmsra`.

k) W polu **Typ zasobu** wpisz `javax.jms.Queue`.

l) Pozostaw pole **Nazwa klasy** bez wprowadzania w nim danych.

m) Dla każdej z następujących par właściwość/wartość kliknij opcję **Dodaj właściwość**i wprowadź nazwę właściwości oraz wartość, jak pokazano w poniższym przykładzie:

- nazwa: nazwa; wartość: IVTQueue
- nazwa: baseQueueManagerName; wartość QM
- nazwa: baseQueueNazwa; wartość: Q1

**Uwaga:** Upewnij się, że używane są poprawne wartości dla własnych ustawień konfiguracyjnych, które mogą być inne niż przedstawione w tym przykładzie.

n) Kliknij przycisk **OK**.

o) Zaznacz pole wyboru **Włączone** , a następnie kliknij opcję **Włącz**.

7. Wdróż plik EAR `wmq.jmsra.ivt.ear` na serwerze GlassFish .

a) Kliknij opcję **Aplikacje** na pasku narzędzi, aby wyświetlić stronę **Aplikacje** .

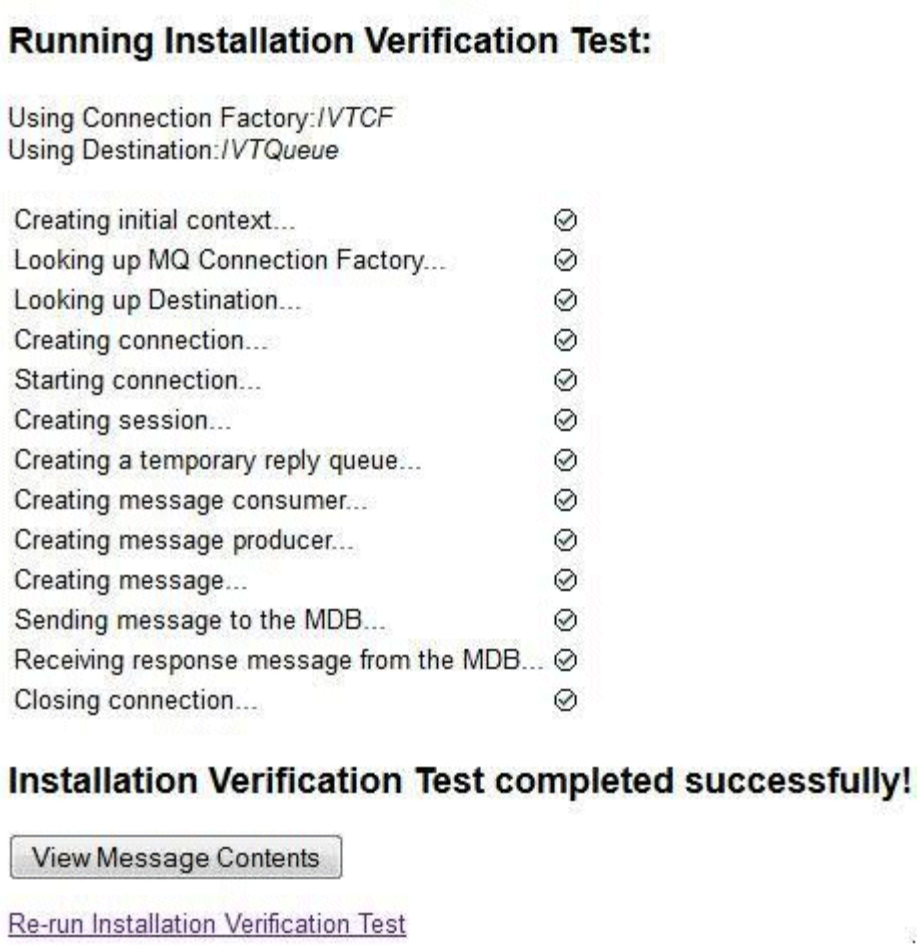
b) Kliknij opcję **Wdróż** , aby dodać aplikację IVT.

c) W polu **Położenie** przejdź do katalogu `wmq.jmsra.ivt.ear`i wybierz go.

d) W polu **Virtual Servers** wybierz opcję **serwer**, a następnie kliknij przycisk **OK**.

8. Uruchom program IVT.
   a) Kliknij opcję **Aplikacje** na pasku narzędzi, aby wyświetlić stronę **Aplikacje** .
   b) Kliknij ikonę `wmq.jmsra.ivt` w tabeli Wdrożone aplikacje.
   c) Kliknij przycisk **Uruchom** w tabeli Moduły i komponenty.
   d) Wybierz odsyłacz http:.
   e) Kliknij opcję **Uruchom narzędzie do weryfikowania instalacji (IVT)**.

   Program IVT został uruchomiony, a w przypadku powodzenia wyświetlane są następujące dane wyjściowe:




*Rysunek 49. Pomyślne dane wyjściowe narzędzia IVT*

## Instalowanie i testowanie adaptera zasobów w środowisku WildFly

Jeśli adapter zasobów IBM MQ jest instalowany w wersji WildFly V10, należy najpierw wprowadzić zmiany w pliku konfiguracyjnym, aby dodać definicję podsystemu dla adaptera zasobów IBM MQ . Następnie można wdrożyć adapter zasobów i przetestować go, instalując i uruchamiając aplikację testu sprawdzającego instalację (IVT).

### Zanim rozpoczniesz

- Instrukcje te dotyczą WildFly V10.
- Ta wersja środowiska WildFly nie obsługuje języka Jakarta EE.

## O tym zadaniu

W tej czynności przyjęto założenie, że użytkownik korzysta z działającego serwera aplikacji WildFly oraz że zna standardowe czynności administracyjne związane z tym serwerem. W tej czynności przyjęto również założenie, że użytkownik ma instalację produktu IBM MQ i zna standardowe czynności administracyjne.

## Procedura

1. Utwórz menedżer kolejek produktu IBM MQ o nazwie ExampleQMi skonfiguruj go zgodnie z opisem w sekcji “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym” na stronie 1101.

   Podczas konfigurowania menedżera kolejek należy zwrócić uwagę na następujące kwestie:

   - Program nasłuchujący musi być uruchomiony na porcie 1414.
   - Kanał, który ma być używany, nosi nazwę SYSTEM.DEF.SVRCONN.
   - Kolejka używana przez aplikację IVT nosi nazwę TEST.QUEUE.

   Kolejka modelowa SYSTEM.DEFAULT.MODEL.QUEUE .
2. Zmodyfikuj plik konfiguracyjny *WildFly_Home*/standalone/configuration/standalone-full.xml i dodaj następujący podsystem:

```
<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. Wdróż adapter zasobów na serwerze, kopiując plik `wmq.jmsra.rar` do katalogu *`WildFly_Home`*`/standalone/deployments`.
4. Wdróż aplikację IVT, kopiując plik `wmq.jmsra.ivt.ear` do katalogu *`WildFly_Home`*`/standalone/deployments`.
5. Uruchom serwer aplikacji, wywołując wiersz komend, przechodząc do katalogu *`WildFly_Home`*`/bin` i uruchamiając komendę:

   ```
   standalone.bat -c standalone-full.xml
   ```

6. Uruchom aplikację IVT.

   Więcej informacji na ten temat zawiera “Weryfikowanie instalacji adaptera zasobów” na stronie 509. W przypadku WildFlydomyślnym URL jest `http://localhost:8080/WMQ_IVT/`.

# Używanie produktów IBM MQ i WebSphere Application Server razem

Za pośrednictwem dostawcy przesyłania komunikatów IBM MQ w produkcie WebSphere Application Serveraplikacje przesyłania komunikatów produktu Java Message Service (JMS) mogą używać systemu IBM MQ jako zewnętrznego dostawcy zasobów przesyłania komunikatów produktu JMS .

## O tym zadaniu

Aplikacje napisane w języku Java i działające w systemie WebSphere Application Server mogą używać specyfikacji Java Message Service (JMS) do przesyłania komunikatów. Przesyłanie komunikatów w tym środowisku może być udostępniane przez menedżer kolejek produktu IBM MQ .

Zaletą używania menedżera kolejek produktu IBM MQ jest to, że łączące się aplikacje produktu JMS mogą w pełni uczestniczyć w funkcjonalności sieci produktu IBM MQ , co umożliwia aplikacjom wymianę komunikatów z menedżerami kolejek działającymi na wielu platformach.

Aplikacje mogą używać *transportu klienta* lub *transportu powiązań* dla obiektu fabryki połączeń kolejki. W przypadku transportu powiązań menedżer kolejek musi istnieć lokalnie względem aplikacji, która wymaga połączenia.

Domyślnie komunikaty JMS przechowywane w kolejkach systemu IBM MQ używają nagłówka MQRFH2 do przechowywania niektórych informacji z nagłówka komunikatu JMS . Wiele wcześniejszych aplikacji IBM MQ nie może przetwarzać komunikatów z tymi nagłówkami i wymaga własnych nagłówków charakterystycznych, na przykład MQCIH dla CICS Bridge lub MQWIH dla IBM MQ Workflow. Więcej informacji na temat tych specjalnych zagadnień zawiera sekcja Odwzorowywanie komunikatów produktu JMS na komunikaty produktu IBM MQ.

**Zadania pokrewne**

Konfigurowanie zasobów JMS w produkcie WebSphere Application Server

Konfigurowanie serwera aplikacji pod kątem używania najnowszego poziomu konserwacyjnego adaptera zasobów

# Używanie produktu WebSphere Application Server z produktem IBM MQ

Wartości IBM MQ i IBM MQ for z/OS mogą być używane z domyślnym dostawcą przesyłania komunikatów dołączonym do produktu WebSphere Application Serverlub jako alternatywa dla tego dostawcy.

Dostawca przesyłania komunikatów produktu IBM MQ jest instalowany jako część produktu WebSphere Application Server. Dotyczy to wersji adaptera zasobów IBM MQ oraz funkcji rozszerzonego klienta transakcyjnego IBM MQ , która umożliwia menedżerowi kolejek uczestniczenie w transakcjach XA

zarządzanych przez serwer aplikacji. Przy użyciu adaptera zasobów można skonfigurować komponenty bean sterowane komunikatami tak, aby używały specyfikacji aktywowania lub portów nasłuchiwania.

Aby serwer aplikacji był obsługiwany, program testowy instalacji adaptera zasobów IBM MQ musi zostać wdrożony na serwerze aplikacji i pomyślnie uruchomiony. Po pomyślnym uruchomieniu programu testowego instalacji adaptera zasobów IBM MQ adapter zasobów IBM MQ może nawiązać połączenie z dowolnym obsługiwanym menedżerem kolejek produktu IBM MQ .

## JMS połączenia z WebSphere Application Server do IBM MQ

Przed rozważeniem poziomów produktu IBM MQ , które mogą być używane z produktem WebSphere Application Server, należy zrozumieć, w jaki sposób aplikacje produktu Java Message Service (JMS) działające na serwerze aplikacji mogą nawiązywać połączenia z menedżerami kolejek systemu IBM MQ .

Aplikacje JMS , które wymagają dostępu do zasobów menedżera kolejek produktu IBM MQ , mogą to zrobić za pomocą jednego z następujących typów transportu:

**POWIĄZANIA**
Tego transportu można użyć, jeśli serwer aplikacji i menedżer kolejek są zainstalowane na tym samym komputerze i w tym samym obrazie systemu operacyjnego. Jeśli używany jest tryb BINDINGS, cała komunikacja między dwoma produktami jest realizowana za pomocą komunikacji międzyprocesowej (IPC).

Dostawca przesyłania komunikatów produktu IBM MQ nie zawiera bibliotek rodzimych wymaganych do nawiązania połączenia z menedżerem kolejek produktu IBM MQ w trybie BINDINGS. Aby można było używać połączenia w trybie BINDINGS, produkt IBM MQ musi być zainstalowany na tym samym komputerze, co serwer aplikacji, a ścieżka rodzimej biblioteki adaptera zasobów musi być skonfigurowana tak, aby wskazywała katalog IBM MQ , w którym znajdują się te biblioteki. Więcej informacji na ten temat zawiera dokumentacja produktu WebSphere Application Server :

- W przypadku systemu WebSphere Application Server traditionalnależy zapoznać się z sekcją Konfigurowanie dostawcy przesyłania komunikatów produktu IBM MQ z bibliotekami rodzimymi.
- W przypadku serwera WebSphere Libertynależy zapoznać się z sekcją Wdrażanie aplikacji JMS na serwerze Liberty w celu użycia dostawcy przesyłania komunikatów produktu IBM MQ.

z/OS Aby w systemie z/OSpołączyć fabrykę połączeń WebSphere Application Server z menedżerem kolejek IBM MQ w trybie powiązań, należy określić poprawne biblioteki IBM MQ w konkatenacji STEPLIB WebSphere Application Server . Więcej informacji na ten temat zawiera sekcja dotycząca bibliotek IBM MQ i sekcja WebSphere Application Server dla biblioteki z/OS STEPLIB w dokumentacji produktu WebSphere Application Server .

**KLIENT**
Transport klienta używa protokołu TCP/IP do komunikacji między systemami WebSphere Application Server i IBM MQ. Jeśli serwer aplikacji i menedżer kolejek znajdują się na różnych komputerach, tryb CLIENT może być używany również wtedy, gdy oba produkty są zainstalowane na tym samym komputerze i w tym samym obrazie systemu operacyjnego.

Aplikacje JMS mogą również określać typ transportu BINDINGS_THEN_CLIENT. Jeśli używany jest ten typ transportu, aplikacja początkowo podejmie próbę nawiązania połączenia z menedżerem kolejek przy użyciu trybu BINDINGS-jeśli nie jest to możliwe, spróbuje użyć transportu CLIENT.

## Informacje o tym, która wersja adaptera zasobów IBM MQ jest zainstalowana w katalogu WebSphere Application Server .

Informacje o tym, która wersja adaptera zasobów IBM MQ jest zainstalowana w produkcie WebSphere Application Server, zawiera nota techniczna Która wersja adaptera zasobów produktu WebSphere MQ (RA) jest dostarczana z serwerem WebSphere Application Server?.

Aby określić poziom adaptera zasobów, który jest obecnie używany przez produkt WebSphere Application Server , można użyć następujących komend Jython i JACL:

**Jython**

```
wmqInfoMBeansUnsplit = AdminControl.queryNames("WebSphere:type=WMQInfo,*")
wmqInfoMBeansSplit = AdminUtilities.convertToList(wmqInfoMBeansUnsplit)
for wmqInfoMBean in wmqInfoMBeansSplit: print wmqInfoMBean; print
AdminControl.invoke(wmqInfoMBean, 'getInfo', '')
```

**Uwaga:** Po wprowadzeniu tej komendy należy dwukrotnie kliknąć przycisk **Wróć** , aby ją uruchomić.

**JACL**

```
set wmqInfoMBeans [$AdminControl queryNames WebSphere:type=WMQInfo,*]
foreach wmqInfoMBean $wmqInfoMBeans {
puts $wmqInfoMBean;
puts [$AdminControl invoke $wmqInfoMBean getInfo [] []]
}
```

## Aktualizowanie adaptera zasobów

Aktualizacje adaptera zasobów IBM MQ zainstalowanego z serwerem aplikacji są dołączane do pakietów poprawek produktu WebSphere Application Server . Aktualizowanie adaptera zasobów IBM MQ przy użyciu opcji **Aktualizuj adapter zasobów ...** Narzędzie w Konsoli administracyjnej produktu WebSphere Application Server nie jest zalecane, ponieważ spowoduje to, że aktualizacje udostępnione w pakietach poprawek produktu WebSphere Application Server nie będą miały zastosowania.

## MQ_INSTALL_ROOT, zmienna

W katalogu WebSphere Application Server 7.0zmienna MQ_INSTALL_ROOT jest używana tylko do znajdowania bibliotek rodzimych i jest nadpisywana przez dowolną ścieżkę do bibliotek rodzimych skonfigurowaną w adapterze zasobów.

## Nawiązywanie połączenia z WebSphere Application Server do IBM MQ

**Ostrzeżenie:**

1. Każda obsługiwana wersja produktu WebSphere Application Server może używać dołączonego do niego adaptera zasobów IBM MQ do nawiązywania połączenia z dowolną obsługiwaną wersją produktu IBM MQ.
2. Jeśli używany jest tryb powiązań, niektóre biblioteki w produkcie WebSphere Application Server muszą być zgodne z wersją menedżera kolejek, z którym nawiązywane jest połączenie:
   - Produkt WebSphere Application Server musi być skonfigurowany do ładowania bibliotek rodzimych dostarczanych z produktem IBM MQ 9.3. Więcej informacji zawiera sekcja “Konfigurowanie bibliotek JNI ( Java Native Interface)” na stronie 100.
   - z/OS W systemie z/OSnależy określić poprawne biblioteki IBM MQ w konkatenacji STEPLIB WebSphere Application Server .

     Szczegółowe informacje o potrzebnych bibliotekach IBM MQ można znaleźć w sekcji IBM MQ oraz w publikacji WebSphere Application Server for z/OS STEPLIB .

     Jeśli na liście LINKLIST (LINKLST) znajdują się biblioteki dla jednej wersji systemu IBM MQ , można połączyć się z inną wersją systemu IBM MQ , nadpisując biblioteki przy użyciu biblioteki STEPLIB.
3. Wersja adaptera zasobów IBM MQ jest niezależna od wersji rodzimej (współużytkowanej) biblioteki udostępnianej przez instalację menedżera kolejek.

   Na przykład produkt WebSphere Application Server 8.5z adapterem zasobów IBM MQ 8.0 może nadal zarządzać połączeniem powiązań z menedżerem kolejek produktu IBM MQ 9.0 przy użyciu bibliotek rodzimych IBM MQ 9.0 .

Więcej informacji na ten temat zawiera “IBM MQ deklaracja obsługi adaptera zasobów” na stronie 452.

Typy transportu BINDINGS i CLIENT mogą być używane do nawiązywania połączeń z produktem IBM MQ z dowolnej wersji produktu WebSphere Application Server. W przypadku typu transportu BINDINGS mają zastosowanie następujące ograniczenia:

- Produkt IBM MQ musi być zainstalowany na tym samym komputerze, co serwer aplikacji.
- Produkt WebSphere Application Server musi być skonfigurowany do ładowania bibliotek rodzimych dostarczanych z produktem IBM MQ.
- z/OS W systemie z/OS, aby połączyć fabrykę połączeń WebSphere Application Server z menedżerem kolejek IBM MQ w trybie powiązań, należy określić poprawne biblioteki IBM MQ w konkatenacji STEPLIB WebSphere Application Server .

W poniższej tabeli przedstawiono wersje produktu WebSphere Application Server , w których można uruchamiać poszczególne wersje adaptera zasobów IBM MQ .

*Tabela 70. Odwzorowanie wersji produktu WebSphere Application Server na wersje adaptera zasobów produktu IBM MQ .*

| **Wersja adaptera zasobów IBM MQ** | **W której wersji produktu WebSphere Application Server można uruchomić tę wersję adaptera zasobów?** |
|---|---|
| IBM MQ 9.0 i nowsze | Adapter zasobów może działać w:<br>• Dowolna wersja produktu WebSphere Libertyzgodna z produktem Java EE 7 .<br>• WebSphere Application Server traditional 9.0 |
| IBM MQ 8.0 | Adapter zasobów może działać w dowolnej wersji systemu WebSphere Liberty zgodnej z Java EE 7 .<br>Adapter zasobów IBM MQ 8.0 nie jest obsługiwany do uruchamiania w systemie WebSphere Application Server traditional. Adapter zasobów już zainstalowany w produkcie WebSphere Application Server traditional powinien być używany do nawiązywania połączeń z menedżerami kolejek systemu IBM MQ 8.0 . |

**Pojęcia pokrewne**

“IBM MQ deklaracja obsługi adaptera zasobów” na stronie 452

Produkt Adapter zasobów IBM MQ , którego należy użyć do komunikacji między aplikacją i menedżerem kolejek, zależy od tego, czy używany jest interfejs API języka Jakarta Messaging 3.0 , czy interfejs API języka JMS 2.0 .

**Informacje pokrewne**

Wymagania systemowe produktu IBM MQ

## Określanie liczby połączeń TCP/IP tworzonych z zakresu od WebSphere Application Server do IBM MQ

Korzystając z funkcji współużytkowania konwersacji, wiele konwersacji może współużytkować instancje kanału MQI, co jest również nazywane połączeniem TCP/IP.

### O tym zadaniu

Aplikacje działające w obrębie produktów WebSphere Application Server 7 i WebSphere Application Server 8, które używają trybu normalnego dostawcy usługi przesyłania komunikatów produktu IBM MQ , będą automatycznie korzystać z tej funkcji. Oznacza to, że wiele aplikacji działających w tej samej

instancji serwera aplikacji, które łączą się z tym samym menedżerem kolejek produktu IBM MQ , może współużytkować tę samą instancję kanału.

Liczba konwersacji, które mogą być współużytkowane w pojedynczej instancji kanału, jest określana przez IBM MQ właściwość kanału **SHARECNV**. Wartością domyślną tej właściwości dla kanałów połączenia z serwerem jest 10.

Sprawdzając liczbę konwersacji, które są tworzone przez WebSphere Application Server 7 i WebSphere Application Server 8, można określić liczbę utworzonych instancji kanału.

Więcej informacji na temat trybu dostawcy przesyłania komunikatów produktu IBM MQ zawiera sekcja Tryb normalny PROVIDERVERSION.

**Pojęcia pokrewne**

Korzystanie z współużytkowania konwersacji
W środowisku, w którym współużytkowanie konwersacji jest dozwolone, konwersacje mogą współużytkować instancję kanału MQI.

“Współużytkowanie połączenia TCP/IP w programie IBM MQ classes for JMS” na stronie 326
Aby współużytkować pojedyncze połączenie TCP/IP, można utworzyć wiele instancji kanału MQI.

### *Fabryki połączeń usługi JMS*

Aplikacje działające w produkcie WebSphere Application Server, które używają fabryki połączeń dostawcy przesyłania komunikatów produktu IBM MQ do tworzenia połączeń i sesji, mają aktywne konwersacje dla każdego połączenia JMS utworzonego z fabryki połączeń oraz dla każdej sesji JMS utworzonej z połączenia JMS .

## Jedna konwersacja dla każdego połączenia JMS utworzonego z fabryki połączeń

Każda fabryka połączeń JMS ma powiązaną pulę połączeń, podzieloną na dwie sekcje: wolną i aktywną. Obie pule są początkowo puste.

Gdy aplikacja tworzy połączenie JMS z fabryki połączeń, produkt WebSphere Application Server sprawdza, czy w wolnej puli istnieje połączenie JMS . Jeśli istnieje, jest on przenoszony do aktywnej puli i podawany do aplikacji. W przeciwnym razie zostanie utworzone nowe połączenie JMS , umieszczone w aktywnej puli i zwrócone do aplikacji. Maksymalna liczba połączeń, które można utworzyć z fabryki połączeń, jest określona przez właściwość puli połączeń fabryki połączeń **Maximum connections**. Wartością domyślną tej właściwości jest 10.

Po zakończeniu działania aplikacji z połączeniem JMS i zamknięciu go, połączenie jest przenoszone z aktywnej puli do wolnej puli, w której jest dostępne do ponownego wykorzystania. Właściwość puli połączeń **Unused timeout** definiuje, jak długo połączenie JMS może pozostawać w puli wolnych połączeń, zanim zostanie rozłączone. Wartością domyślną tej właściwości jest 1800 sekund (30 minut).

Gdy połączenie JMS jest tworzone po raz pierwszy, rozpoczyna się konwersacja między WebSphere Application Server i IBM MQ . Konwersacja pozostaje aktywna do momentu zamknięcia połączenia po przekroczeniu wartości właściwości **Unused timeout** dla wolnej puli.

## Jedna konwersacja dla każdej sesji JMS utworzonej z połączenia JMS

Każde połączenie JMS utworzone na podstawie fabryki połączeń dostawcy przesyłania komunikatów produktu IBM MQ ma powiązaną pulę sesji produktu JMS . Pule sesji działają w ten sam sposób, co pule połączeń. Maksymalna liczba JMS sesji, które można utworzyć z pojedynczego połączenia JMS , jest określana przez właściwość puli sesji fabryki połączeń **Maximum connections**. Wartością domyślną tej właściwości jest 10.

Konwersacja rozpoczyna się po pierwszym utworzeniu sesji JMS . Konwersacja pozostaje aktywna do momentu zamknięcia sesji JMS , ponieważ pozostaje w wolnej puli przez czas dłuższy niż wartość właściwości **Unused timeout** dla puli sesji.

## Obliczanie wartości właściwości SHARECNV

Maksymalną liczbę konwersacji z pojedynczej fabryki połączeń do programu IBM MQ można obliczyć za pomocą następującej formuły:

```
Maximum number of conversations =
    connection Pool Maximum Connections +
   (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

Liczba instancji kanału, które zostaną utworzone w celu umożliwienia takiej liczby konwersacji, może zostać opracowana przy użyciu następującego obliczenia:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Każda reszta z tego obliczenia może zostać zaokrąglona w górę.

W przypadku prostej fabryki połączeń, która używa wartości domyślnej dla właściwości puli połączeń **Maximum connections** i puli sesji **Maximum connections** , maksymalna liczba konwersacji, które mogą istnieć między produktami WebSphere Application Server i IBM MQ dla tej fabryki połączeń, wynosi:

```
Maximum number of conversations =
   connection Pool Maximum Connections +
  (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

Na przykład:

```
        = 10 + (10 * 10)
        = 10 + 100
        = 110
```

Jeśli ta fabryka połączeń nawiązuje połączenie z produktem IBM MQ przy użyciu kanału z właściwością **SHARECNV** ustawioną na wartość 10, to maksymalna liczba instancji kanału, które zostaną utworzone dla tej fabryki połączeń, wynosi:

```
Maximum number of channel instances = Maximum number of conversations / SHARECNV for the
channel being used
```

Na przykład:

```
        = 110 / 10
        = 11 (rounded up to nearest connection)
```

### *Specyfikacje aktywowania*

Aplikacje komponentu bean sterowanego komunikatami, które są skonfigurowane do używania specyfikacji aktywowania, mają aktywne konwersacje dla specyfikacji aktywowania w celu monitorowania miejsca docelowego produktu JMS oraz dla każdej sesji serwera używanej do uruchamiania instancji komponentu bean sterowanego komunikatami w celu przetwarzania komunikatów.

Następujące konwersacje są aktywne dla aplikacji komponentów bean sterowanych komunikatami, które są skonfigurowane do używania specyfikacji aktywowania:

- Jedna konwersacja dla specyfikacji aktywowania służąca do monitorowania miejsca docelowego produktu JMS pod kątem odpowiednich komunikatów. Ta konwersacja jest uruchamiana natychmiast po uruchomieniu specyfikacji aktywowania i pozostaje aktywna do momentu zatrzymania specyfikacji aktywowania.
- Jedna konwersacja dla każdej sesji serwera używanej do uruchamiania instancji komponentu bean sterowanego komunikatami w celu przetwarzania komunikatów.

Zaawansowana właściwość specyfikacji aktywowania **Maximum server sessions** określa maksymalną liczbę sesji serwera, które mogą być jednocześnie aktywne dla danej specyfikacji aktywowania. Ta właściwość ma wartość domyślną 10. Sesje serwera są tworzone w miarę potrzeb i zamykane, jeśli były bezczynne przez okres określony przez zaawansowaną właściwość specyfikacji aktywowania **Server session pool timeout**. Wartością domyślną tej właściwości jest 300000 milisekund (5 minut).

Konwersacje są uruchamiane po utworzeniu sesji serwera i zatrzymywane po zatrzymaniu specyfikacji aktywowania lub po upłynięciu limitu czasu sesji serwera.

Oznacza to, że maksymalną liczbę konwersacji z pojedynczej specyfikacji aktywowania do programu IBM MQ można obliczyć przy użyciu następującej formuły:

```
Maximum number of conversations = Maximum server sessions + 1
```

Liczbę instancji kanału, które zostały utworzone w celu umożliwienia tej liczby konwersacji, można znaleźć, wykonując następujące obliczenia:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Każda reszta z tego obliczenia może zostać zaokrąglona w górę.

W przypadku prostej specyfikacji aktywowania, która używa wartości domyślnej dla właściwości **Maximum server sessions** , maksymalna liczba konwersacji, które mogą istnieć między wersjami WebSphere Application Server i IBM MQ dla tej specyfikacji aktywowania, jest obliczana w następujący sposób:

```
Maximum number of conversations = Maximum server sessions + 1
```

Na przykład:

```
    = 10 + 1
    = 11
```

Jeśli ta specyfikacja aktywowania nawiązuje połączenie z produktem IBM MQ przy użyciu kanału, który ma właściwość **SHARECNV** ustawioną na wartość 10, liczba utworzonych instancji kanału jest obliczana jako:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Na przykład:

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

### *Porty nasłuchiwania działające w trybie ASF (Application Server Facilities)*

Porty nasłuchiwania działające w trybie ASF i używane przez aplikacje komponentu bean sterowanego komunikatami tworzą konwersacje dla każdej sesji serwera. Jeden monitoruje miejsce docelowe pod kątem odpowiednich komunikatów, a drugi uruchamia instancję komponentu bean sterowanego komunikatami w celu przetworzenia komunikatów. Liczbę konwersacji dla każdego portu nasłuchiwania można obliczyć na podstawie maksymalnej liczby sesji.

Domyślnie porty nasłuchiwania będą działać w trybie ASF w ramach specyfikacji 1.1 definiującej mechanizm, który powinien być używany przez serwery aplikacji do wykrywania komunikatów i dostarczania ich do komponentów bean sterowanych komunikatami w celu przetwarzania. Aplikacje komponentu bean sterowane komunikatami, które są skonfigurowane do używania portów nasłuchiwania w tym domyślnym trybie konwersacji tworzenia operacji:

**Jedna konwersacja dla portu nasłuchiwania w celu monitorowania miejsca docelowego pod kątem odpowiednich komunikatów**

Porty nasłuchiwania są skonfigurowane do używania fabryki połączeń JMS . Po uruchomieniu portu nasłuchiwania wykonywane jest żądanie połączenia JMS z puli wolnych połączeń fabryki połączeń. Połączenie jest zwracane do wolnej puli po zatrzymaniu portu nasłuchiwania. Więcej informacji na temat sposobu używania puli połączeń oraz wpływu na liczbę konwersacji z programem IBM MQzawiera sekcja “Fabryki połączeń usługi JMS” na stronie 521.

**Jedna konwersacja dla każdej sesji serwera używanej do uruchamiania instancji komponentu bean sterowanego komunikatami w celu przetwarzania komunikatów**

Właściwość portu nasłuchiwania **Maximum sessions** określa maksymalną liczbę sesji serwera, które mogą być jednocześnie aktywne dla danego portu nasłuchiwania. Ta właściwość ma wartość domyślną 10. Sesje serwera są tworzone w miarę potrzeb i korzystają z sesji JMS , które są pobierane z puli sesji powiązanej z połączeniem JMS używanym przez port nasłuchiwania.

Jeśli sesja serwera była bezczynna przez czas określony przez niestandardową właściwość usługi nasłuchiwania komunikatów **SERVER.SESSION.POOL.UNUSED.TIMEOUT**, sesja jest zamykana, a używana sesja JMS jest zwracana do puli wolnych pul sesji. Sesja JMS pozostanie w puli wolnych pul sesji do czasu, aż będzie potrzebna lub zostanie zamknięta, ponieważ była bezczynna w puli wolnych sesji przez czas dłuższy niż wartość właściwości **Unused timeout** puli sesji.

Więcej informacji na temat sposobu używania puli sesji i zarządzania konwersacjami między systemami WebSphere Application Server i IBM MQ zawiera sekcja “Fabryki połączeń usługi JMS” na stronie 521.

Więcej informacji na temat niestandardowej właściwości usługi nasłuchiwania komunikatów **SERVER.SESSION.POOL.UNUSED.TIMEOUT**, patrz sekcja Monitorowanie pul sesji serwera dla portów nasłuchiwania w dokumentacji produktu WebSphere Application Server .

## Obliczanie maksymalnej liczby konwersacji z jednego portu nasłuchiwania do programu IBM MQ

Maksymalną liczbę konwersacji z jednego portu nasłuchiwania do programu IBM MQ można obliczyć za pomocą następującej formuły:

```
Maximum number of conversations = Maximum sessions + 1
```

Liczba instancji kanału, które zostaną utworzone w celu umożliwienia takiej liczby konwersacji, może zostać opracowana przy użyciu następującego obliczenia:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Każda reszta z tego obliczenia może zostać zaokrąglona w górę.

W przypadku prostego portu nasłuchiwania, który używa wartości domyślnej dla właściwości **Maximum sessions** , maksymalna liczba konwersacji, które mogą istnieć między systemami WebSphere Application Server i IBM MQ dla tego portu nasłuchiwania, jest obliczana w następujący sposób:

```
Maximum number of conversations = Maximum sessions + 1
```

Na przykład:

```
    = 10 + 1
    = 11
```

Jeśli ten port nasłuchiwania łączy się z produktem IBM MQ przy użyciu kanału, który ma właściwość **SHARECNV** ustawioną na wartość 10, to liczba instancji kanału, które zostaną utworzone, jest obliczana w następujący sposób:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Na przykład:

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

### *Porty nasłuchiwania działające w trybie bez ASF (Application Server Facilities)*

Porty nasłuchiwania działające w trybie bez ASF można skonfigurować do monitorowania miejsca docelowego kolejki i miejsca docelowego tematu przy użyciu sesji serwera. Sesje serwera mogą mieć wiele konwersacji, z których maksymalna liczba może być obliczana w każdym przypadku.

Porty nasłuchiwania można skonfigurować w taki sposób, aby były uruchamiane w trybie bez ASF, który zmienia sposób monitorowania miejsc docelowych JMS przez porty nasłuchiwania. Aplikacje komponentu bean sterowane komunikatami, używające portów nasłuchiwania w trybie działania bez ASF, tworzą konwersację dla każdej sesji serwera używanej do uruchamiania instancji komponentu bean sterowanego komunikatami w celu przetwarzania komunikatów. Właściwość portu nasłuchiwania **maksymalna liczba sesji** określa maksymalną liczbę sesji serwera, które mogą być jednocześnie aktywne dla danego portu nasłuchiwania. Wartością domyślną tej właściwości jest 10.

W trybie bez ASF port nasłuchiwania monitorujący miejsce docelowe kolejki automatycznie tworzy liczbę sesji serwera określoną przez właściwość portu nasłuchiwania **Maksymalna liczba sesji**. Wszystkie te sesje serwera wykorzystują sesje JMS pobrane z puli sesji powiązanej z połączeniem JMS używanym przez port nasłuchiwania i stale monitorują miejsce docelowe JMS pod kątem odpowiednich komunikatów.

Jeśli port nasłuchiwania jest skonfigurowany do monitorowania miejsca docelowego tematu, wartość **Maksymalna liczba sesji** jest ignorowana i używana jest pojedyncza sesja serwera.

Sesje serwera używane przez port nasłuchiwania działający w trybie bez ASF pozostają aktywne do momentu zatrzymania portu nasłuchiwania. W tym momencie używane sesje JMS są zwracane do puli wolnych pul sesji dla połączenia JMS używanego przez port nasłuchiwania.

Więcej informacji na temat sposobu używania puli sesji i zarządzania konwersacjami między systemami WebSphere Application Server i IBM MQ zawiera sekcja “Fabryki połączeń usługi JMS” na stronie 521.

Więcej informacji na temat trybu działania ASF i trybu bez ASF z WebSphere Application Serveroraz sposobu konfigurowania portów nasłuchiwania w celu użycia trybu bez ASF zawiera sekcja Przetwarzanie komunikatów w trybie ASF i trybie bez ASF.

## Obliczanie maksymalnej liczby konwersacji podczas monitorowania miejsca docelowego kolejki

Maksymalną liczbę konwersacji z pojedynczego portu nasłuchiwania, uruchomionych w trybie bez ASF i monitorujących miejsce docelowe kolejki do IBM MQ można obliczyć za pomocą następującej formuły:

```
Maximum number of conversations = Maximum sessions
```

Liczbę instancji kanału, które zostaną utworzone w celu umożliwienia takiej liczby konwersacji, można znaleźć, używając następującego obliczenia:

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

Każda reszta z tego obliczenia może zostać zaokrąglona w górę.

W przypadku prostego portu nasłuchiwania działającego w trybie bez ASF, który używa wartości domyślnej dla właściwości **Maksymalna liczba sesji** i monitorowania miejsca docelowego kolejki, maksymalna liczba konwersacji, które mogą istnieć między produktami WebSphere Application Server i IBM MQ dla tego portu nasłuchiwania, wynosi:

```
Maximum number of conversations = Maximum sessions
```

Na przykład:

```
     = 10
```

Jeśli ten port nasłuchiwania łączy się z produktem IBM MQ przy użyciu kanału, który ma właściwość **SHARECNV** ustawioną na wartość 10, to liczba utworzonych instancji kanału jest obliczana w następujący sposób:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Na przykład:

```
     = 10 / 10
     = 1
```

## Obliczanie maksymalnej liczby konwersacji podczas monitorowania miejsca docelowego tematu

Dla portu nasłuchiwania działającego w trybie bez ASF i skonfigurowanego do monitorowania miejsca docelowego tematu, liczba konwersacji z portu nasłuchiwania do IBM MQ wynosi:

```
  Maximum number of conversations = 1
```

Liczbę instancji kanału, które zostaną utworzone w celu umożliwienia takiej liczby konwersacji, można znaleźć, używając następującego obliczenia:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Każda reszta z tego obliczenia może zostać zaokrąglona w górę.

W przypadku prostego portu nasłuchiwania działającego w trybie bez ASF, który używa wartości domyślnej dla właściwości **Maksymalna liczba sesji** i monitorowania miejsca docelowego tematu, maksymalna liczba konwersacji, które mogą istnieć między produktami WebSphere Application Server i IBM MQ dla tego portu nasłuchiwania, wynosi:

```
 Maximum number of conversations = Maximum sessions
```

Na przykład:

```
     = 10
```

Jeśli ten port nasłuchiwania łączy się z produktem IBM MQ przy użyciu kanału, który ma właściwość **SHARECNV** ustawioną na wartość 10, to liczba utworzonych instancji kanału jest obliczana w następujący sposób:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Na przykład:

```
     = 10 / 10
     = 1
```

## Konfigurowanie aliasów uwierzytelniania w celu zabezpieczenia połączenia produktu WebSphere Application Server z produktem IBM MQ

Aliasy uwierzytelniania są odwzorowywane na kombinację nazwy użytkownika i hasła, która może być używana do zabezpieczania połączenia produktu WebSphere Application Server z produktem IBM MQ. Istnieje możliwość skonfigurowania fabryki połączeń z aliasem uwierzytelniania.

### *Korzystanie z aliasów uwierzytelniania w aplikacjach korporacyjnych*

Gdy aplikacja korporacyjna działająca w produkcie WebSphere Application Server próbuje utworzyć połączenie JMS z produktem IBM MQ, wyszukuje ona definicję fabryki połączeń dostawcy przesyłania komunikatów produktu IBM MQ w repozytorium Java Naming Directory Interface (JNDI) serwera aplikacji.

Jeśli definicja fabryki połączeń dostawcy przesyłania komunikatów produktu IBM MQ znajduje się w repozytorium JNDI serwera aplikacji, wywoływana jest jedna z następujących metod:

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

Jeśli fabryka połączeń została skonfigurowana ze zdefiniowanym aliasem uwierzytelniania J2C , nazwa użytkownika i hasło w aliasie uwierzytelniania mogą zostać przeniesione do produktu IBM MQ podczas tworzenia połączenia za pomocą fabryki połączeń.

## Fabryki połączeń i aliasy uwierzytelniania

Fabryki połączeń dostawcy przesyłania komunikatów produktu IBM MQ zawierają informacje na temat nawiązywania połączeń z menedżerami kolejek produktu IBM MQ . Aplikacje korporacyjne działające w obrębie produktu WebSphere Application Server mogą używać fabryk połączeń do tworzenia połączeń JMS z produktem IBM MQ.

WebSphere Application Server zapisuje definicje fabryk połączeń w repozytorium, do którego można uzyskać dostęp za pomocą JNDI. Podczas tworzenia fabryki połączeń nadawana jest jej nazwa JNDI , która jednoznacznie identyfikuje ją w zasięgu serwera aplikacji (komórki, węzła lub serwera), w którym została zdefiniowana.

Na przykład fabryka połączeń dostawcy przesyłania komunikatów produktu IBM MQ zdefiniowana w zasięgu komórki WebSphere Application Server zawiera informacje o sposobie nawiązywania połączenia z menedżerem kolejek (`myQM`) przy użyciu transportu BINDINGS. Tej fabryce połączeń jest nadawana JNDI nazwa `jms/myCF` , która jednoznacznie ją identyfikuje.

Fabryki połączeń można również skonfigurować w taki sposób, aby używała aliasu uwierzytelniania. Aliasy uwierzytelniania są odwzorowywane na kombinację nazwy użytkownika i hasła. W zależności od sposobu użycia fabryki połączeń nazwa użytkownika i hasło w aliasie uwierzytelniania mogą (lub nie) zostać przeniesione do produktu IBM MQ podczas tworzenia połączenia JMS .

**Ważne:** W wersjach wcześniejszych niż IBM MQ 8.0domyślny menedżer uprawnień do obiektów IBM MQ (OAM) wykonywał sprawdzenie autoryzacji tylko w celu upewnienia się, że nazwa użytkownika przekazana do programu IBM MQpodczas nawiązywania połączenia miała uprawnienie dostępu do menedżera kolejek.

Nie sprawdzono poprawności podanego hasła. Aby wykonać sprawdzenie uwierzytelniania i sprawdzić, czy identyfikator użytkownika i hasło są zgodne, należy zapisać wyjście zabezpieczeń kanału IBM MQ . Szczegółowe informacje na ten temat można znaleźć w sekcji “Programy obsługi wyjścia zabezpieczeń kanału” na stronie 999.

W produkcie IBM MQ 8.0menedżer kolejek sprawdza oprócz nazwy użytkownika hasło.

## Korzystanie z fabryki połączeń

Poniższe tematy zawierają informacje dotyczące używania fabryki połączeń przy użyciu wyszukiwań bezpośrednich i pośrednich:

- “Korzystanie z fabryki połączeń za pośrednictwem wyszukiwania bezpośredniego” na stronie 531
- “Korzystanie z fabryki połączeń za pośrednictwem wyszukiwania pośredniego” na stronie 532

## Korzystanie z transportu CLIENT

Fabryki połączeń skonfigurowane do używania transportu CLIENT muszą określić kanał połączenia serwera IBM MQ (SVRCONN), który będzie używany do nawiązywania połączenia z menedżerem kolejek.

Jeśli właściwość identyfikatora użytkownika agenta kanału IBM MQ (MCAUSER) pozostaje pusta dla kanału, który ma być używany przez fabrykę połączeń, wówczas fabryka połączeń może być używana z bezpośrednim lub pośrednim szukaniem.

Jeśli właściwość MCAUSER jest ustawiona na identyfikator użytkownika, ten identyfikator użytkownika jest przekazywany do pliku IBM MQ , gdy fabryka połączeń jest używana do tworzenia połączenia z produktem IBM MQ, niezależnie od tego, czy aplikacja korporacyjna używa wyszukiwania bezpośredniego, czy pośredniego.

## tabele podsumowania

W poniższych tabelach podsumowano, jakie identyfikatory użytkowników są przepływane w dół do produktu IBM MQ , gdy używany jest odpowiednio transport BINDINGS i transport CLIENT:

*Tabela 71. Tryb BINDINGS*

| **Konfiguracja** | **Wywołania aplikacji `ConnectionFactory.createConnection()`** | **Wywołania aplikacji `ConnectionFactory.createConnection(String username, String password)`** |
|---|---|---|
| Deskryptor wdrażania aplikacji nie zawiera odwołania do zasobu dla fabryki połączeń | Identyfikator użytkownika dla procesu serwera aplikacji jest przepływa w dół do IBM MQ. | Identyfikator użytkownika i hasło, które zostały przekazane do metody `ConnectionFactory.createConnection(String username, String password)` , są przekazywane w dół do IBM MQ. |
| Deskryptor wdrażania aplikacji zawiera odwołanie do zasobu dla fabryki połączeń, a właściwość **`res-auth`** jest ustawiona na wartość Application (Aplikacja). | Identyfikator użytkownika dla procesu serwera aplikacji jest przepływa w dół do IBM MQ. | Identyfikator użytkownika i hasło, które zostały przekazane do metody `ConnectionFactory.createConnection(String username, String password)` , są przekazywane w dół do IBM MQ. |
| Deskryptor wdrażania aplikacji zawiera odwołanie do zasobu dla fabryki połączeń, a właściwość **`res-auth`** jest ustawiona na wartość Container (Kontener). | Identyfikator użytkownika i hasło określone w aliasie uwierzytelniania dla fabryki połączeń są przepływane w dół do IBM MQ. | Identyfikator użytkownika i hasło określone w aliasie uwierzytelniania dla fabryki połączeń są przepływane w dół do IBM MQ. |

*Tabela 71. Tryb BINDINGS (kontynuacja)*

| **Konfiguracja** | **Wywołania aplikacji `ConnectionFactory.createConnection()`** | **Wywołania aplikacji `ConnectionFactory.createConnection(String username, String password)`** |
|---|---|---|
| Deskryptor wdrażania aplikacji zawiera odwołanie do zasobu dla fabryki połączeń, która ma właściwość **`res-auth`** ustawioną na wartość Container (Kontener), a aplikacja została skonfigurowana przy użyciu aliasu uwierzytelniania. | Identyfikator użytkownika i hasło określone w aliasie uwierzytelniania, który jest używany przez aplikację, są przepływane w dół do IBM MQ. | Identyfikator użytkownika i hasło określone w aliasie uwierzytelniania, który jest używany przez aplikację, są przepływane w dół do IBM MQ. |

*Tabela 72. Tryb CLIENT*

| **Konfiguracja** | **Wywołania aplikacji `ConnectionFactory.createConnection()`** | **Wywołania aplikacji `ConnectionFactory.createConnection(String username, String password)`** |
|---|---|---|
| Deskryptor wdrażania aplikacji nie zawiera odwołania do zasobu dla fabryki połączeń, a fabryka połączeń jest skonfigurowana do używania kanału IBM MQ z nieustawioną właściwością MCAUSER | Identyfikator użytkownika dla procesu serwera aplikacji jest przepływa w dół do IBM MQ. | Identyfikator użytkownika i hasło, które zostały przekazane do metody `ConnectionFactory.createConnection(String username, String password)` , są przekazywane w dół do IBM MQ. |
| Deskryptor wdrażania aplikacji nie zawiera odwołania do zasobu dla fabryki połączeń, a fabryka połączeń jest skonfigurowana do używania kanału IBM MQ z właściwością MCAUSER ustawioną na identyfikator użytkownika. | Identyfikator użytkownika określony we właściwości MCAUSER kanału IBM MQ , który ma być używany przez fabrykę połączeń, jest przepływany w dół do produktu IBM MQ. | Identyfikator użytkownika określony we właściwości MCAUSER kanału IBM MQ , który ma być używany przez fabrykę połączeń, jest przepływany w dół do produktu IBM MQ. |
| Deskryptor wdrażania aplikacji zawiera odwołanie do zasobu dla fabryki połączeń, dla której właściwość **`res-auth`** jest ustawiona na wartość *Aplikacja* , a fabryka połączeń jest skonfigurowana do używania kanału IBM MQ z nieustawioną właściwością MCAUSER | Identyfikator użytkownika dla procesu serwera aplikacji jest przepływa w dół do IBM MQ. | Identyfikator użytkownika i hasło, które zostały przekazane do metody `ConnectionFactory.createConnection(String username, String password)` , są przekazywane w dół do IBM MQ. |

*Tabela 72. Tryb CLIENT (kontynuacja)*

| **Konfiguracja** | **Wywołania aplikacji `ConnectionFactory.createConnection()`** | **Wywołania aplikacji `ConnectionFactory.createConnection(String username, String password)`** |
|---|---|---|
| Deskryptor wdrażania aplikacji zawiera odwołanie do zasobu dla fabryki połączeń, dla której właściwość **`res-auth`** jest ustawiona na wartość *Aplikacja* , a fabryka połączeń jest skonfigurowana do używania kanału IBM MQ , dla którego właściwość MCAUSER jest ustawiona na identyfikator użytkownika. | Identyfikator użytkownika określony we właściwości MCAUSER w kanale IBM MQ , który ma być używany przez fabrykę połączeń, jest podawany w dół do wartości IBM MQ. | Identyfikator użytkownika określony we właściwości MCAUSER w kanale IBM MQ , który ma być używany przez fabrykę połączeń, jest podawany w dół do wartości IBM MQ. |
| Deskryptor wdrażania aplikacji zawiera odwołanie do zasobu dla fabryki połączeń, która ma właściwość **`res-auth`** ustawioną na wartość "*Container* (Kontener), a fabryka połączeń skonfigurowana jest do używania kanału IBM MQ z nieustawioną właściwością MCAUSER | Identyfikator użytkownika i hasło określone w aliasie uwierzytelniania dla fabryki połączeń są przepływane w dół do IBM MQ. | Identyfikator użytkownika i hasło określone w aliasie uwierzytelniania dla fabryki połączeń są przepływane w dół do IBM MQ. |
| Deskryptor wdrażania aplikacji zawiera odwołanie do zasobu dla fabryki połączeń, która ma właściwość **`res-auth`** ustawioną na wartość*Container* , a fabryka połączeń skonfigurowana jest do używania kanału IBM MQ z właściwością MCAUSER ustawioną na identyfikator użytkownika. | Identyfikator użytkownika określony we właściwości MCAUSER w kanale IBM MQ , który ma być używany przez fabrykę połączeń, jest podawany w dół do wartości IBM MQ. | Identyfikator użytkownika określony we właściwości MCAUSER w kanale IBM MQ , który ma być używany przez fabrykę połączeń, jest podawany w dół do wartości IBM MQ. |
| Deskryptor wdrażania aplikacji zawiera odwołanie do zasobu dla fabryki połączeń, która ma właściwość **`res-auth`** ustawioną na wartość*Kontener* , a aplikacja została skonfigurowana z aliasem uwierzytelniania, a fabryka połączeń jest skonfigurowana do używania kanału IBM MQ z nieustawioną właściwością MCAUSER | Identyfikator użytkownika i hasło określone w aliasie uwierzytelniania, który jest używany przez aplikację, są przepływane w dół do IBM MQ. | Identyfikator użytkownika i hasło określone w aliasie uwierzytelniania, który jest używany przez aplikację, są przepływane w dół do IBM MQ. |

| *Tabela 72. Tryb CLIENT (kontynuacja)* | | |
|---|---|---|
| **Konfiguracja** | **Wywołania aplikacji `ConnectionFactory.createConnection()`** | **Wywołania aplikacji `ConnectionFactory.createConnection(String username, String password)`** |
| Deskryptor wdrażania aplikacji zawiera odwołanie do zasobu dla fabryki połączeń, która ma właściwość **`res-auth`** ustawioną na wartość *Container* (Kontener), a aplikacja została skonfigurowana z aliasem uwierzytelniania, a fabryka połączeń jest skonfigurowana do używania kanału IBM MQ , który ma ustawioną wartość MCAUSER na identyfikator użytkownika. | Identyfikator użytkownika określony we właściwości MCAUSER w kanale IBM MQ , który ma być używany przez fabrykę połączeń, jest podawany w dół do wartości IBM MQ. | Identyfikator użytkownika określony we właściwości MCAUSER w kanale IBM MQ , który ma być używany przez fabrykę połączeń, jest podawany w dół do wartości IBM MQ. |

### *Korzystanie z fabryki połączeń za pośrednictwem wyszukiwania bezpośredniego*

Po zdefiniowaniu fabryki połączeń dostawcy przesyłania komunikatów produktu IBM MQ aplikacja korporacyjna może wyszukać definicję fabryki połączeń i użyć jej do utworzenia połączenia produktu JMS z menedżerem kolejek produktu IBM MQ . Można to zrobić poprzez bezpośrednie spojrzenie w górę.

Aby użyć wyszukiwania bezpośredniego, aplikacja korporacyjna łączy się z repozytorium JNDI serwera aplikacji, wywołując następującą metodę:

```
InitialContext ctx = new InitialContext();
```

Po nawiązaniu połączenia z repozytorium JNDI aplikacja korporacyjna identyfikuje definicję fabryki połączeń przy użyciu nazwy JNDI fabryki połączeń w następujący sposób:

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("jms/myCF");
```

**Uwagi:**

- Programista aplikacji musi znać nazwę JNDI wymaganej fabryki połączeń podczas projektowania aplikacji korporacyjnej. Ponieważ nazwa JNDI jest zakodowana na stałe w aplikacji, jeśli nazwa JNDI ulegnie zmianie, należy ponownie zapisać i wdrożyć aplikację.
- Jeśli definicja fabryki połączeń jest używana w ten sposób, nazwa użytkownika i hasło określone w aliasie uwierzytelniania (który został skonfigurowany do użycia przez fabrykę połączeń) nie są przepływane do produktu IBM MQ. Ma to na celu uniemożliwienie nieautoryzowanym aplikacjom zidentyfikowania fabryki połączeń i możliwość użycia jej do nawiązania połączenia z zabezpieczonym systemem IBM MQ .

Nazwa użytkownika i hasło, które są przepływane w dół do produktu IBM MQ , zależą od metody użytej do utworzenia połączenia JMS z fabryki połączeń.

Jeśli aplikacja tworzy połączenie JMS przy użyciu tej metody:

```
ConnectionFactory.createConnection()
```

Domyślna tożsamość użytkownika jest przekazywana do programu IBM MQ. Jest to nazwa użytkownika i hasło, które uruchomiły serwer aplikacji, na którym działa aplikacja korporacyjna.

Alternatywnie aplikacja może utworzyć połączenie JMS , wywołując metodę:

```
ConnectionFactory.createConnection(String username, String password)
```

Jeśli aplikacja wykonała bezpośrednie wyszukiwania w fabryce połączeń, a następnie wywołała tę metodę, nazwa użytkownika i hasło, które zostały przekazane do metody `createConnection()` , są przekazywane w dół do metody IBM MQ.

**Ważne:** Przed wersją IBM MQ 8.0program IBM MQ przetwarzał sprawdzenie autoryzacji tylko w celu upewnienia się, że nazwa użytkownika, który został wyłączony, miała uprawnienie dostępu do menedżera kolejek.

Nie sprawdzono hasła. Aby wykonać sprawdzenie uwierzytelniania i sprawdzić, czy nazwa użytkownika i hasło są poprawne, należy zapisać wyjście zabezpieczeń kanału IBM MQ . Szczegółowe informacje na ten temat można znaleźć w sekcji “Programy obsługi wyjścia zabezpieczeń kanału” na stronie 999.

W produkcie IBM MQ 8.0menedżer kolejek sprawdza oprócz nazwy użytkownika hasło.

### *Korzystanie z fabryki połączeń za pośrednictwem wyszukiwania pośredniego*

Jeśli podczas pisania aplikacji korporacyjnej nazwa JNDI fabryki połączeń jest nieznana lub jeśli aplikacja ma zostać zainstalowana na różnych serwerach aplikacji przy użyciu innej fabryki połączeń, o innej nazwie JNDI (w zależności od serwera aplikacji, na którym jest zainstalowana), można ją znaleźć przy użyciu odwołania do zasobu. Można to zrobić za pomocą wyszukiwania pośredniego.

## Przykład

Zamiast bezpośredniego wyszukiwania fabryki połączeń przy użyciu parametru `jms/myCF`, aplikacja korporacyjna zawiera odwołanie do zasobu o nazwie lokalnej JNDI : `jms/myResourceReferenceCF`.

Aby użyć tej nazwy JNDI , aplikacja nawiązuje połączenie z repozytorium JNDI serwera aplikacji w taki sam sposób, jak w przypadku wykonywania wyszukiwania bezpośredniego:

```
InitialContext ctx = new InitialContext();
```

Zamiast identyfikować `jms/myCF` bezpośrednio, aplikacja identyfikuje teraz nazwę JNDI odwołania do zasobu:

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("java:comp/env/jms/
myResourceReferenceCF");
```

Aby poinformować serwer aplikacji, że aplikacja korporacyjna wykonuje pośrednie wyszukiwania, potrzebny jest przedrostek `java:comp/env` dla nazwy lokalnej JNDI .

Po wdrożeniu aplikacji użytkownik odwzorowuje JNDI nazwę odwołania do zasobu `jms/myResourceReferenceCF` na nazwę JNDI fabryki połączeń, która została już utworzona przez aplikację: `jms/myCF`.

Po uruchomieniu aplikacja wyszukuje fabrykę połączeń JMS przy użyciu lokalnej nazwy JNDI , którą serwer aplikacji odwzorowuje na: `jms/myCF`. Ta fabryka połączeń jest następnie używana przez aplikację do tworzenia połączenia z produktem IBM MQ.

## Aliasy uwierzytelniania i wyszukiwania pośrednie

Odwołanie do zasobu umożliwia również zdefiniowanie dodatkowych właściwości, które zmieniają zachowanie udostępnionej fabryki połączeń. Jedną z właściwości odwołania do zasobu jest **`res-auth`**. Wartość tej właściwości określa, czy aplikacja korporacyjna powinna używać aliasu uwierzytelniania fabryki połączeń, na który jest odwzorowywane odwołanie do zasobu podczas tworzenia połączenia z produktem IBM MQ (jeśli zdefiniowano alias uwierzytelniania), czy też aplikacja określa własną nazwę użytkownika i hasło.

Wartością domyślną tej właściwości jest *Application*(Aplikacja). Oznacza to, że nazwa użytkownika i hasło, które są przepływane do menedżera kolejek w momencie tworzenia połączenia JMS , są określane przez samą aplikację. Alias uwierzytelniania fabryki połączeń, na który jest odwzorowywane odwołanie do zasobu, nie jest używany.

Aplikacje mogą tworzyć połączenia JMS za pomocą jednej z następujących metod:

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

Jeśli aplikacja używa produktu `ConnectionFactory.createConnection()`, a parametr **res-auth** ma wartość *Aplikacja*, domyślna tożsamość użytkownika jest ustawiana w dół do wartości IBM MQ. Jest to nazwa użytkownika i hasło, które uruchomiły serwer aplikacji, na którym działa aplikacja korporacyjna.

Jeśli aplikacja używa serwera `ConnectionFactory.createConnection(String username, String password)`, a parametr **res-auth** ma wartość *Aplikacja*, nazwa użytkownika i hasło przekazywane do metody są wysyłane do systemu IBM MQ.

Aby użyć aliasu uwierzytelniania zdefiniowanego w fabryce połączeń, na który jest odwzorowywane odwołanie do zasobu podczas tworzenia połączenia, należy ustawić właściwość **res-auth** na wartość *Container*(Kontener). Gdy aplikacja tworzy połączenie JMS , używane są szczegóły aliasu uwierzytelniania, nawet jeśli wywołanie `createConnection` określa nazwę użytkownika i hasło.

### Nadpisywanie aliasu uwierzytelniania podczas korzystania z wyszukiwania pośredniego

Jeśli aplikacja używa odwołania do zasobu, które ma właściwość **res-auth** ustawioną na wartość *Container*, można nadpisać alias uwierzytelniania używany podczas tworzenia połączeń JMS .

Aby przesłonić alias uwierzytelniania, odwołanie do zasobu musi zawierać dodatkową właściwość o nazwie **authDataAlias**, która jest odwzorowana na istniejący alias uwierzytelniania, który został już utworzony w środowisku serwera aplikacji, w którym zostanie wdrożona aplikacja. Tę właściwość można określić dla dowolnych odwołań do zasobów, które są tworzone przy użyciu narzędzi Rational udostępnianych przez produkt IBM.

Za pomocą tej metody można użyć innego aliasu uwierzytelniania, jeśli używana jest fabryka połączeń produktu JMS , która została wyszukana pośrednio. Jeśli podany alias uwierzytelniania nie istnieje, można określić nowy alias po zainstalowaniu aplikacji korporacyjnej. Więcej informacji na ten temat zawiera sekcja *Odwołania do zasobów* w dokumentacji produktu WebSphere Application Server .

**Informacje pokrewne dotyczące produktu WebSphere Application Server 8.5.5**

Odwołania do zasobów

**Informacje pokrewne dotyczące produktu WebSphere Application Server 8.0**

Odwołania do zasobów

**Informacje pokrewne dotyczące produktu WebSphere Application Server 7.0**

Odwołania do zasobów

## Równoważenie obciążenia komponentów bean sterowanych komunikatami w przypadku używania klastrów WebSphere Application Server

Jeśli używane są aplikacje komponentu bean sterowanego komunikatami wdrożone w klastrze WebSphere Application Server 7.0 i WebSphere Application Server 8.0 i skonfigurowane do działania w trybie normalnym dostawcy usługi przesyłania komunikatów produktu IBM MQ , większość komunikatów jest przetwarzana przez jeden z elementów klastra. Obciążenie elementów klastra można zrównoważyć w celu rozdzielenia przetwarzania komunikatów na więcej niż jeden element klastra.

Produkt IBM MQ zawiera funkcję o nazwie **Asynchronous consume**, która umożliwia aplikacjom asynchroniczne konsumowanie komunikatów z kolejki przy użyciu funkcji API o nazwie **MQCB** i **MQCTL**.

Aplikacje bean sterowane komunikatami działające w systemach WebSphere Application Server 7.0 i WebSphere Application Server 8.0, które korzystają z trybu normalnego dostawcy usługi przesyłania

komunikatów produktu IBM MQ , będą automatycznie korzystać z tej funkcji. Po uruchomieniu aplikacji zostanie skonfigurowany asynchroniczny konsument w miejscu docelowym JMS , które zostało skonfigurowane do monitorowania przez wywołanie funkcji **MQCB**. Następnie wywoływana jest funkcja API **MQCTL** w celu wskazania, że aplikacja jest gotowa do odbierania komunikatów z miejsca docelowego JMS .

Po wdrożeniu aplikacji komponentów bean sterowanych komunikatami w klastrze WebSphere Application Server każdy element klastra będzie konfigurować konsument asynchroniczny dla miejsca docelowego JMS , które komponent bean sterowany komunikatami monitoruje pod kątem komunikatów. Menedżer kolejek produktu IBM MQ , który udostępnia miejsce docelowe JMS , jest następnie odpowiedzialny za powiadamianie elementu klastra o odpowiednim komunikacie w miejscu docelowym produktu JMS , który ma zostać przetworzony.

Jeśli produkt WebSphere Application Server nawiązuje połączenie z menedżerem kolejek produktu IBM MQ , komunikaty docierające do miejsca docelowego produktu JMS będą bardziej równomiernie dystrybuowane do wszystkich asynchronicznych konsumentów, które zostały zarejestrowane w tym miejscu docelowym produktu JMS . W przypadku aplikacji komponentów bean sterowanych komunikatami wdrożonych w klastrze WebSphere Application Server 7.0 i WebSphere Application Server 8.0 oznacza to, że komunikaty będą rozdzielane bardziej równomiernie między elementy klastra.

**Zadania pokrewne**

Konfigurowanie właściwości JMS **PROVIDERVERSION**

# Korzystanie z pakietu IBM MQ Headers

Pakiet IBM MQ Headers udostępnia zestaw interfejsów i klas programu pomocy, których można użyć do manipulowania nagłówkami języka IBM MQ komunikatu. Zwykle pakiet nagłówków IBM MQ jest używany, ponieważ usługi administracyjne mają być wykonywane przy użyciu serwera komend (przy użyciu komunikatów PCF (Programmable Command Format)).

## O tym zadaniu

Pakiet IBM MQ Headers znajduje się w pakietach `com.ibm.mq.headers` i `com.ibm.mq.headers.pcf` . Tego narzędzia można użyć dla obu alternatywnych interfejsów API, które są udostępniane przez produkt IBM MQ do użytku w aplikacjach Java :

- IBM MQ classes for Java (zwana również IBM MQ Podstawową Java).
- IBM MQ classes for Java Message Service (IBM MQ classes for JMS, zwany również IBM MQ JMS).

IBM MQ Podstawowe Java aplikacje zwykle manipulują obiektami MQMessage, a klasy obsługi nagłówków mogą bezpośrednio wchodzić w interakcję z tymi obiektami, ponieważ w ich rodzimym rozumieniu są interfejsy IBM MQ podstawowe Java .

W produkcie IBM MQ JMSładunek komunikatu jest zwykle obiektem typu String lub tablicą bajtów, którym można manipulować za pomocą strumieni DataInput i DataOutput . Pakiet nagłówków IBM MQ może być używany do interakcji z tymi strumieniami danych i jest odpowiedni do manipulowania komunikatami produktu MQ wysyłanymi i odbieranymi przez aplikacje produktu IBM MQ JMS .

Dlatego, mimo że pakiet IBM MQ Headers zawiera odwołania do pakietu IBM MQ Base Java , jest on również przeznaczony do użycia w aplikacjach IBM MQ JMS i jest odpowiedni do użycia w środowiskach Java Platform, Enterprise Edition (Java EE).

Typowym sposobem korzystania z pakietu IBM MQ Headers jest manipulowanie komunikatami administracyjnymi w formacie Programmable Command Format (PCF), na przykład z jednego z następujących powodów:

- Aby uzyskać dostęp do szczegółów dotyczących zasobu IBM MQ .
- Służy do monitorowania głębokości kolejki.
- Zablokowanie dostępu do kolejki.

Korzystając z komunikatów PCF z interfejsem API IBM MQ JMS , tego rodzaju administrowanie zasobami zorientowanymi na aplikacje może być wykonywane z poziomu aplikacji Java EE bez konieczności korzystania z interfejsu API IBM MQ Base Java .

### Procedura

- Informacje na temat używania pakietu IBM MQ Headers do manipulowania nagłówkami komunikatów dla produktu IBM MQ classes for Javazawiera sekcja “używanie zIBM MQ classes for Java” na stronie 535.
- Informacje na temat używania pakietu IBM MQ Headers do manipulowania nagłówkami komunikatów dla produktu IBM MQ classes for JMSzawiera sekcja “używanie zIBM MQ classes for JMS” na stronie 535.

## używanie zIBM MQ classes for Java

Aplikacje IBM MQ classes for Java zwykle manipulują obiektami MQMessage, a klasy obsługi nagłówków mogą bezpośrednio współdziałać z tymi obiektami, ponieważ w ich rodzimym rozumieniu są interfejsy IBM MQ classes for Java .

### O tym zadaniu

IBM MQ udostępnia kilka przykładowych aplikacji, które demonstrują sposób używania pakietu IBM MQ Headers z interfejsem API IBM MQ Base Java (IBM MQ classes for Java).

Przykłady przedstawiają dwie rzeczy:

- Sposób tworzenia komunikatu PCF w celu wykonania czynności administracyjnej i przeanalizowania komunikatu odpowiedzi.
- Sposób wysyłania tego komunikatu PCF za pomocą komendy IBM MQ classes for Java.

W zależności od używanej platformy, przykłady te są instalowane w katalogu `pcf` w katalogu `samples` lub `tools` instalacji IBM MQ (patrz sekcja “Katalogi instalacyjne produktu IBM MQ classes for Java” na stronie 366).

### Procedura

1. Utwórz komunikat PCF, aby wykonać działanie administracyjne i przeanalizować komunikat odpowiedzi.
2. Wyślij ten komunikat PCF za pomocą programu IBM MQ classes for Java.

**Pojęcia pokrewne**

“Obsługa nagłówków komunikatów produktu IBM MQ w produkcie IBM MQ classes for Java” na stronie 394
Dostępne są klasy języka Java reprezentujące różne typy nagłówków komunikatów. Dostępne są również dwie klasy pomocnicze.

“Obsługa komunikatów PCF w produkcie IBM MQ classes for Java” na stronie 399
Klasy Java są udostępniane w celu tworzenia i analizowania komunikatów o strukturze PCF oraz w celu ułatwienia wysyłania żądań PCF i gromadzenia odpowiedzi PCF.

## używanie zIBM MQ classes for JMS

Aby używać nagłówków IBM MQ z elementem IBM MQ classes for JMS, należy wykonać te same podstawowe kroki, co w przypadku elementu IBM MQ classes for Java. Komunikat PCF może zostać utworzony, a odpowiedź może zostać przeanalizowana dokładnie w taki sam sposób, używając pakietu nagłówków IBM MQ i tego samego kodu przykładowego, co w przypadku kodu IBM MQ classes for Java.

## O tym zadaniu

Aby wysłać komunikat PCF przy użyciu funkcji API języka IBM MQ , ładunek komunikatu musi zostać zapisany w komunikacie JMS (bajtowym) i wysłany przy użyciu standardowych funkcji API języka JMS . Należy tylko pamiętać, że komunikat nie może zawierać nagłówka JMS RFH2 ani żadnych innych nagłówków z konkretnymi wartościami w strukturze MQMD.

Aby wysłać komunikat PCF, wykonaj następujące kroki. Sposób tworzenia komunikatu PCF i wyodrębniania informacji z komunikatu odpowiedzi jest taki sam, jak w przypadku elementu IBM MQ classes for Java (patrz sekcja “używanie zIBM MQ classes for Java” na stronie 535).

## Procedura

1. Utwórz miejsce docelowe kolejki produktu JMS , które reprezentuje system SYSTEM.ADMIN.COMMAND.QUEUE.

   Aplikacje IBM MQ JMS wysyłają komunikaty PCF do systemu SYSTEM.ADMIN.COMMAND.QUEUEi wymagają dostępu do obiektu docelowego JMS, który reprezentuje tę kolejkę. Miejsce docelowe musi mieć ustawione następujące właściwości:

   ```
   WMQ_MQMD_WRITE_ENABLED = YES
   WMQ_MESSAGE_BODY = MQ
   ```

   Jeśli używana jest baza danych WebSphere Application Server, należy zdefiniować te właściwości jako właściwości niestandardowe w miejscu docelowym.

   Aby programowo utworzyć miejsce docelowe z poziomu aplikacji, należy użyć następującego kodu:

   ```
   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. Przekształć komunikat PCF w komunikat JMS Bytes zawierający poprawne wartości MQMD.

   Należy utworzyć komunikat JMS Bytes i zapisać w nim komunikat PCF. Należy utworzyć kolejkę odpowiedzi, ale nie musi ona mieć żadnych konkretnych ustawień.

   Poniższy przykładowy fragment kodu przedstawia sposób tworzenia komunikatu JMS bajtowego i zapisywania w nim obiektu com.ibm.mq.headerspcf.PCFMessage . Obiekt PCFMessage (pcfCmd) został wcześniej zbudowany przy użyciu pakietu IBM MQ Headers. (Należy zauważyć, że pakiet do załadowania komunikatu PCFMessage to com.ibm.mq.headers.pcf.PCFMessage).

   ```
    // create the JMS Bytes Message
   final BytesMessage msg = session.createBytesMessage();

   // Create the wrapping streams to put the bytes into the message payload
   ByteArrayOutputStream baos = new ByteArrayOutputStream();
   DataOutput dataOutput = new DataOutputStream(baos);

   // Set the JMSReplyTo so the answer comes back
   msg.setJMSReplyTo(new MQQueue("adminResp"));

   // write the pcf into the stream
   pcfCmd.write(dataOutput);
   baos.flush();
   msg.writeBytes(baos.toByteArray());

   // we have taken control of the MD, so need to set all
   // flags in the MD that we require - main one is the format
   msg.setJMSPriority(4);
   msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_PERSISTENCE,
     CMQC.MQPER_NOT_PERSISTENT);
   msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_EXPIRY, 300);
   msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_REPORT,
       CMQC.MQRO_PASS_CORREL_ID);
   msg.setStringProperty(WMQConstants.JMS_IBM_MQMD_FORMAT, "MQADMIN");

   // and send the message
   sender.send(msg);
   ```

3. Wyślij komunikat i odbierz odpowiedź za pomocą standardowych interfejsów API JMS .
4. Przekształć komunikat odpowiedzi w komunikat PCF na potrzeby przetwarzania.

   Aby pobrać komunikat odpowiedzi i przetworzyć go jako komunikat PCF, należy użyć następującego kodu:

```
 // 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);
```

**Pojęcia pokrewne**

“Komunikaty produktu JMS” na stronie 149
Komunikaty produktu JMS składają się z nagłówka, właściwości i treści. JMS definiuje pięć typów treści komunikatu.

# IBM i Konfigurowanie produktu IBM MQ w systemie IBM i z systemem Java i JMS

Ta kolekcja tematów zawiera przegląd sposobu konfigurowania i testowania produktu IBM MQ z Java i JMS w systemie IBM i przy użyciu komend CL lub środowiska qshell.

**Uwaga:**

- IBM MQ 8.0, `ldap.jar`, `jndi.jar` i `jta.jar` są częścią pakietu JDK.
- V 9.3.0 JM 3.0 V 9.3.0 W produkcie IBM MQ 9.3.0produkt Jakarta Messaging 3.0 jest obsługiwany na potrzeby tworzenia nowych aplikacji. Produkt IBM MQ 9.3.0 nadal obsługuje produkt JMS 2.0 dla istniejących aplikacji. Nie jest obsługiwane używanie zarówno interfejsu API Jakarta Messaging 3.0 , jak i interfejsu API JMS 2.0 w tej samej aplikacji. Więcej informacji na ten temat zawiera sekcja Używanie klas IBM MQ dla przesyłania komunikatów JMS/Jakarta.

## Korzystanie z komend CL

Ustawiona zmienna CLASSPATH jest przeznaczona do testowania przy użyciu produktu MQ Base Java, usługi JMS z interfejsem JNDI i usługi JMS bez interfejsu JNDI.

Jeśli w katalogu `/home/Userprofile` nie jest używany plik `.profile` , należy ustawić poniżej zmienne środowiskowe na poziomie systemu. Można sprawdzić, czy zostały one ustawione za pomocą komendy **WRKENVVAR** .

1. Aby wyświetlić zmienne środowiskowe dla całego systemu, wydaj komendę: **WRKENVVAR LEVEL (*SYS)**
2. Aby wyświetlić zmienne środowiskowe specyficzne dla zadania, wydaj komendę: **WRKENVVAR LEVEL (*JOB)**
3. Jeśli zmienna CLASSPATH nie jest ustawiona, wykonaj następującą komendę:

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)
```

JMS 2.0

```
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. Jeśli parametr QIBM_MULTI_THREADED nie jest ustawiony, wydaj następującą komendę:

```
ADDENVVAR ENVVAR(QIBM_MULTI_THREADED) VALUE('Y') LEVEL(*SYS)
```

5. Jeśli opcja QIBM_USE_DESCRIPTOR_STDIO nie jest ustawiona, wydaj następującą komendę:

```
ADDENVVAR ENVVAR(QIBM_USE_DESCRIPTOR_STDIO) VALUE('I') LEVEL(*SYS)
```

6. Jeśli wartość QSH_REDIRECTION_TEXTDATA nie jest ustawiona, wydaj następującą komendę:

```
ADDENVVAR ENVVAR(QSH_REDIRECTION_TEXTDATA) VALUE('Y') LEVEL(*SYS)
```

## Korzystanie ze środowiska qshell

Jeśli używane jest środowisko QSHELL, można skonfigurować plik `.profile` w katalogu `/home/Userprofile` . Więcej informacji można znaleźć w dokumentacji Qshell Interpreter (qsh).

Określ następujące wartości w pliku `.profile`. Należy zauważyć, że instrukcja CLASSPATH musi znajdować się w jednym wierszu lub musi być oddzielona w różnych wierszach za pomocą znaku \ .

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
```

JMS 2.0

```
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
```

Upewnij się, że biblioteka QMQMJAVA znajduje się na liście bibliotek, wprowadzając komendę **DSPLIBL**.

Jeśli biblioteki QMQMJAVA nie ma na liście, dodaj ją za pomocą następującej komendy: **ADDLIBLE LIB (QMQMJAVA)**

## IBM i Testowanie produktu IBM MQ w systemie IBM i przy użyciu produktu Java

Sposób testowania produktu IBM MQ z produktem Java przy użyciu przykładowego programu MQIVP.

### Testowanie IBM MQ podstawy Java

Wykonaj następującą procedurę:

1. Sprawdź, czy menedżer kolejek jest uruchomiony i czy jego stan jest aktywny, wprowadzając następującą komendę:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. Sprawdź, czy jest to środowisko JAVA.CHANNEL został utworzony za pomocą następującej komendy:

   ```
   WRKMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
   ```

   a. Jeśli jest to JAVA.CHANNEL nie istnieje, wprowadź następującą komendę:

   ```
   CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
   ```

3. Sprawdź, czy program nasłuchujący menedżera kolejek jest uruchomiony dla portu 1414 lub dowolnego używanego portu, wprowadzając komendę **WRKMQMLSR** .

   a. Jeśli dla menedżera kolejek nie został uruchomiony żaden program nasłuchujący, wprowadź następującą komendę:

   ```
   STRMQMLSR PORT(xxxx) MQMNAME(QMGRNAME)
   ```

#### Uruchamianie przykładowego programu testowego MQIVP

1. Uruchom powłokę qshell z wiersza komend, wprowadzając komendę STRQSH.
2. Sprawdź, czy ustawiona jest poprawna zmienna CLASSPATH, wprowadzając komendę **export** , a następnie wydając komendę **cd** w następujący sposób:

   ```
   cd /qibm/proddata/mqm/java/samples/wmqjava/samples
   ```

3. Uruchom program **java** , wprowadzając następującą komendę:

   ```
   java MQIVP
   ```

Po wyświetleniu monitu można nacisnąć klawisz ENTER:

- Typ połączenia
- Adres IP
- Nazwa menedżera kolejek

aby użyć wartości domyślnych. Powoduje to sprawdzenie powiązań produktu, które można znaleźć w bibliotece QMQMJAVA.

Zostaną wyświetlone dane wyjściowe podobne do poniższego przykładu. Należy zauważyć, że informacje o prawach autorskich zależą od używanej wersji produktu.

```
> java MQIVP
MQSeries for Java Installation Verification Program
```

```
5724-H72 (C) Copyright IBM Corp. 2011, 2024. 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 ...>
$
```

### Testowanie połączenia klienta IBM MQ Java

Należy określić:

- Typ połączenia
- Adres IP
- Port
- Kanał połączenia z serwerem
- Menedżer kolejek

Zostaną wyświetlone dane wyjściowe podobne do poniższego przykładu. Należy zauważyć, że informacje o prawach autorskich zależą od używanej wersji produktu.

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. 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 Testowanie produktu IBM MQ w systemie IBM i przy użyciu produktu JMS

Sposób testowania produktu IBM MQ za pomocą interfejsu JMS z interfejsem JNDI i bez niego

### Testowanie interfejsu JMS bez interfejsu JNDI przy użyciu przykładu IVTRun

Wykonaj następującą procedurę:

1. Sprawdź, czy menedżer kolejek jest uruchomiony i czy jego stan jest aktywny, wprowadzając następującą komendę:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. Uruchom powłokę qshell z wiersza komend, wprowadzając komendę **STRQSH** .
3. Użyj komendy **cd** , aby zmienić katalog w następujący sposób:

```
cd /qibm/proddata/mqm/java/bin
```

4. Uruchom plik skryptu:

```
IVTRun -nojndi [-m qmgrname]
```

Zostaną wyświetlone dane wyjściowe podobne do poniższego przykładu. Należy zauważyć, że informacje o prawach autorskich zależą od wersji używanych produktów:

```
 IVTRun -nojndi -m ELCRTP19

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, 2024.
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:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000c012
JMSTimestamp: 1020273404500
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:QP0ZSPWT 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
$>
$
```

## Testowanie trybu klienta IBM MQ JMS bez interfejsu JNDI

Wykonaj następującą procedurę:

1. Sprawdź, czy menedżer kolejek jest uruchomiony i czy jego stan jest aktywny, wprowadzając następującą komendę:

```
WRKMQM MQMNAME(QMGRNAME)
```

2. Sprawdź, czy kanał połączenia z serwerem został utworzony, wprowadzając następującą komendę:

```
WRKMQMCHL CHLNAME( SYSTEM.DEF.SVRCONN ) CHLTYPE(*SVRCN)
MQMNAME(QMGRNAME)
```

3. Sprawdź, czy program nasłuchujący został uruchomiony dla poprawnego portu, wprowadzając komendę **WRKMQMLSR** .
4. Uruchom powłokę qshell z wiersza komend, wprowadzając komendę **STRQSH** .
5. Sprawdź, czy zmienna CLASSPATH jest poprawna, wprowadzając komendę **export** .
6. Użyj komendy **cd** , aby zmienić katalog w następujący sposób:

```
cd /qibm/proddata/mqm/java/bin
```

7. Uruchom plik skryptu:

```
IVTRun -nojndi -client -m QMgrName -host hostname [-port port] [-channel channel]
```

Zostaną wyświetlone dane wyjściowe podobne do poniższego przykładu. Należy zauważyć, że informacje o prawach autorskich zależą od wersji używanych produktów.

```
> IVTRun -nojndi -client -m ELCRTP19 -host ELCRTP19 -port 1414 -channel SYSTEM.DEF.SVRCONN

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
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:QMQM
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
$
```

## Testowanie produktu IBM MQ JMS przy użyciu interfejsu JNDI

Sprawdź, czy menedżer kolejek jest uruchomiony i czy jego stan jest aktywny, wprowadzając następującą komendę:

```
WRKMQM MQMNAME(QMGRNAME)
```

### Korzystanie z przykładowego skryptu testowego IVTRun

Wykonaj następującą procedurę:

1. Wprowadź odpowiednie zmiany w pliku `JMSAdmin.config` . Aby edytować ten plik, należy użyć komendy **EDTF** (Edycja pliku-Edit File) z wiersza komend systemu IBM i .

   ```
   EDTF '/qibm/proddata/mqm/java/bin/JMSAdmin.config'
   ```

   .

   a. Aby użyć protokołu LDAP dla produktu Weblogic, usuń komentarz z:

   ```
   INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
   ```

   b. Aby użyć LDAP dla WebSphere Application Server, usuń komentarz z:

   ```
   INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
   ```

   c. Aby przetestować system plików, usuń komentarz z:

   ```
   INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
   ```

   d. Upewnij się, że wybrano poprawny adres PROVIDER_URL, usuwając komentarz z odpowiedniego wiersza.

   e. Przekształć w komentarz wszystkie pozostałe wiersze, używając symbolu *#* .

   f. Po wprowadzeniu wszystkich zmian naciśnij klawisze **F2=Save** i **F3=Exit**.

2. Uruchom powłokę qshell z wiersza komend, wprowadzając komendę **STRQSH** .
3. Sprawdź, czy zmienna CLASSPATH jest poprawna, wprowadzając komendę **export** .
4. Użyj komendy **cd** , aby zmienić katalog w następujący sposób:

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

5. Uruchom skrypt **IVTSetup** , aby utworzyć obiekty administrowane (*FabrykaMQQueueConnection* i *MQQueue*), wprowadzając komendę **IVTSetup** .
6. Uruchom skrypt IVTRun, wprowadzając następującą komendę:

   ```
   IVTRun -url providerURL [-icf initCtxFact]
   ```

Zostaną wyświetlone dane wyjściowe podobne do poniższego przykładu. Należy zauważyć, że informacje o prawach autorskich zależą od wersji używanych produktów.

```
> 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, 2024. All Rights Reserved.
Starting WebSphere MQ classes for Java Message Service Administration

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, 2024. 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:QP0ZSPWT 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
$
```

# Programowanie aplikacji Java przy użyciu repozytorium Maven

Podczas tworzenia aplikacji Java dla systemu IBM MQprzy użyciu repozytorium Maven do automatycznego instalowania zależności nie ma potrzeby jawnego instalowania niczego przed użyciem interfejsów IBM MQ .

## Centralne repozytorium narzędzia Maven

Narzędzie Maven to narzędzie służące do budowania aplikacji, a także udostępnia repozytorium na potrzeby przechowywania artefaktów, do których aplikacja może mieć dostęp.

Repozytorium narzędzia Maven (lub repozytorium centralne) ma strukturę, która umożliwia tworzenie różnych wersji plików, takich jak pliki JAR, które są łatwo wykrywane przy użyciu powszechnie znanego mechanizmu nazewnictwa. Narzędzia budowania mogą następnie używać tych nazw do dynamicznego pobierania zależności dla aplikacji. W definicji aplikacji, która w przypadku używania narzędzia Maven jako narzędzia budowania jest nazywana plikiem POM, należy podać nazwy zależności, a proces budowania będzie wiedział, co należy z nich zrobić.

### Pliki klienta IBM MQ

Kopie interfejsów klienta IBM MQ Java są dostępne w centralnym repozytorium w katalogu `com.ibm.mq` `GroupId`. Można znaleźć plik `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0) oraz plik `com.ibm.mq.allclient.jar` (JMS 2.0). Te pliki są zwykle używane dla programów autonomicznych. Można również znaleźć plik `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) i plik `wmq.jmsra.rar` (JMS 2.0), który jest używany na serwerach aplikacji Java EE ). Zarówno plik `jakarta.client.jar` , jak i plik `allclient.jar` zawierają znaki IBM MQ classes for JMS i IBM MQ classes for Java.

**Ważne:** Użycie formatu Apache Maven Assembly Plugin *jar-with-dependencies* do zbudowania aplikacji, która zawiera przemieszczalny plik JAR IBM MQ , nie jest obsługiwane.

W pliku `pom.xml` przetwarzanym przez komendę maven należy dodać zależności dla tych plików JAR, jak pokazano w poniższych przykładach:

- V 9.3.0 JM 3.0 V 9.3.0 Aby wyświetlić relację między kodem aplikacji i `com.ibm.mq.jakarta.client.jar`:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.jakarta.client</artifactId>
        <version>9.3.0.0</version>
    </dependency>
```

- JMS 2.0 Aby wyświetlić relację między kodem aplikacji i `com.ibm.mq.allclient.jar`:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.allclient</artifactId>
        <version>9.2.2.0</version>
    </dependency>
```

- V 9.3.0 JM 3.0 V 9.3.0 W przypadku używania adaptera zasobów Jakarta EE :

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jakarta.jmsra</artifactId>
     <version>9.3.0.0</version>
 </dependency>
```

- JMS 2.0 W przypadku używania adaptera zasobów produktu JMS 2.0 Java EE :

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jmsra</artifactId>
     <version>9.2.2.0</version>
 </dependency>
```

Przykład prostego projektu w produkcie Eclipse do uruchamiania projektu JMS zawiera IBM Developer artykuł Tworzenie aplikacji Java dla systemu MQ było łatwiejsze dzięki Maven.

## Tworzenie aplikacji w języku C++

IBM MQ udostępnia klasy C++ równoważne obiektom IBM MQ i pewne dodatkowe klasy równoważne tablicom typów danych. Udostępnia on wiele funkcji, które nie są dostępne za pośrednictwem interfejsu MQI.

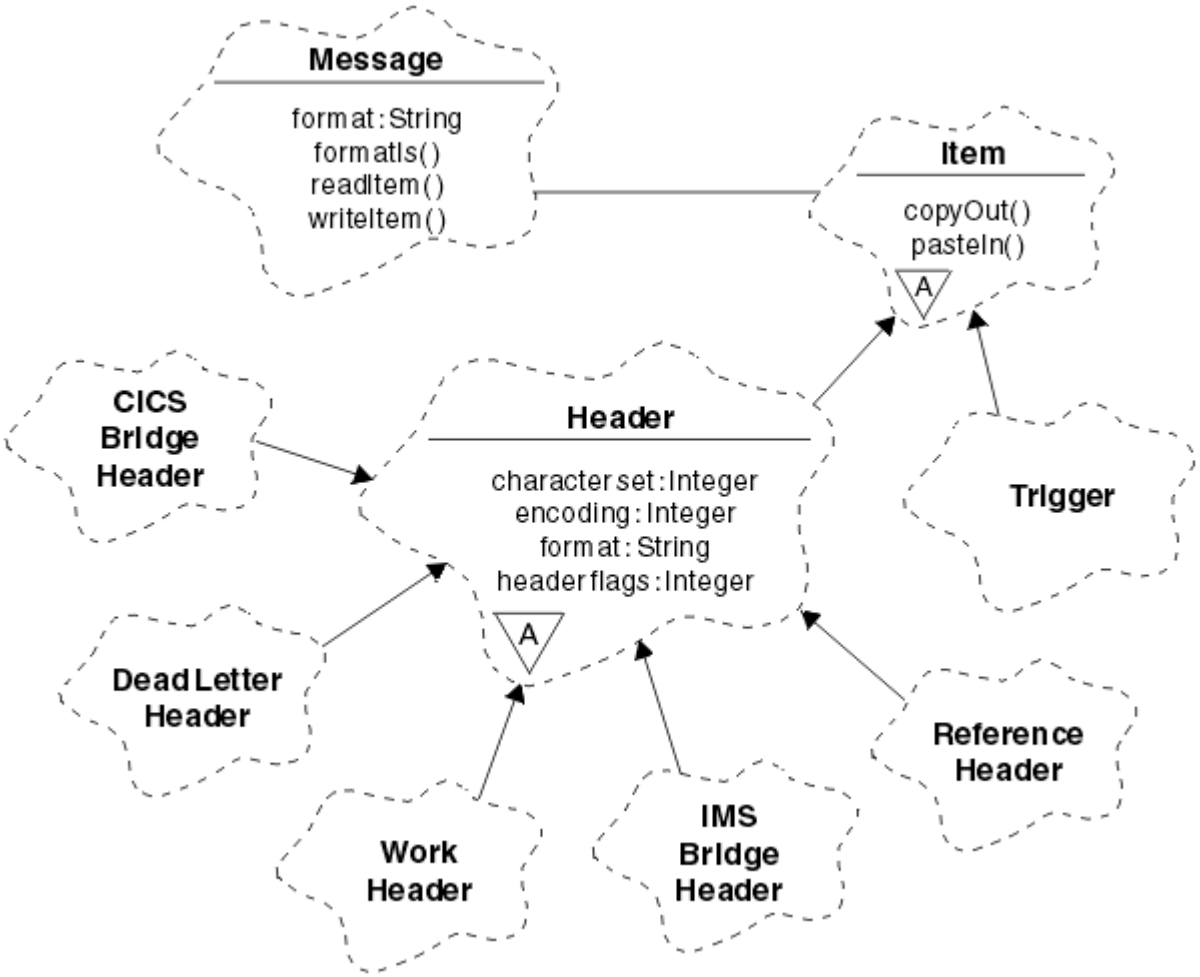
IBM WebSphere MQ 7.0, rozszerzenia interfejsów programistycznych języka IBM MQ nie są stosowane do klas języka C + +.

IBM MQ C++ udostępnia następujące funkcje:

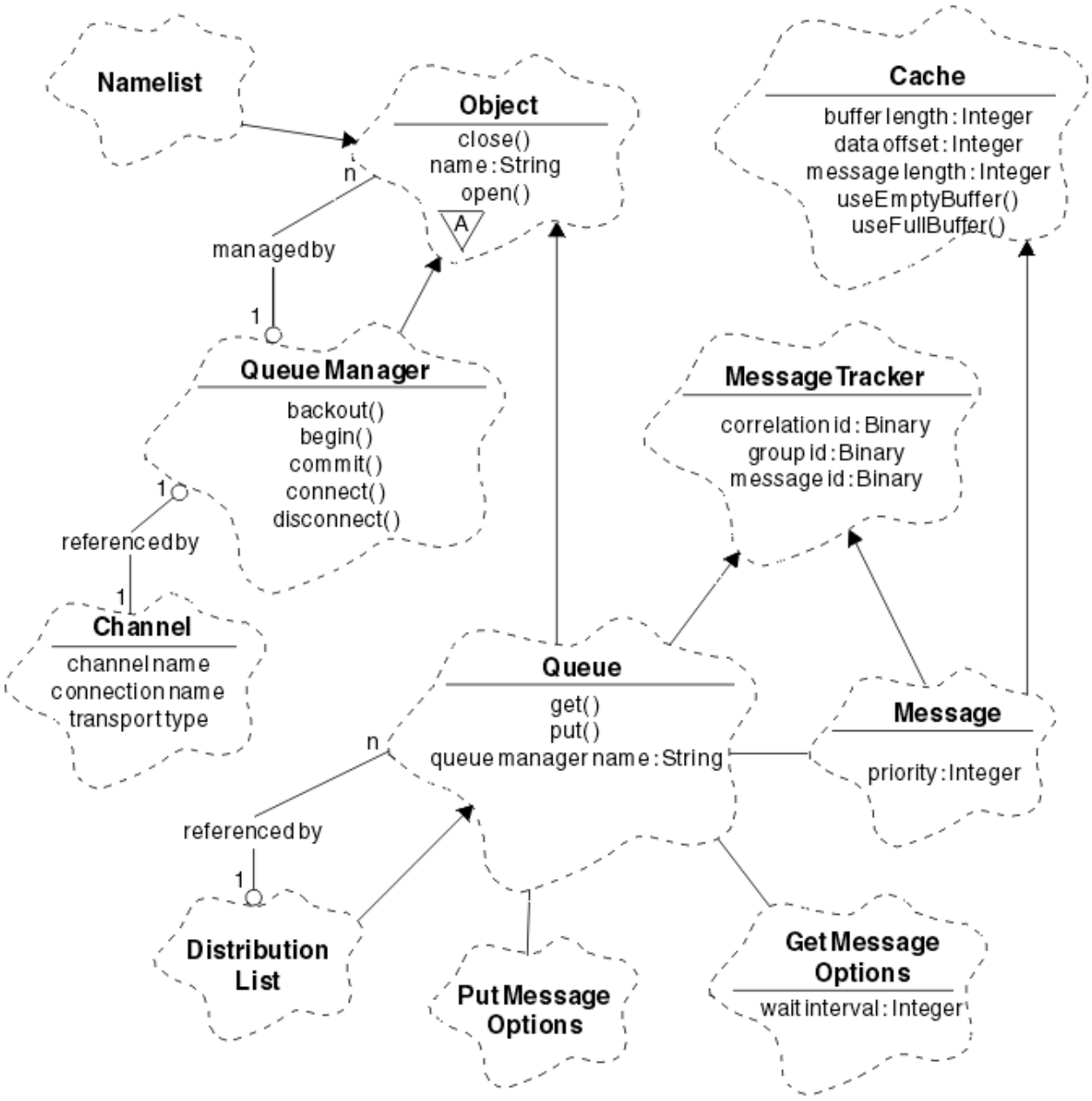
- Automatyczne inicjowanie struktur danych IBM MQ .
- Połączenie menedżera kolejek JIT i otwarcie kolejki.
- Niejawne zamknięcie kolejki i rozłączenie menedżera kolejek.

- Transmisja i odbiór z nagłówka niedostarczonego komunikatu.
- Przesyłanie i odbieranie nagłówka mostu IMS .
- Odwołanie do transmisji i odbioru nagłówka komunikatu.
- Wyzwalanie odbioru komunikatu.
- CICS bridge transmisja i odbiór nagłówka.
- Transmisja i odbiór nagłówka pracy.
- Definicja kanału klienta.

Poniższe diagramy klas Booch pokazują, że wszystkie klasy są zasadniczo równoległe do tych jednostek IBM MQ w proceduralnych obiektach MQI (na przykład przy użyciu języka C), które mają uchwyty lub struktury danych. Wszystkie klasy dziedziczą z klasy ImqError (patrz ImqError klasa C++ ), co umożliwia powiązanie warunku błędu z każdym obiektem.




*Rysunek 50. Klasy IBM MQ C++ (obsługa elementów)*




*Rysunek 51. Klasy IBM MQ C++ (zarządzanie kolejkami)*

Aby poprawnie interpretować diagramy klas Booch, należy pamiętać o następujących konwencjach:

- Metody i istotne atrybuty są wyświetlane poniżej nazwy *klasy* .
- Mały trójkąt w chmurze oznacza *klasę abstrakcyjną*.
- *Dziedziczenie* jest oznaczane strzałką do klasy nadrzędnej.
- Nieozdobiona linia między chmurami oznacza *relację kooperatywną* między klasami.
- Linia oznaczona liczbą oznacza *relację referencyjną* między dwiema klasami. Liczba wskazuje liczbę obiektów, które mogą uczestniczyć w danej relacji w dowolnym momencie.

W sygnaturach metod C++ klas zarządzania kolejkami używane są następujące klasy i typy danych (patrz sekcja Rysunek 51 na stronie 547 ) i klasy obsługi elementów (patrz Rysunek 50 na stronie 546 ):

- Klasa ImqBinary (patrz klasaImqBinary C++ ), która hermetyzuje tablice bajtów, takie jak MQBYTE24.
- Typ danych ImqBoolean , który jest zdefiniowany jako **typedef unsigned char ImqBoolean**.
- Klasa ImqString (patrz ImqString klasa C++ ), która hermetyzuje tablice znaków, takie jak MQCHAR64.

Jednostki ze strukturami danych są sumowane w odpowiednich klasach obiektów. Poszczególne pola struktury danych (patrz Skorowidz interfejsu MQI i C++ ) są dostępne za pomocą metod.

Obiekty z uchwytami znajdują się w hierarchii klas ImqObject (patrz ImqObject C++ ) i udostępnić hermetyzowane interfejsy interfejsu MQI. Obiekty tych klas wykazują inteligentne zachowanie, które może zmniejszyć liczbę wymaganych wywołań metod w odniesieniu do proceduralnego interfejsu MQI. Można na przykład ustanowić i usunąć połączenia z menedżerem kolejek zgodnie z wymaganiami lub otworzyć kolejkę z odpowiednimi opcjami, a następnie ją zamknąć.

Klasa ImqMessage (patrz ImqMessage C++ class ) Hermetyzuje strukturę danych MQMD i działa również jako punkt wstrzymujący dla danych użytkownika i *elementów* (patrz sekcja "Odczytywanie komunikatów w języku C++" na stronie 557 ). przez udostępnienie buforowanych narzędzi buforowych. Można udostępniać bufory o stałej długości dla danych użytkownika i używać ich wiele razy. Ilość danych znajdujących się w buforze może się różnić w zależności od użycia. Alternatywnie system może udostępnić bufor o elastycznej długości i zarządzać nim. Istotne znaczenie ma zarówno wielkość buforu (ilość dostępna do odbioru komunikatów), jak i ilość rzeczywiście wykorzystanych bajtów (liczba bajtów do transmisji lub liczba bajtów faktycznie odebranych).

**Pojęcia pokrewne**

Przegląd techniczny

"Przykładowe programy w języku C++" na stronie 548
Dostarczane są cztery przykładowe programy demonstrujące pobieranie i umieszczanie komunikatów.

"Uwagi dotyczące języka C++" na stronie 552
Ta kolekcja tematów zawiera szczegółowe informacje na temat użycia języka C + + oraz konwencji, które należy wziąć pod uwagę podczas pisania aplikacji korzystających z interfejsu kolejki komunikatów (Message Queue Interface-MQI).

"Przygotowywanie danych komunikatu w języku C++" na stronie 556
Dane komunikatu są przygotowywane w buforze, który może być dostarczany przez system lub aplikację. Obie metody mają pewne zalety. Podano przykłady użycia buforu.

"Tworzenie aplikacji dla składnika IBM MQ" na stronie 5
Użytkownik może tworzyć aplikacje do wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i zasobami pokrewnymi. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i środowiskach.

**Odsyłacze pokrewne**

"Budowanie programów w języku IBM MQ C++" na stronie 563
Na liście znajduje się URL obsługiwanych kompilatorów wraz z komendami używanymi do kompilowania, konsolidowania i uruchamiania programów i przykładów w języku C++ na platformach IBM MQ .

Odwołanie do interfejsu MQI i C++

Klasy IBM MQ C++

# Przykładowe programy w języku C++

Dostarczane są cztery przykładowe programy demonstrujące pobieranie i umieszczanie komunikatów.

Przykładowe programy to:

- HELLO WORLD (imqwrld.cpp)
- SPUT (imqsput.cpp)
- SGET (imqsget.cpp)
- DPUT (imqdput.cpp)

Przykładowe programy znajdują się w katalogach, które przedstawia Tabela 73 na stronie 549.

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

| *Tabela 73. Położenie programów przykładowych* | | |
|---|---|---|
| **Środowisko** | **Katalog zawierający źródło** | **Katalog zawierający zbudowane programy** |
| AIX | *MQ_INSTALLATION_PATH*/samp | *MQ_INSTALLATION_PATH*/samp/bin/ia |
| V 9.3.5 AIX | *MQ_INSTALLATION_PATH*/samp | *MQ_INSTALLATION_PATH*/samp/bin/ca (patrz uwaga “1” na stronie 549 ) |
| IBM i | `/QIBM/ProdData/mqm/samp/` | (patrz uwaga “2” na stronie 549 ) |
| Linux | *MQ_INSTALLATION_PATH*/samp | Brak |
| Windows | *MQ_INSTALLATION_PATH*\tools\cplus\samples | *MQ_INSTALLATION_PATH*\tools\cplus\przykłady \bin\vn (patrz uwaga “3” na stronie 549 ) |
| z/OS | thlqual.SCSQCPPS | |

**Uwagi:**

1. V 9.3.5 AIX Programy zbudowane przy użyciu kompilatora XLC 17 znajdują się w folderze "`ca`", podczas gdy programy zbudowane przy użyciu kompilatora XLC 16 znajdują się w folderze "`ia`".
2. IBM i Programy utworzone przy użyciu kompilatora ILE C++ dla systemu IBM i znajdują się w bibliotece QMQM. Pliki źródłowe znajdują się w pliku `/QIBM/ProdData/mqm/samp`.
3. Windows Programy zbudowane za pomocą produktu Microsoft Visual Studio Visual Studio znajdują się w katalogu *MQ_INSTALLATION_PATH*`\tools\cplus\samples\bin\vn`. Więcej informacji na temat tych kompilatorów zawiera sekcja “Budowanie programów w języku C++ w systemie Windows” na stronie 569.

## Przykładowy program HELLO WORLD (imqwrld.cpp)

Ten przykładowy program w języku C++ przedstawia sposób umieszczania i pobierania zwykłego datagramu (struktury C) za pomocą klasy ImqMessage .

Ten program przedstawia sposób umieszczania i pobierania zwykłego datagramu (struktury C) za pomocą klasy ImqMessage . W tym przykładzie wykorzystano kilka wywołań metod, korzystając z niejawnych wywołań metod, takich jak **open**, **close**i **disconnect**.

### Na wszystkich platformach z wyjątkiem platformy z/OS

Jeśli używane jest połączenie serwera z serwerem IBM MQ, wykonaj jedną z następujących procedur:

- Aby użyć istniejącej kolejki domyślnej, SYSTEM.DEFAULT.LOCAL.QUEUE, uruchom program **`imqwrlds`** bez przekazywania żadnych parametrów
- Aby użyć tymczasowej dynamicznie przypisanej kolejki, uruchom komendę **`imqwrlds`** , przekazując nazwę domyślnej kolejki modelowej SYSTEM.DEFAULT.MODEL.QUEUE.

Jeśli używane jest połączenie klienckie z produktem IBM MQ, wykonaj jedną z następujących procedur:

- Skonfiguruj zmienną środowiskową MQSERVER (więcej informacji na ten temat zawiera sekcja *MQSERVER* ) i uruchom program **imqwrldc**.
- Uruchom komendę **imqwrldc** , przekazując jako parametry **queue-name**, **queue-manager-name**i **channel-definition**, gdzie typowym **channel-definition** może być SYSTEM.DEF.SVRCONN/TCP/*nazwa_hosta* (1414)

## wł.z/OS

z/OS

Utwórz i uruchom zadanie wsadowe przy użyciu przykładowego kodu JCL **imqwrldr**.

Więcej informacji na ten temat zawiera sekcja z/OS Batch, RRS Batch i CICS .

## Kod przykładowy

```
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 ;
}
```

## Przykładowe programy SPUT (imqsput.cpp) i SGET (imqsget.cpp)

Programy w języku C++ umieszczają komunikaty w nazwanej kolejce i pobierają z niej komunikaty.

W tych przykładach przedstawiono użycie następujących klas:

- ImqError (patrz ImqError klasa C++ )
- ImqMessage (patrz ImqMessage klasa C++ )

- ImqObject (patrz ImqObject klasa C++ )
- ImqQueue (patrz ImqQueue klasa C++ )
- ImqQueueManager (patrz ImqQueueklasa menedżera C++ )

Postępuj zgodnie z odpowiednimi instrukcjami, aby uruchomić programy.

### Na wszystkich platformach z wyjątkiem platformy z/OS

1. Uruchom komendę **imqsputs** *nazwa_kolejki*.
2. Wpisz wiersze tekstu w konsoli. Te wiersze są umieszczane jako komunikaty w określonej kolejce.
3. Wprowadź pusty wiersz, aby zakończyć wprowadzanie.
4. Uruchom komendę **imqsgets** *nazwa_kolejki* , aby pobrać wszystkie wiersze i wyświetlić je na konsoli.

z/OS Więcej informacji zawiera sekcja “Budowanie programów w języku C++ w systemie z/OS Batch, RRS Batch i CICS” na stronie 571.

### wł.z/OS

z/OS

1. Utwórz i uruchom zadanie wsadowe przy użyciu przykładowego kodu JCL **imqsputr**. Komunikaty są odczytywane z zestawu danych SYSIN.
2. Utwórz i uruchom zadanie wsadowe przy użyciu przykładowego kodu JCL **imqsgetr**. Komunikaty są pobierane z kolejki i wysyłane do zestawu danych SYSPRINT.

### Przykładowy program DPUT (imqdput.cpp)

Ten przykładowy program C++ umieszcza komunikaty na liście dystrybucyjnej składającej się z dwóch kolejek.

Komenda DPUT pokazuje użycie klasy listy ImqDistribution(patrz sekcja ImqDistributionKlasa listy C++ ). Ten przykład nie jest obsługiwany w systemie z/OS.

1. Uruchom komendę **imqdputs** *queue-name-1 queue-name-2* , aby umieścić komunikaty w dwóch nazwanych kolejkach.
2. Uruchom komendy **imqsgets** *queue-name-1* i **imqsgets** *queue-name-2* , aby pobrać komunikaty z tych kolejek.

## Uwagi dotyczące języka C++

Ta kolekcja tematów zawiera szczegółowe informacje na temat użycia języka C + + oraz konwencji, które należy wziąć pod uwagę podczas pisania aplikacji korzystających z interfejsu kolejki komunikatów (Message Queue Interface-MQI).

### Pliki nagłówkowe C++

Pliki nagłówkowe są udostępniane jako część definicji interfejsu MQI, aby ułatwić pisanie aplikacji IBM MQ w języku C + +.

Te pliki nagłówkowe zostały podsumowane w poniższej tabeli.

*Tabela 74. Pliki nagłówkowe C/C++*

| **Nazwa pliku** | **Spis treści** |
|---|---|
| IMQI.HPP | Klasy MQI języka C + + (w tym CMQC.H i IMQTYPE.H) |
| IMQTYPE.H | Definiuje typ danych **ImqBoolean** . |
| CMQC.H | Struktury danych MQI i stałe manifestu |

Aby poprawić przenośność aplikacji, należy zakodować nazwę pliku nagłówkowego małymi literami w dyrektywie preprocesora **#include** :

```
#include <imqi.hpp> // C++ classes
```

## Metody i atrybuty języka C++

Nazwy metod zawierają wielkie i małe litery. Do parametrów i wartości zwracanych mają zastosowanie różne uwagi. Dostęp do atrybutów można uzyskać za pomocą metod set i get (w zależności od potrzeb).

Parametry metod, które są typu *const* , służą tylko do wprowadzania danych. Parametry z sygnaturami, w tym wskaźnik (*) lub odwołanie (&) są przekazywane przez referencję. Wartości zwracane, które nie zawierają wskaźnika lub odwołania, są przekazywane według wartości; w przypadku zwracanych obiektów są to nowe obiekty, które stają się odpowiedzialne za program wywołujący.

Niektóre sygnatury metod zawierają elementy, które przyjmują wartości domyślne, jeśli nie są określone. Takie elementy są zawsze na końcu sygnatur i są oznaczane znakiem równości (=); wartość po znaku równości wskazuje wartość domyślną, która ma zastosowanie w przypadku pominięcia elementu.

Wszystkie nazwy metod w tych klasach zawierają wielkie i małe litery. Każde słowo, z wyjątkiem pierwszego w nazwie metody, rozpoczyna się wielką literą. Skróty nie są używane, chyba że ich znaczenie jest powszechnie zrozumiałe. Używane skróty to *id* (dla tożsamości) i *sync* (dla synchronizacji).

Dostęp do atrybutów obiektu można uzyskać za pomocą metod set i get. Metoda set rozpoczyna się od słowa *set* ; Metoda get nie ma przedrostka. Jeśli atrybut jest *tylko do odczytu*, nie ma metody set.

Atrybuty są inicjowane do poprawnych stanów podczas tworzenia obiektu, a stan obiektu jest zawsze spójny.

## Typy danych w języku C++

Wszystkie typy danych są definiowane przez instrukcję **typedef** języka C.

Typ **ImqBoolean** jest zdefiniowany jako **znak bez znaku** w parametrze IMQTYPE.H i może mieć wartości TRUE i FALSE. Zamiast tablic **MQBYTE** można używać obiektów klasy **ImqBinary** i obiektów klasy **ImqString** zamiast **char ***. Wiele metod zwraca obiekty zamiast wskaźników **char** lub **MQBYTE** , aby ułatwić zarządzanie pamięcią masową. Wszystkie zwracane wartości stają się odpowiedzialne za program wywołujący, a w przypadku zwróconego obiektu pamięć można usunąć.

## Manipulowanie łańcuchami binarnymi w języku C++

Łańcuchy danych binarnych są deklarowane jako obiekty klasy **ImqBinary** . Obiekty tej klasy można kopiować, porównywać i ustawiać przy użyciu znanych operatorów języka C. Udostępniono przykładowy kod.

Poniższy przykładowy kod przedstawia operacje na łańcuchu binarnym:

```
#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.
...
```

## Manipulowanie łańcuchami znaków w języku C++

Dane znakowe są często zwracane w obiektach klasy **ImqString** , które mogą być rzutowane na **char *** za pomocą operatora konwersji. Klasa ImqString zawiera metody ułatwiające przetwarzanie łańcuchów znaków.

Jeśli dane znakowe są akceptowane lub zwracane za pomocą metod MQI C++, dane znakowe są zawsze kończone znakiem o kodzie zero i mogą mieć dowolną długość. Jednak niektóre ograniczenia są narzucane przez system IBM MQ , co może spowodować obcinanie informacji. Aby ułatwić zarządzanie pamięcią masową, dane znakowe są często zwracane w obiektach klasy **ImqString** . Te obiekty mogą być rzutowane na **char *** za pomocą udostępnionego operatora konwersji i używane do celów *tylko do odczytu* w wielu sytuacjach, w których wymagany jest znak **char *** .

**Uwaga:** Wynik konwersji **char *** z obiektu klasy **ImqString** może być pusty.

Chociaż funkcje języka C mogą być używane w **char ***, istnieją specjalne metody klasy **ImqString** , które są preferowane; **długość operatora** () jest odpowiednikiem **strlen** i **storage** () wskazuje pamięć przydzieloną dla danych znakowych.

## Początkowy stan obiektów w języku C++

Wszystkie obiekty mają spójny stan początkowy odzwierciedlony przez ich atrybuty. Wartości początkowe są definiowane w opisach klas.

## Korzystanie z języka C z języka C++

Jeśli używane są funkcje języka C z programu w języku C + +, należy dołączyć odpowiednie nagłówki.

Poniższy przykład przedstawia plik `string.h` dołączony do programu w języku C++:

```
extern "C" {
#include <string.h>
}
```

## Konwencje notacyjne C++

W tym przykładzie przedstawiono sposób wywoływania metod i deklarowania parametrów.

W tym przykładzie kodu użyto metod i parametrów **ImqBoolean ImqQueue**:: **get** ( **ImqMessage &** *msg* )

Zadeklaruj i użyj parametrów w następujący sposób:

```
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( );

...
}
```

## Operacje niejawne w języku C++

Aby spełnić wymagania wstępne dotyczące pomyślnego wykonania metody, można wykonać kilka operacji niejawnie ( *w samą porę*). Są to operacje połączenia, otwarcia, ponownego otwarcia, zamknięcia i rozłączenia. Przy użyciu atrybutów klasy można sterować połączeniem i otwarciem niejawnego zachowania.

### Połączenie

Obiekt menedżera ImqQueuejest automatycznie łączony dla każdej metody, która powoduje wywołanie interfejsu MQI (patrz Skorowidz interfejsu MQI i C++ ).

### Otwórz

Obiekt ImqObject jest otwierany automatycznie dla każdej metody, której wynikiem jest wywołanie MQGET, MQINQ, MQPUT lub MQSET. Użyj metody **openFor** , aby określić jedną lub więcej odpowiednich wartości **opcji otwierania** .

### Otwórz ponownie

Obiekt ImqObject jest ponownie otwierany automatycznie dla każdej metody, która powoduje wywołanie MQGET, MQINQ, MQPUT lub MQSET, gdy obiekt jest już otwarty, ale istniejące **opcje otwierania** nie są odpowiednie, aby umożliwić pomyślne wywołanie MQI. Obiekt jest tymczasowo zamykany przy użyciu tymczasowej wartości **opcji zamykania** (MQCO_NONE). Użyj metody **openFor** , aby dodać odpowiednią**open option (opcja otwarcia)**.

Ponowne otwarcie może spowodować problemy w określonych okolicznościach:

- Tymczasowa kolejka dynamiczna jest niszczona po jej zamknięciu i nie można jej ponownie otworzyć.
- Kolejka otwarta na wyłączne wejście (jawnie lub domyślnie) może być dostępna dla innych osób w oknie możliwości podczas zamykania i ponownego otwierania.
- Pozycja kursora przeglądania jest tracona po zamknięciu kolejki. Ta sytuacja nie uniemożliwia zamknięcia i ponownego otwarcia, ale uniemożliwia późniejsze użycie kursora do czasu ponownego użycia komendy MQGMO_BROWSE_FIRST.
- Kontekst ostatniego pobranego komunikatu zostanie utracony po zamknięciu kolejki.

Jeśli wystąpi taka sytuacja lub można ją przewidzieć, należy unikać ponownego otwierania, jawnie ustawiając odpowiednie **opcje otwierania** przed otwarciem obiektu (jawnie lub niejawnie).

Jawne ustawienie **opcji otwierania** dla złożonych sytuacji obsługi kolejek powoduje zwiększenie wydajności i pozwala uniknąć problemów związanych z używaniem opcji ponownego otwierania.

### Zamknij

Obiekt ImqObject jest zamykany automatycznie w każdym punkcie, w którym stan obiektu przestaje działać, na przykład w przypadku zerwania odwołania do połączenia ImqObject lub zniszczenia obiektu ImqObject .

### Rozłącz

Menedżer ImqQueuejest odłączany automatycznie w każdym punkcie, w którym połączenie nie jest już dostępne, na przykład w przypadku zerwania odwołania do połączenia ImqObject lub zniszczenia obiektu menedżera ImqQueue.

## Łańcuchy binarne i znakowe w języku C++

Klasa ImqString hermetyzuje tradycyjny format danych *char** . Klasa ImqBinary hermetyzuje binarną tablicę bajtów. Niektóre metody, które ustawiają dane znakowe, mogą je obciąć.

Metody ustawiające znak ( **char *** ) Dane zawsze zawierają kopię danych, ale niektóre metody mogą obciąć kopię, ponieważ pewne limity są narzucane przez IBM MQ.

Klasa ImqString (patrz ImqString C++ class ) Hermetyzuje tradycyjny **znak *** i zapewnia obsługę następujących funkcji:

- Porównanie
- Konkatenacja
- Kopiowanie

- Konwersja z liczby całkowitej na tekst i z tekstu na liczbę całkowitą
- Wyodrębnianie leksemu (słowa)
- Konwersja wielkich liter

Klasa ImqBinary (patrz ImqBinary C++ class ) hermetyzuje binarne tablice bajtów o dowolnej wielkości. W szczególności jest on używany do przechowywania następujących atrybutów:

- **token rozliczania** (MQBYTE32)
- **znacznik połączenia** (MQBYTE128)
- **Identyfikator korelacji** (MQBYTE24)
- **token narzędzia** (MQBYTE8)
- **Identyfikator grupy** (MQBYTE24)
- **ID instancji** (MQBYTE24)
- **ID komunikatu** (MQBYTE24)
- **Znacznik komunikatu** (MQBYTE16)
- **Identyfikator instancji transakcji** (MQBYTE16)

Gdzie te atrybuty należą do obiektów następujących klas:

- ImqCICSBridgeHeader (patrz ImqCICSBridgeHeader klasa C++ )
- ImqGetMessageOptions (patrz ImqGetMessageOptions klasa C++ )
- ImqIMSBridgeHeader (patrz ImqIMSBridgeHeader klasa C++ )
- ImqMessageTracker (patrz ImqMessageTracker C++ class )
- ImqQueueManager (patrz ImqQueueklasa menedżera C++ )
- Nagłówek ImqReference(patrz ImqReferenceHeader C++ class )
- Nagłówek ImqWork(patrz ImqWorkHeader C++ class )

Klasa ImqBinary zapewnia również obsługę porównywania i kopiowania.

### Nieobsługiwane funkcje w języku C++

Klasy i metody języka C++ IBM MQ są niezależne od platformy IBM MQ . Dlatego mogą one oferować niektóre funkcje, które nie są obsługiwane na niektórych platformach.

W przypadku próby użycia funkcji na platformie, na której nie jest ona obsługiwana, funkcja jest wykrywana przez produkt IBM MQ , ale nie przez powiązania języka C + +. Program IBM MQ zgłasza błąd, podobnie jak każdy inny błąd MQI.

## Przesyłanie komunikatów w języku C++

Ta kolekcja tematów zawiera szczegółowe informacje na temat przygotowywania, odczytywania i zapisywania komunikatów w języku C + +.

### Przygotowywanie danych komunikatu w języku C++

Dane komunikatu są przygotowywane w buforze, który może być dostarczany przez system lub aplikację. Obie metody mają pewne zalety. Podano przykłady użycia buforu.

Podczas wysyłania komunikatu dane komunikatu są najpierw przygotowywane w buforze zarządzanym przez obiekt ImqCache (patrz klasa ImqCache C++ ). Bufor jest powiązany (przez dziedziczenie) z każdym obiektem ImqMessage (patrz klasa języka C++ImqMessage ): może być dostarczany przez aplikację (przy użyciu metody **useEmptyBuffer** lub **useFullBuffer** ) lub automatycznie przez system. Zaletą aplikacji dostarczającej bufor komunikatów jest brak konieczności kopiowania danych w wielu przypadkach, ponieważ aplikacja może bezpośrednio korzystać z przygotowanych obszarów danych. Wadą jest to, że podany bufor ma stałą długość.

Bufor można ponownie wykorzystać, a liczba przesłanych bajtów może być za każdym razem różna, za pomocą metody **setMessageLength** przed transmisją.

Liczba dostępnych bajtów jest automatycznie podawana przez system, a dane mogą być kopiowane do buforu komunikatów za pomocą na przykład metody ImqCache **write** lub metody ImqMessage **writeItem** . Bufor komunikatów zwiększa się zgodnie z potrzebą. Wraz ze wzrostem wielkości buforu nie występuje utrata wcześniej zapisanych danych. Duży lub wieloczęściowy komunikat można zapisać w kolejnych częściach.

W poniższych przykładach przedstawiono uproszczone wysyłanie komunikatów.

1. Użyj przygotowanych danych w buforze dostarczonym przez użytkownika

```
char szBuffer[ ] = "Hello world" ;

msg.useFullBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
```

2. Użyj przygotowanych danych w buforze dostarczonym przez użytkownika, w którym wielkość buforu przekracza wielkość danych

```
char szBuffer[ 24 ] = "Hello world" ;

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.setMessageLength( 12 );
```

3. Kopiowanie danych do buforu użytkownika

```
char szBuffer[ 12 ];

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

4. Kopiowanie danych do buforu dostarczonego przez system

```
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

5. Kopiowanie danych do buforu dostarczonego przez system przy użyciu obiektów (obiekty ustawiają format komunikatu, a także treść)

```
ImqString strText( "Hello world" );

msg.writeItem( strText );
```

## Odczytywanie komunikatów w języku C++

Bufor może być dostarczany przez aplikację lub system. Dostęp do danych można uzyskać bezpośrednio z buforu lub odczytywać sekwencyjnie. Istnieje klasa równoważna każdemu typowi komunikatu. Podano przykładowy kod.

Podczas odbierania danych aplikacja lub system może dostarczyć odpowiedni bufor komunikatów. Ten sam bufor może być używany zarówno dla wielu transmisji, jak i dla wielu odbiorów dla konkretnego obiektu ImqMessage . Jeśli bufor komunikatów jest dostarczany automatycznie, zwiększa się w celu dostosowania do długości odbieranych danych. Jednak bufor komunikatów dostarczony przez aplikację może nie być wystarczająco duży, aby pomieścić odebrane dane. W zależności od opcji używanych do odbierania komunikatów może wystąpić obcięcie lub niepowodzenie.

Dostęp do danych przychodzących można uzyskać bezpośrednio z buforu komunikatów, w którym to przypadku długość danych wskazuje łączną ilość danych przychodzących. Alternatywnie dane przychodzące mogą być odczytywane sekwencyjnie z buforu komunikatów. W tym przypadku wskaźnik

danych odnosi się do następnego bajtu danych przychodzących, a wskaźnik danych i długość danych są aktualizowane za każdym razem, gdy dane są odczytywane.

*Elementy* to elementy komunikatu, wszystkie w obszarze użytkownika buforu komunikatów, które muszą być przetwarzane sekwencyjnie i oddzielnie. Oprócz zwykłych danych użytkownika element może być nagłówkiem niedostarczonego komunikatu lub komunikatem wyzwalacza. Elementy są zawsze powiązane z formatami komunikatów; formaty komunikatów **nie** są zawsze powiązane z elementami.

Dla każdego elementu istnieje klasa obiektu, która odpowiada rozpoznawalnemu formatowi komunikatu IBM MQ . Istnieje jeden dla nagłówka niedostarczonego komunikatu i jeden dla komunikatu wyzwalacza. Brak klasy obiektu dla danych użytkownika. Oznacza to, że po wyczerpaniu rozpoznawalnych formatów przetwarzanie pozostałej części pozostaje w programie użytkowym. Klasy dla danych użytkownika można zapisać, specjalizując się w klasie ImqItem .

W poniższym przykładzie przedstawiono komunikat, który uwzględnia pewną liczbę potencjalnych pozycji, które mogą poprzedzać dane użytkownika, w sytuacji wyimaginowanej. Dane użytkownika, które nie są elementami, są definiowane jako wszystko, co występuje po elementach, które można zidentyfikować. Bufor automatyczny (domyślnie) jest używany do przechowywania dowolnej ilości danych komunikatu.

```
ImqQueue queue ;
ImqMessage msg ;

if ( queue.get( msg ) ) {

  /* Process all items of data in the message buffer. */
  do while ( msg.dataLength( ) ) {
    ImqBoolean bFormatKnown = FALSE ;
    /* There remains unprocessed data in the message buffer. */

    /* Determine what kind of item is next. */

    if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
      ImqDeadLetterHeader header ;
      /* The next item is a dead-letter header.                */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( header ) ) {
        /* The dead-letter header has been extricated from the */
        /* buffer and transformed into a dead-letter object.   */
        /* The encoding and character set of the dead-letter   */
        /* object itself are MQENC_NATIVE and MQCCSI_Q_MGR.    */
        /* The encoding and character set from the dead-letter */
        /* header have been copied to the message attributes   */
        /* to reflect any remaining data in the buffer.        */

        /* Process the information in the dead-letter object.  */
        /* Note that the encoding and character set have       */
        /* already been processed.                             */
        ...
      }
      /* There might be another item after this, */
      /* or just the user data.                  */
    }
    if ( msg.formatIs( MQFMT_TRIGGER ) ) {
      ImqTrigger trigger ;
      /* The next item is a trigger message.                   */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;
      if ( msg.readItem( trigger ) ) {

        /* The trigger message has been extricated from the */
        /* buffer and transformed into a trigger object.    */
        /* Process the information in the trigger object. */
        ...
      }

      /* There is usually nothing after a trigger message. */
    }

    if ( msg.formatIs( FMT_USERCLASS ) ) {
      UserClass object ;
      /* The next item is an item of a user-defined class.     */
```

```
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( object ) ) {
        /* The user-defined data has been extricated from the */
        /* buffer and transformed into a user-defined object. */

        /* Process the information in the user-defined object. */
        ...
      }

      /* Continue looking for further items. */
    }
    if ( ! bFormatKnown ) {
      /* There remains data that is not associated with a specific*/
      /* item class.                                               */
      char * pszDataPointer = msg.dataPointer( );       /* Address.*/
      int iDataLength = msg.dataLength( );              /* Length. */

      /* The encoding and character set for the remaining data are */
      /* reflected in the attributes of the message object, even   */
      /* if a dead-letter header was present.                      */
      ...

    }

  }
}
```

W tym przykładzie FMT_USERCLASS jest stałą reprezentującą 8-znakową nazwę formatu powiązaną z obiektem klasy `UserClassi` zdefiniowaną przez aplikację.

Język `UserClass` pochodzi z klasy ImqItem (patrz ImqItem C++ class ) i implementuje wirtualne metody **copyOut** i **pasteIn** z tej klasy.

Następne dwa przykłady przedstawiają kod z klasy ImqDeadLetterHeader (patrz ImqDeadLetterHeader C++ class ). W pierwszym przykładzie przedstawiono niestandardowy komunikat obudowany- *pisanie* kodu.

```
// 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 ;
}
```

W drugim przykładzie przedstawiono niestandardowy komunikat obudowany- *odczyt* kodu.

```
// 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 ;
}
```

W przypadku automatycznego buforu pamięć masowa buforu jest *ulotna*. Oznacza to, że dane buforu mogą być przechowywane w innym miejscu fizycznym po każdym wywołaniu metody **get** . Dlatego przy każdym odwołaniu do danych buforu należy używać metod **bufferPointer** lub **dataPointer** w celu uzyskania dostępu do danych komunikatu.

Użytkownik może chcieć, aby program odkłada na bok stały obszar do odbierania danych komunikatu. W takim przypadku należy wywołać metodę **useEmptyBuffer** przed użyciem metody **get** .

Użycie stałego, nieautomatycznego obszaru ogranicza komunikaty do maksymalnej wielkości, dlatego ważne jest, aby rozważyć użycie opcji MQGMO_ACCEPT_TRUNCATED_MSG obiektu ImqGetMessageOptions . Jeśli ta opcja nie jest określona (wartość domyślna), można oczekiwać kodu przyczyny MQRC_TRUNCATED_MSG_FAILED. Jeśli ta opcja jest określona, w zależności od projektu aplikacji może być oczekiwany kod przyczyny MQRC_TRUNCATED_MSG_ACCEPTED.

W następnym przykładzie przedstawiono sposób użycia stałego obszaru pamięci masowej do odbierania komunikatów:

```
char * pszBuffer = new char[ 100 ];

msg.useEmptyBuffer( pszBuffer, 100 );
gmo.setOptions( MQGMO_ACCEPT_TRUNCATED_MSG );
queue.get( msg, gmo );

delete [ ] pszBuffer ;
```

W tym fragmencie kodu bufor może być zawsze adresowany bezpośrednio za pomocą komendy *pszBuffer*, a nie za pomocą metody **bufferPointer** . Jednak lepszym rozwiązaniem jest użycie metody **dataPointer** w celu uzyskania dostępu ogólnego przeznaczenia. Aplikacja (a nie obiekt klasy ImqCache ) musi usunąć zdefiniowany przez użytkownika (nieautomatyczny) bufor.

**Uwaga:** Określenie wskaźnika pustego i zerowej długości za pomocą buforu **useEmpty** nie oznacza buforu o stałej długości o zerowej długości, jak można by oczekiwać. Ta kombinacja jest interpretowana jako żądanie zignorowania poprzedniego buforu zdefiniowanego przez użytkownika, a zamiast tego wraca do użycia buforu automatycznego.

## Zapisywanie komunikatu do kolejki niedostarczonych komunikatów w języku C++

Przykładowy kod programu do zapisywania komunikatu w kolejce niedostarczonych komunikatów.

Typowym przypadkiem komunikatu wieloczęściowego jest komunikat zawierający nagłówek niedostarczonego komunikatu. Dane z komunikatu, którego nie można przetworzyć, są dołączane do nagłówka niedostarczonego komunikatu.

```
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 );
```

## Zapisywanie komunikatu do mostu IMS w języku C++

Przykładowy kod programu do zapisywania komunikatu do mostu IMS .

Komunikaty wysyłane do mostu IBM MQ - IMS mogą używać specjalnego nagłówka. Nagłówek mostu IMS jest dodawany do zwykłych danych komunikatu.

```
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 );
```

## Zapisywanie komunikatu do CICS bridge w języku C++

Przykładowy kod programu do zapisywania komunikatu w CICS bridge.

Komunikaty wysyłane do IBM MQ for z/OS przy użyciu CICS bridge wymagają specjalnego nagłówka. Nagłówek CICS bridge jest dodawany do zwykłych danych komunikatu.

```
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 );
```

## Pisanie komunikatu z nagłówkiem roboczym w języku C++

Przykładowy kod programu do zapisywania komunikatu przeznaczonego dla kolejki zarządzanej przez program z/OS Workload Manager.

Komunikaty wysyłane do produktu IBM MQ for z/OS, które są przeznaczone dla kolejki zarządzanej przez program z/OS Workload Manager, wymagają specjalnego nagłówka. Nagłówek pracy jest dodawany do zwykłych danych komunikatu.

```
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 );
```

# Budowanie programów w języku IBM MQ C++

Na liście znajduje się URL obsługiwanych kompilatorów wraz z komendami używanymi do kompilowania, konsolidowania i uruchamiania programów i przykładów w języku C++ na platformach IBM MQ .

Listę kompilatorów dla każdej obsługiwanej platformy i wersji systemu IBM MQzawiera sekcja Wymagania systemowe produktu IBM MQ.

Komenda, którą należy skompilować i skonsolidować program IBM MQ C++, zależy od instalacji i wymagań. Poniższe przykłady przedstawiają typowe komendy kompilowania i konsolidowania dla niektórych kompilatorów przy użyciu domyślnej instalacji produktu IBM MQ na wielu platformach.

## AIX Budowanie programów w języku C++ w systemie AIX

Budowanie programów w języku IBM MQ C++ w systemie AIX przy użyciu kompilatora XL C Enterprise Edition .

V 9.3.5 Więcej informacji na temat różnych odwzorowań opcji kompilatora między kompilatorami XLC 16 i XLC 17 zawiera sekcja Odwzorowywanie opcji.

V 9.3.5 Deprecated Obsługa kompilatora XL C/C++ for AIX 16 w systemie AIX jest nieaktualna od wersji IBM MQ 9.3.5.

### Klient

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

**32-bitowa aplikacja niewątkowa**

```
xlC -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

**32-bitowa aplikacja wielowątkowa**

```
xlC_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

**64-bitowa aplikacja niewątkowa**

```
xlC -q64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

**Aplikacja z wątkami 64-bitowymi**

```
xlC_r -q64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

**V 9.3.5 32-bitowa aplikacja niegwintowana (XLC 17)**

```
ibm-clang++_r -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca -limqb23ca -lmqic
```

**V 9.3.5 32-bitowa aplikacja gwintowana (XLC 17)**

```
ibm-clang++_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca_r -limqb23ca_r -lmqic_r
```

**V 9.3.5 Aplikacja niegwintowana w trybie 64-bitowym (XLC 17)**

```
ibm-clang++_r -m64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca -limqb23ca -lmqic
```

**V 9.3.5 Aplikacja z wątkami 64-bitowymi (XLC 17)**

```
ibm-clang++_r -m64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca_r -limqb23ca_r -lmqic_r
```

**Serwer**

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

**32-bitowa aplikacja niewątkowa**

```
xlC -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

**32-bitowa aplikacja wielowątkowa**

```
xlC_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

**64-bitowa aplikacja niewątkowa**

```
xlC -q64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

**Aplikacja z wątkami 64-bitowymi**

```
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
```

**V 9.3.5 32-bitowa aplikacja niegwintowana (XLC 17)**

```
ibm-clang++_r -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca -limqb23ca -lmqm
```

**V 9.3.5 32-bitowa aplikacja gwintowana (XLC 17)**

```
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
```

### V 9.3.5 Aplikacja niegwintowana w trybie 64-bitowym (XLC 17)

```
ibm-clang++_r -m64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca -limqb23ca -lmqm
```

### V 9.3.5 Aplikacja z wątkami 64-bitowymi (XLC 17)

```
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
```

## IBM i Budowanie programów w języku C++ w systemie IBM i

Budowanie programów w języku IBM MQ C++ w systemie IBM i przy użyciu kompilatora ILE C + +.

IBM ILE C++ for IBM i jest rodzimym kompilatorem programów w języku C + +. Poniższe instrukcje opisują sposób użycia tego kompilatora do tworzenia aplikacji IBM MQ C++ przy użyciu aplikacji *Hello World!* Przykładowy program IBM MQ jako przykład.

1. Zainstaluj kompilator ILE C++ for IBM i zgodnie z instrukcjami w sekcji *Read Me first!* Podręcznik, który towarzyszy produktowi.
2. Upewnij się, że biblioteka QCXXN znajduje się na liście bibliotek.
3. Utwórz przykładowy program HELLO WORLD:
   a. Utwórz moduł:

   ```
   CRTCPPMOD MODULE(MYLIB/IMQWRLD) +
   SRCSTMF('/QIBM/ProdData/mqm/samp/imqwrld.cpp') +
   INCDIR('/QIBM/ProdData/mqm/inc') DFTCHAR(*SIGNED) +
   TERASPACE(*YES)
   ```

   Kod źródłowy programów przykładowych w języku C++ znajduje się w katalogu `/QIBM/ProdData/mqm/samp` , a pliki włączane w katalogu `/QIBM/ProdData/mqm/inc`.

   Źródło można również znaleźć w bibliotece `SRCFILE(QCPPSRC/LIB) SRCMBR(IMQWRLD)`.

   b. Powiąż ten obiekt z programami usługowymi dostarczonymi przez IBM MQ, aby utworzyć obiekt programu:

   ```
   CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
   BNDSRVPGM(QMQM/IMQB23I4 QMQM/IMQS23I4)
   ```

   Aby zbudować aplikację wielowątkową, należy użyć wielobieżnych programów usługowych:

   ```
   CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
   BNDSRVPGM(QMQM/IMQB23I4[_R] QMQM/IMQS23I4[_R])
   ```

   c. Uruchom przykładowy program HELLO WORLD, korzystając z systemu SYSTEM.DEFAULT.LOCAL.QUEUE:

   ```
   CALL PGM(MYLIB/IMQWRLD)
   ```

## Linux Budowanie programów w języku C++ w systemie Linux

Budowanie programów w języku IBM MQ C++ w systemie Linux przy użyciu kompilatora GNU g + +.

### System p

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

**Klient:System p**

**32-bitowa aplikacja niewątkowa**

```
g++ -m32 -o imqsputc_32 imqsput.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
```

**32-bitowa aplikacja wielowątkowa**

```
g++ -m32 -o imqsputc_r32 imqsput.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
```

**64-bitowa aplikacja niewątkowa**

```
g++ -m64 -o imqsputc_64 imqsput.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
```

**Aplikacja z wątkami 64-bitowymi**

```
g++ -m64 -o imqsputc_r64 imqsput.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
```

**Serwer: System p**

**32-bitowa aplikacja niewątkowa**

```
g++ -m32 -o imqsput_32 imqsput.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
```

**32-bitowa aplikacja wielowątkowa**

```
g++ -m32 -o imqsput_r32 imqsput.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
```

**64-bitowa aplikacja niewątkowa**

```
g++ -m64 -o imqsput_64 imqsput.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
```

**Aplikacja z wątkami 64-bitowymi**

```
g++ -m64 -o imqsput_r64 imqsput.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_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

**Klient:IBM Z**

**32-bitowa aplikacja niewątkowa**

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

**32-bitowa aplikacja wielowątkowa**

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.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
```

**64-bitowa aplikacja niewątkowa**

```
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
```

**Aplikacja z wątkami 64-bitowymi**

```
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
```

**Serwer: IBM Z**

**32-bitowa aplikacja niewątkowa**

```
g++ -m31 -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
```

**32-bitowa aplikacja wielowątkowa**

```
g++ -m31 -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
```

**64-bitowa aplikacja niewątkowa**

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

**Aplikacja z wątkami 64-bitowymi**

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.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 (32-bitowy)

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

**Klient: x86-64 (32-bitowy)**

**32-bitowa aplikacja niewątkowa**

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.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
```

**32-bitowa aplikacja wielowątkowa**

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.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
```

**64-bitowa aplikacja niewątkowa**

```
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 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl -limqb23gl
-lmqic
```

**Aplikacja z wątkami 64-bitowymi**

```
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 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

**Serwer: x86-64 (32-bitowy)**

**32-bitowa aplikacja niewątkowa**

```
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 -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl -limqb23gl -lmqm
```

**32-bitowa aplikacja wielowątkowa**

```
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 -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

**64-bitowa aplikacja niewątkowa**

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.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
```

**Aplikacja z wątkami 64-bitowymi**

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.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 Budowanie programów w języku C++ w systemie Windows

Budowanie programów w języku IBM MQ C++ w systemie Windows przy użyciu kompilatora języka C + + systemu Microsoft Visual Studio .

**Ostrzeżenie:** Biblioteki dostarczane przez IBM MQ są bibliotekami dynamicznymi, a nie statycznymi. Produkt IBM MQ udostępnia coś, co jest znane jako "`import libraries`" i może być używane tylko podczas kompilacji. W przypadku środowiska wykonawczego należy użyć bibliotek dynamicznych.

W systemie IBM MQ 8.0.0 Fix Pack 4 IBM MQ dostarczany jest klient podlegający redystrybucji, zawierający biblioteki wymagane do uruchamiania aplikacji IBM MQ . Biblioteki te mogą być pakowane i redystrybuowane razem z aplikacjami klienckimi. Więcej informacji na ten temat zawiera sekcja Klienty podlegające redystrybucji w systemie Windows.

Pliki biblioteki (.lib) i pliki dll do użytku z aplikacjami 32-bitowymi są instalowane w produkcie *MQ_INSTALLATION_PATH*`/Tools/Lib`. Pliki używane z aplikacjami 64-bitowymi są instalowane w produkcie *MQ_INSTALLATION_PATH*`/Tools/Lib64`. *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

#### Klient

```
cl -MD imqsput.cpp /Feimqsputc.exe imqb23vn.lib imqc23vn.lib
```

#### Serwer

```
cl -MD imqsput.cpp /Feimqsput.exe imqb23vn.lib imqs23vn.lib
```

### Instalowanie uniwersalnego środowiska wykonawczego C

Jeśli używany jest produkt Windows 8.1 lub Windows Server 2012 R2, należy zainstalować aktualizację uniwersalnego środowiska wykonawczego C (Universal CRT) z systemu Microsoft. To środowisko wykonawcze jest częścią systemu Windows 10 i serwera Windows Server 2016.

Uniwersalna aktualizacja CRT to Microsoft aktualizacja KB3118401. Aby sprawdzić, czy ta aktualizacja jest dostępna, należy wyszukać plik o nazwie `ucrtbase.dll` w katalogu `C:\Windows\System32` . Jeśli nie, można pobrać aktualizację z następującej strony Microsoft : https://www.catalog.update.microsoft.com/Search.aspx?q=kb3118401.

Próba uruchomienia programu IBM MQ lub programu skompilowanego samodzielnie za pomocą programu Microsoft Visual Studio 2017bez zainstalowanego środowiska wykonawczego spowoduje wystąpienie błędów, takich jak następujący błąd:

```
The program can't start because api-ms-win-crt-runtime-|1-1-0.dll
is missing from your computer. Try reinstalling the program to
fix this problem.
```

### Udostępnianie środowisk wykonawczych dla programów Microsoft Visual Studio 2012

Jeśli program IBM MQ został skompilowany przy użyciu programu Microsoft Visual Studio 2012, należy pamiętać, że instalator IBM MQ nie instaluje środowisk wykonawczych oprogramowania Microsoft Visual Studio 2012 C/C + +. Jeśli poprzednia wersja produktu IBM MQ została zainstalowana na tym samym komputerze, środowiska wykonawcze Microsoft Visual Studio 2012 są dostępne w tej instalacji.

Jeśli jednak używany jest program, który został zbudowany przy użyciu systemu Microsoft Visual Studio 2012 i nie została zainstalowana poprzednia wersja systemu IBM MQ , należy wykonać jedną z następujących czynności:

- Pobierz i zainstaluj **Microsoft Visual C++ Redistributable for VisualStudio 2017 (32 and 64-bit versions)** z serwisu Microsoft.
- Zrekompiluj program z poziomem Microsoft Visual Studio 2017lub innym poziomem Microsoft Visual Studio , dla którego zainstalowane są środowiska wykonawcze.

## Biblioteki klienckie C + + utworzone przy użyciu kompilatora języka Microsoft Visual Studio 2015

IBM MQ udostępnia biblioteki klienckie C + +, które są zbudowane przy użyciu kompilatora C + + Microsoft Visual Studio 2015 oraz kompilatora C + + Microsoft Visual Studio 2017 .

Dostępne są zarówno 32-bitowe, jak i 64-bitowe wersje bibliotek IBM MQ C + +. 32-bitowe biblioteki są instalowane w folderze `bin\vs2015` , a 64-bitowe biblioteki są instalowane w folderach `bin64\vs2015` .

Domyślnie produkt IBM MQ jest skonfigurowany do używania bibliotek Microsoft Visual Studio 2017 . Aby używać bibliotek Microsoft Visual Studio 2015 , należy ustawić zmienną środowiskową `MQ_PREFIX_VS_LIBRARIES` na wartość `MQ_PREFIX_VS_LIBRARIES=vs2015` przed zainstalowaniem produktu IBM MQlub przed użyciem komendy **setmqenv** lub **setmqinst** .

## Korzystanie z bibliotek C++ o innej nazwie IBM MQ

IBM MQ udostępnia dodatkowe biblioteki klienckie C + +, które mają różne nazwy. Te biblioteki są budowane przy użyciu kompilatorów Microsoft Visual Studio 2015 i Microsoft Visual Studio 2017 C + +. Te biblioteki są udostępniane jako dodatek do istniejących bibliotek C + +, które są również budowane przy użyciu kompilatora języka C + + Microsoft Visual Studio 2017 . Ponieważ te dodatkowe biblioteki IBM MQ C++ mają różne nazwy, można uruchamiać aplikacje IBM MQ C++ zbudowane przy użyciu języka IBM MQ C++ i skompilowane przy użyciu języka Microsoft Visual Studio 2017 i wcześniejszych wersji produktu na tym samym komputerze.

Dodatkowe biblioteki Microsoft Visual Studio 2017 mają następujące nazwy:

- `imqb23vnvs2017.dll`
- `imqc23vnvs2017.dll`
- `imqs23vnvs2017.dll`
- `imqx23vnvs2017.dll`

Dodatkowe biblioteki Microsoft Visual Studio 2015 mają następujące nazwy:

- `imqb23vnvs2015.dll`
- `imqc23vnvs2015.dll`
- `imqs23vnvs2015.dll`
- `imqx23vnvs2015.dll`

Dostępne są zarówno 32-bitowe, jak i 64-bitowe wersje tych bibliotek. Biblioteki 32-bitowe są instalowane w folderze `bin` , a biblioteki 64-bitowe są instalowane w folderze `bin64` . Odpowiednie biblioteki importu są instalowane w katalogach `Tools\lib` i `Tools\lib64` .

Jeśli aplikacja używa plików `imq*vs2015.lib` , należy je skompilować przy użyciu kompilatora języka Microsoft Visual Studio 2015 . Aby uruchomić aplikacje IBM MQ C++ skompilowane przy użyciu języka Microsoft Visual Studio 2015lub aplikacje skompilowane przy użyciu wcześniejszej wersji produktu na tym samym komputerze, zmienna środowiskowa PATH musi być poprzedzona przedrostkiem w sposób przedstawiony w poniższych przykładach:

- Dla aplikacji 32-bitowych:

```
SET PATH=installation folder\bin\vs2015;%PATH%
```

- Dla aplikacji 64-bitowych:

```
SET PATH=installation folder\bin64\vs2015;%PATH%
```

**Pojęcia pokrewne**

Windows: zmiany w stosunku do IBM MQ 8.0

## z/OS Budowanie programów w języku C++ w systemie z/OS Batch, RRS Batch i CICS

Budowanie programów w języku IBM MQ C++ w systemie z/OS dla środowisk wsadowych, RRS lub CICS oraz uruchamianie programów przykładowych.

Można pisać programy w języku C++ dla trzech środowisk obsługiwanych przez IBM MQ for z/OS :

- Wsadowe
- Zadanie wsadowe RRS
- CICS

### Kompilowanie, prekonsolidowanie i konsolidowanie

Utwórz aplikację z/OS , kompilując, wstępnie dowiązując i edytując kod źródłowy w języku C + +.

Język IBM MQ C++ for z/OS jest zaimplementowany jako biblioteki DLL języka z/OS dla języka IBM C++ for z/OS . Za pomocą bibliotek DLL należy konkatenować dostarczone definicje po stronie z danymi wyjściowymi kompilatora w czasie łączenia wstępnego. Dzięki temu konsolidator będzie mógł sprawdzać wywołania funkcji składowych języka C + + systemu IBM MQ .

**Uwaga:** Istnieją trzy zestawy boczków dla każdego z trzech środowisk.

Aby zbudować aplikację IBM MQ for z/OS C++, należy utworzyć i uruchomić zadanie JCL. Użyj następującej procedury:

1. Jeśli aplikacja działa w systemie CICS, należy użyć procedury dostarczonej przez CICSw celu przetłumaczenia komend CICS w programie.

   Ponadto w przypadku aplikacji CICS należy wykonać następujące czynności:

   a. Dodaj bibliotekę SCSQLOAD do konkatenacji DFHRPL.

   b. Zdefiniuj grupę CSQCAT1 CEDA, używając elementu IMQ4B100 w bibliotece SCSQPROC.

   c. Zainstaluj CSQCAT1.
2. Skompiluj program, aby utworzyć kod obiektu. Kod JCL kompilacji musi zawierać instrukcje, które udostępniają kompilatorowi pliki definicji danych produktu. Definicje danych są dostarczane w następujących bibliotekach IBM MQ for z/OS :

   - **thlqual**, SCSQC370
   - **thlqual**.SCSQHPPS

   Upewnij się, że podano opcję kompilatora `/cxx` .

   **Uwaga:** Nazwa **thlqual** jest kwalifikatorem wysokiego poziomu biblioteki instalacyjnej IBM MQ w systemie z/OS.
3. Wstępnie dowiąż kod obiektu utworzony w kroku “2” na stronie 571, w tym następujące definicje po stronie, które są dostarczane w pliku **thlqual**.SCSQDEFS:

   a. imqs23dm i imqb23dm dla zadania wsadowego

   b. imqs23dr i imqb23dr dla zadania wsadowego RRS

   c. imqs23dc i imqb23dc w systemie CICS

   Są to odpowiednie biblioteki DLL.

   a. imqs23im i imqb23im dla zadania wsadowego

b. imqs23ir i imqb23ir dla zadania wsadowego RRS

c. imqs23ic i imqb23ic dla systemu CICS

4. Przeprowadź edycję kodu obiektu utworzonego w kroku “3” na stronie 571, aby utworzyć moduł ładujący i zapisać go w bibliotece dynamicznej aplikacji.

Aby uruchomić programy wsadowe lub RRS, należy dołączyć biblioteki **thlqual**.SCSQAUTH i **thlqual**.SCSQLOAD do konkatenacji zestawu danych STEPLIB lub JOBLIB.

Aby uruchomić program CICS , należy najpierw poprosić administratora systemu o zdefiniowanie go w systemie CICS jako programu IBM MQ i transakcji. Następnie można uruchomić go w zwykły sposób.

#### Uruchom programy przykładowe

Programy są opisane w sekcji “Przykładowe programy w języku C++” na stronie 548.

Przykładowe aplikacje są dostarczane tylko w formie źródłowej. Są to następujące pliki:

*Tabela 75. Przykładowe pliki programu z/OS*

| Próba | Program źródłowy (w bibliotece thlqual.SCSQCPPS) | JCL (w bibliotece thlqual.SCSQPROC) |
|---|---|---|
| Hello World | imqwrld | imqwrldr, |
| SPUT | imqsput | imqsputr |
| SGET (pobierz) | imqsget, | imqsgetr, |

Aby uruchomić przykłady, należy je skompilować i skonsolidować tak, jak w przypadku dowolnego programu w języku C + + (patrz sekcja “Budowanie programów w języku C++ w systemie z/OS Batch, RRS Batch i CICS” na stronie 571 ). Użyj dostarczonego kodu JCL do utworzenia i uruchomienia zadania wsadowego. Najpierw należy dostosować kod JCL, postępując zgodnie z dołączonym do niego komentarzem.

## z/OS Budowanie programów w języku C++ w systemie z/OS UNIX System Services

Budowanie programów w języku IBM MQ C++ w systemie z/OS UNIX System Services (z/OS UNIX).

Aby zbudować aplikację w powłoce z/OS UNIX , należy przyznać kompilatorowi dostęp do plików włączanych języka IBM MQ (znajdujących się w katalogach `thlqual.SCSQC370` i `hlqual.SCSQHPPS` ) oraz utworzyć dowiązanie do dwóch z tych plików DLL (znajdujących się w katalogu `thlqual.SCSQDEFS` ). W czasie wykonywania aplikacja musi mieć dostęp do IBM MQ zestawów danych `thlqual.SCSQLOAD`, `thlqual.SCSQAUTH`i jednego z zestawów danych specyficznych dla języka, na przykład `thlqual.SCSQANLE` .[6].

#### Kompilacja

1. Skopiuj przykład do systemu plików za pomocą komendy TSO **oput** lub użyj protokołu FTP. W dalszej części tego przykładu przyjęto, że przykład został skopiowany do katalogu o nazwie `/u/fred/samplei` nazwano go `imqwrld.cpp`.
2. Zaloguj się do powłoki z/OS UNIX i przejdź do katalogu, w którym umieszczono przykład.
3. Skonfiguruj kompilator C++ tak, aby mógł akceptować pliki DLL sidedeck i .cpp jako dane wejściowe:

```
/u/fred/sample:> export _CXX_EXTRA_ARGS=1
/u/fred/sample:> export _CXX_CXXSUFFIX="cpp"
```

[6] Można utworzyć powiązanie z dowolnym z wymienionych w sekcji " Wstępne dowiązanie kodu obiektu , aby uruchomić z/OS UNIX w dowolnym z trzech środowisk, “Budowanie programów w języku C++ w systemie z/OS Batch, RRS Batch i CICS” na stronie 571

4. Skompiluj i dowiąż przykładowy program. Poniższa komenda łączy program z wsadowymi Sidedecks; zamiast nich można użyć wsadowych Sidedecks usługi RRS. Znak \ jest używany do podzielenia komendy na więcej niż jeden wiersz. Nie wpisuj tego znaku; wprowadź komendę w jednym wierszu:

```
/u/fred/sample:> c++ -o imqwrld -I "//'thlqual.SCSQC370'" \
-I "//'thlqual.SCSQHPPS'" imqwrld.cpp \
"//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"
```

Więcej informacji na temat komendy TSO **oput** zawiera publikacja z/OS UNIX Command Reference.

Można również użyć programu narzędziowego make, aby uprościć budowanie programów w języku C ++. Poniżej znajduje się przykładowy plik makefile do budowania programu przykładowego HELLO WORLD C++. Oddziela etapy kompilacji i łączenia. Przed uruchomieniem komendy make skonfiguruj środowisko zgodnie z opisem w kroku “3” na stronie 572 .

```
flags = -I "//'thlqual.SCSQC370'" -I "//'thlqual.SCSQHPPS'"
decks = "//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"

imqwrld: imqwrld.o
   c++ -o imqwrld imqwrld.o $(decks)

imqwrld.o: imqwrld.cpp
   c++ -c -o imqwrld $(flags) imqwrld.cpp
```

Więcej informacji na temat korzystania z programu make można znaleźć w sekcji z/OS UNIX System Services Programming Tools .

**Działający**

1. Zaloguj się do powłoki z/OS UNIX i przejdź do katalogu, w którym został zbudowany przykład.
2. Skonfiguruj zmienną środowiskową STEPLIB, aby uwzględnić zestawy danych IBM MQ :

```
/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. Uruchom przykład:

```
/u/fred/sample:> ./imqwrld
```

# Tworzenie aplikacji .NET

IBM MQ classes for .NET zezwól aplikacjom .NET na nawiązywanie połączeń z produktem IBM MQ jako IBM MQ MQI client lub nawiązywanie połączeń bezpośrednio z serwerem IBM MQ .

W przypadku aplikacji, które korzystają z produktu Microsoft .NET Framework i chcą korzystać z narzędzi produktu IBM MQ, należy użyć produktu IBM MQ classes for .NET. Aby uzyskać więcej informacji, zapoznaj się z sekcją: “Instalowanie produktu IBM MQ classes for .NET Framework” na stronie 581.

Od wersji IBM MQ 9.1.1 IBM MQ obsługuje .NET Core dla aplikacji w środowiskach Windows . Aby uzyskać więcej informacji, zapoznaj się z sekcją: “Instalowanie produktu IBM MQ classes for .NET” na stronie 574.

Od wersji IBM MQ 9.1.2 IBM MQ obsługuje .NET Core dla aplikacji w środowiskach Linux .

W produkcie IBM MQ 9.1.4aplikacje zarządzane przez produkt IBM MQ .NET mogą automatycznie równoważyć połączenia między menedżerami kolejek w klastrze. Obsługiwane są zarówno biblioteki .NET Framework , jak i .NET Standard . Więcej informacji na ten temat zawiera sekcja Informacje o jednolitych klastrach i sekcja Automatyczne równoważenie aplikacji.

Obiektowy interfejs produktu IBM MQ .NET różni się od interfejsu MQI tym, że używa metod obiektów zamiast komend MQI.

Proceduralny aplikacyjny interfejs programistyczny IBM MQ jest oparty na czasownikach, takich jak te z poniższej listy:

```
MQCONN, MQDISC, MQOPEN, MQCLOSE,
MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

Wszystkie te komendy przyjmują jako parametr uchwyt do obiektu IBM MQ , na którym mają działać. Ponieważ .NET jest zorientowany obiektowo, interfejs programistyczny .NET obraca tę rundę. Program składa się z zestawu obiektów IBM MQ , na które użytkownik działa, wywołując metody na tych obiektach. Programy można pisać w dowolnym języku obsługiwanym przez .NET.

Jeśli używany jest interfejs proceduralny, należy rozłączyć się z menedżerem kolejek za pomocą wywołania MQDISC ( *Hconn*, CompCode, Reason), gdzie *Hconn* jest uchwytem menedżera kolejek.

W interfejsie produktu .NET menedżer kolejek jest reprezentowany przez obiekt klasy MQQueueManager. Rozłączenie z menedżerem kolejek następuje przez wywołanie metody Disconnect () dla tej klasy.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.Disconnect();
```

IBM MQ classes for .NET to zestaw klas, które umożliwiają aplikacjom .NET interakcję z produktem IBM MQ. Reprezentują one różne komponenty produktu IBM MQ , z których korzysta aplikacja, takie jak menedżery kolejek, kolejki, kanały i komunikaty. Szczegółowe informacje na temat tych klas zawiera sekcja Klasy i interfejsy produktu IBM MQ .NET.

Przed skompilowaniem napisanych aplikacji należy zainstalować środowisko .NET . Instrukcje dotyczące instalowania produktu IBM MQ classes for .NET i środowiska .NET można znaleźć w sekcji “Instalowanie produktu IBM MQ classes for .NET Framework” na stronie 581.

**Pojęcia pokrewne**

Przegląd techniczny

“Tworzenie aplikacji dla składnika IBM MQ” na stronie 5
Użytkownik może tworzyć aplikacje do wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i zasobami pokrewnymi. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i środowiskach.

**Zadania pokrewne**

Kontakt z działem wsparcia IBM

Rozwiązywanie problemów z produktem IBM MQ.NET

“Tworzenie aplikacji Microsoft Windows Communication Foundation przy użyciu języka IBM MQ” na stronie 1300
Kanał niestandardowy produktu Microsoft Windows Communication Foundation (WCF) dla produktu IBM MQ wysyła i odbiera komunikaty między klientami i usługami WCF.

“Tworzenie aplikacji XMS .NET” na stronie 635
IBM MQ Message Service Client (XMS) for .NET (XMS .NET) udostępnia aplikacyjny interfejs programistyczny (API) o nazwie XMS , który ma ten sam zestaw interfejsów co Java Message Service (JMS) Interfejs API. IBM MQ Message Service Client (XMS) for .NET zawiera w pełni zarządzaną implementację języka XMS, która może być używana przez dowolny język zgodny z językiem .NET .

## Windows Linux Instalowanie produktu IBM MQ classes for .NET

Produkt IBM MQ classes for .NET, w tym przykłady, jest instalowany razem z produktem IBM MQ w systemie Windows i Linux

## Wymagania wstępne i instalacja

W systemie IBM MQ 9.2.0biblioteka klienta IBM MQ .NET zbudowana przy użyciu pliku .NET Standard jest dostępna w systemach Windows i Linux. Aby uruchomić program IBM MQ classes for .NET Standard, należy zainstalować produkt Microsoft .NET Core. Microsoft.NET Core 3.1 jest minimalną wersją wymaganą do uruchomienia programu IBM MQ classes for .NET Standard.

V 9.3.0 V 9.3.0 Począwszy od wersji IBM MQ 9.3.0, IBM MQ obsługuje aplikacje .NET 6 korzystające z produktu IBM MQ classes for .NET Standard. Jeśli używana jest aplikacja .NET Core 3.1 , można uruchomić tę aplikację z niewielką modyfikacją w pliku `csproj` , ustawiając parametr `targetframeworkversion` na wartość `"net6.0"`, bez konieczności ponownego kompilowania.

V 9.3.1 IBM MQ 9.3.1 udostępnia bibliotekę klienta IBM MQ .NET zbudowaną w oparciu o środowisko .NET 6 jako środowisko docelowe. Począwszy od wersji IBM MQ 9.3.1, Microsoft .NET 6.0 jest minimalną wymaganą wersją do uruchamiania aplikacji korzystających z bibliotek IBM MQ , które zostały zbudowane przy użyciu środowiska .NET 6 jako środowiska docelowego.

V 9.3.1 W systemie IBM MQ 9.3.1biblioteka klienta IBM MQ .NET zbudowana przy użyciu pliku .NET Standard jest dostępna w nowym folderze `netstandard2.0` , a biblioteka klienta IBM MQ .NET zbudowana przy użyciu pliku .NET 6 jako struktura docelowa jest dostępna w produkcie *MQ_INSTALLATION_PATH*`/bin` w systemie Windows i w produkcie *MQ_INSTALLATION_PATH*`/lib64` w systemie Linux.

Najnowsza wersja środowiska IBM MQ classes for .NET jest domyślnie instalowana jako część standardowej instalacji produktu IBM MQ w składniku *Java and .NET Messaging and Web Services* .

Windows Więcej informacji na temat wymagań wstępnych i instalacji w systemie Windows:

- Informacje na temat oprogramowania wymaganego wstępnie do uruchomienia komendy IBM MQ classes for .NETzawiera sekcja Wymagania dla produktu IBM MQ classes for .NET.
- Instrukcje instalacji zawiera sekcja Instalowanie serwera IBM MQ w systemie Windows lub Instalowanie klienta IBM MQ w systemach Windows .

Linux Więcej informacji na temat wymagań wstępnych i instalacji w systemie Linux:

- Informacje na temat oprogramowania wymaganego wstępnie do uruchomienia komendy IBM MQ classes for .NETzawiera sekcja Wymagania dla produktu IBM MQ classes for .NET.
- Instrukcje instalacji pakietu rpm zawiera sekcja Instalowanie klienta IBM MQ w systemach Linux.
- W przypadku systemu Linux Ubuntukorzystającego z pakietów Debian należy zapoznać się z sekcją Instalowanie klienta IBM MQ w systemach Linux.

Biblioteka IBM MQ classes for .NET Standard , `amqmdnetstd.dll`, jest dostępna do pobrania z repozytorium NuGet . Więcej informacji na ten temat zawiera sekcja “Pobieranie pliku IBM MQ classes for .NET z repozytorium platformy NuGet” na stronie 580.

## `amqmdnetstd.dll` biblioteka

V 9.3.1 W systemie IBM MQ 9.3.1biblioteka `amqmdnetstd.dll` jest dostępna w następujących miejscach:

**Biblioteka zbudowana przy użyciu środowiska .NET Standard 2.0 jako struktury docelowej**

- Windows W systemie Windows: *MQ_INSTALLATION_PATH*`\bin\netstandard2.0`.
- Linux W systemie Linux: *MQ_INSTALLATION_PATH*`\lib64\netstandard2.0`.

Deprecated Te biblioteki są nieaktualne i produkt IBM zamierza je usunąć w przyszłych wersjach.

**Biblioteka zbudowana przy użyciu środowiska .NET 6 jako struktury docelowej**

- Windows W systemie Windows: *MQ_INSTALLATION_PATH*\bin. Aplikacje przykładowe są instalowane w produkcie *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base.
- Linux W systemie Linux: *MQ_INSTALLATION_PATH*\lib64. Przykłady .NET znajdują się w sekcji *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base.

LTS W przypadku systemu IBM MQ 9.3.0 Long Term Supportbiblioteka amqmdnetstd.dll jest dostępna w następujących miejscach:

- Windows W systemie Windows: *MQ_INSTALLATION_PATH*\bin. Aplikacje przykładowe są instalowane w produkcie *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base.
- Linux W systemie Linux: *MQ_INSTALLATION_PATH*/lib64 path. Przykłady .NET znajdują się w sekcji *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base.

**Ostrzeżenie:** Deprecated V 9.3.1 W produkcie IBM MQ 9.3.1biblioteki klienta IBM MQ .NET zbudowane przy użyciu środowiska .NET Standard 2.0 jako środowiska docelowego są nieaktualne, a aplikacje odwołujące się do tych bibliotek zgłaszają ostrzeżenie CS0618 podczas kompilacji.

LTS Stabilized Biblioteka amqmdnet.dll dla systemu .NET Framework jest nadal dostarczana, ale ta biblioteka jest ustabilizowana, co oznacza, że nie zostaną do niej wprowadzone żadne nowe funkcje. W przypadku wszystkich najnowszych składników należy przeprowadzić migrację do biblioteki amqmdnetstd.dll . Można jednak kontynuować korzystanie z biblioteki amqmdnet.dll w systemie IBM MQ 9.1 lub nowszym w wersji Long Term Support lub Continuous Delivery .

V 9.3.1 Jeśli aplikacja .NET Framework jest kompilowana przy użyciu pliku amqmdnetstd.dll lub amqmxmsstd.dll z wersji wcześniejszej niż IBM MQ 9.3.1 i ta sama aplikacja jest uruchamiana przy użyciu bibliotek klienta IBM MQ opartych na pliku .NET 6 , to wyjątek typu FileLoad jest zgłaszany przez .NET:

```
Wychwycono wyjątek: System.IO.FileLoadException: Nie można załadować pliku lub zespołu
'amqmdnetstd, Version =x.x.x.x, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e' lub
jedną z jej zależności. Odnaleziona definicja manifestu zespołu nie jest zgodna z
numer referencyjny zespołu. (Wyjątek z HRESULT: 0x80131040)

Nazwa pliku: ' amqmdnetstd, Version =x.x.x.x, Culture=neutral,
PublicKeyToken=23d6cb914eeaac0e'
```

Aby rozwiązać ten błąd, biblioteki znajdujące się w produkcie *MQ_INSTALLATION_PATH*/bin/netstandard2.0 muszą zostać skopiowane do katalogu, w którym działa aplikacja .NET Framework .

## Komenda dspmqver

Komenda **dspmqver** umożliwia wyświetlenie informacji o wersji i kompilacji dla komponentu .NET Core .

## Porównanie opcji IBM MQ classes for .NET Framework i IBM MQ classes for .NET (biblioteki.NET Standard i .NET 6 )

Poniższa tabela zawiera listę funkcji produktu IBM MQ classes for .NET Framework w porównaniu ze składnikami produktu IBM MQ classes for .NET (biblioteki.NET Standard i .NET 6 ).

*Tabela 76. Różnice między IBM MQ classes for .NET Framework i IBM MQ classes for .NET (biblioteki.NET Standard i .NET 6 )*

| **Funkcja** | **IBM MQ classes for .NET Framework** | **IBM MQ classes for .NET (biblioteki.NET Standard i .NET 6 )** |
|---|---|---|
| Nazwy klas (interfejsy API) | Wszystkie klasy pozostają takie same w każdej sieci. | Wszystkie klasy pozostają takie same w każdej sieci. |
| System operacyjny | Windows | Windows<br>Kontenery Docker<br>Linux<br>macOS |
| Plik `app.config` (plik konfiguracyjny do włączania śledzenia w kliencie redystrybuowanym) | Plik `app.config` służy do włączania śledzenia dla pakietu podlegającego redystrybucji i autonomicznego klienta IBM MQ .NET .<br>Więcej informacji na temat zmiennych używanych do śledzenia, w tym zmiennych **MQTRACEPATH** i **MQTRACELEVEL**, zawiera sekcja Śledzenie klienta IBM MQ classes for .NET Framework przy użyciu pliku konfiguracyjnego aplikacji . | `app.config` nie jest obsługiwana. Użyj zmiennych środowiskowych. |

*Tabela 76. Różnice między IBM MQ classes for .NET Framework i IBM MQ classes for .NET (biblioteki.NET Standard i .NET 6 ) (kontynuacja)*

| **Funkcja** | **IBM MQ classes for .NET Framework** | **IBM MQ classes for .NET (biblioteki.NET Standard i .NET 6 )** |
|---|---|---|
| Śledzenie | W przypadku pełnej instalacji klienta IBM MQmożna użyć komendy **strmqtrc** , aby włączyć śledzenie dla IBM MQ classes for .NET Framework.<br><br>W przypadku klientów podlegających redystrybucji plik `app.config` jest również używany do włączania śledzenia.<br><br>Więcej informacji na ten temat zawiera sekcja Śledzenie aplikacji produktu IBM MQ .NET.<br><br>V 9.3.3 W produkcie IBM MQ 9.3.3można włączać i wyłączać śledzenie, korzystając z pliku `mqclient.ini` i ustawiając odpowiednie właściwości w sekcji Trace. Można również dynamicznie włączać i wyłączać śledzenie za pomocą pliku `mqclient.ini` . Więcej informacji na ten temat zawiera sekcja Śledzenie aplikacji produktu IBM MQ .NET przy użyciu pliku mqclient.ini. | Zmienna środowiskowa **MQDOTNET_TRACE_ON** jest używana do włączania śledzenia dla klientów podlegających redystrybucji. Wartości mniejsze lub równe 0 nie włączają śledzenia. Wartość 1 włącza śledzenie na poziomie domyślnym. Wartość większa niż 1 powoduje włączenie szczegółowego śledzenia. Ustawienie tej zmiennej środowiskowej na inną wartość, taką jak łańcuch, nie powoduje włączenia śledzenia. Patrz Śledzenie aplikacji IBM MQ .NET przy użyciu zmiennych środowiskowych.<br><br>Zmienna środowiskowa **MQDOTNET_TRACE_ON** sprawdza, czy katalog śledzenia IBM MQ jest dostępny. Jeśli katalog śledzenia jest dostępny, plik śledzenia jest generowany w katalogu śledzenia. Jeśli jednak produkt IBM MQ nie jest zainstalowany, plik śledzenia jest kopiowany do bieżącego katalogu roboczego.<br><br>Inne zmienne środowiskowe, w tym **MQERRORPATH**, **MQLOGLEVEL**, **MQSERVER**itd. używane dla IBM MQ classes for .NET Framework, mogą być używane i działać w ten sam sposób.<br><br>V 9.3.3 W produkcie IBM MQ 9.3.3można włączać i wyłączać śledzenie, korzystając z pliku `mqclient.ini` i ustawiając odpowiednie właściwości w sekcji Trace. Można również dynamicznie włączać i wyłączać śledzenie za pomocą pliku `mqclient.ini` . Więcej informacji na ten temat zawiera sekcja Śledzenie aplikacji produktu IBM MQ .NET przy użyciu pliku mqclient.ini. |
| Tryby transportu | Zarządzany, Niezarządzany i Powiązania. | Zarządzany |

*Tabela 76. Różnice między IBM MQ classes for .NET Framework i IBM MQ classes for .NET (biblioteki.NET Standard i .NET 6 ) (kontynuacja)*

| Funkcja | IBM MQ classes for .NET Framework | IBM MQ classes for .NET (biblioteki.NET Standard i .NET 6 ) |
|---|---|---|
| TLS | Magazyn kluczy systemu Windows służy do przechowywania certyfikatów. | Windows W systemie Windows certyfikaty należy przechowywać w magazynie kluczy. Dozwolone wartości: *USER lub *SYSTEM. Zależnie od danych wejściowych klient IBM MQ .NET przeszukuje magazyn kluczy systemu Windows bieżącego użytkownika lub całego systemu.<br>Linux W Linux zaleca się użycie klasy X509Store w celu zainstalowania certyfikatów. Program .NET Core instaluje certyfikaty w następującym położeniu: "`.dotnet/corefx/cryptography/x509stores`". |
| CCDT | Obsługiwany | Obsługiwana. Ustawienia ścieżki do tabeli CCDT są takie same jak w przypadku klas produktu .NET Framework. |
| Automatyczne ponowne łączenie klienta | Obsługiwany | Obsługiwany |
| Rozproszone transakcje | Obsługiwany | Nieobsługiwane |
| Instalowanie bibliotek dołączanych dynamicznie (bibliotek DLL) w pamięci podręcznej Global Assembly Cache (GAC) | Biblioteki DLL są instalowane w pamięci GAC jako część instalacji produktu IBM MQ. | Biblioteki DLL nie są instalowane w pamięci GAC jako część instalacji produktu IBM MQ. |

**Uwaga:** Windows Identyfikatory zabezpieczeń (SID) systemu Windows :

Uwierzytelnianie na poziomie domeny nie jest obsługiwane dla IBM MQ classes for .NET (biblioteki.NET Standard i .NET 6 ). Identyfikator zalogowanego użytkownika jest używany do uwierzytelniania.

## Tworzenie aplikacji IBM MQ .NET Core w systemie macOS

macOS

Aplikacje IBM MQ .NET Core można tworzyć w systemie macOS.

Biblioteki produktu IBM MQ .NET nie są dostarczane z pakietem macOS , dlatego należy je skopiować z klienta Windows lub Linux IBM MQ do katalogu macOS. Następnie można użyć tych bibliotek do tworzenia aplikacji produktu IBM MQ .NET Core w systemie macOS.

Po opracowaniu te aplikacje mogą być uruchamiane w środowiskach Windows lub Linux .

**Pojęcia pokrewne**

“Instalowanie produktu IBM MQ classes for .NET Framework” na stronie 581

Produkt IBM MQ classes for .NET Framework, w tym przykłady, jest instalowany razem z produktem IBM MQ. Istnieje wymaganie wstępne dla produktu Microsoft.NET Framework w systemie Windows.

“Instalowanie produktu IBM MQ classes for XMS .NET” na stronie 640
Produkt IBM MQ classes for XMS .NET, w tym przykłady, jest instalowany razem z produktem IBM MQ w systemie Windows i Linux.

## Windows Linux Pobieranie pliku IBM MQ classes for .NET z repozytorium platformy NuGet

IBM MQ classes for .NET są dostępne do pobrania z repozytorium NuGet , dzięki czemu mogą być łatwo używane przez programistów .NET .

### O tym zadaniu

NuGet jest menedżerem pakietów dla platform programistycznych Microsoft , w tym .NET. Narzędzia klienckie NuGet umożliwiają tworzenie i konsumowanie pakietów. Pakiet NuGet to pojedynczy skompresowany plik z rozszerzeniem `.nupkg` , który zawiera skompilowany kod (DLL), inne pliki powiązane z tym kodem oraz opisowy manifest zawierający informacje takie jak numer wersji pakietu.

Pakiet produktu `IBMMQDotnetClient` NuGet , który zawiera bibliotekę `amqmdnetstd.dll` , można pobrać z galerii NuGet , która jest centralnym repozytorium pakietów używanym przez wszystkich autorów i konsumentów pakietów.

**Uwaga:** V 9.3.1 Począwszy od wersji IBM MQ 9.3.1pakiet NuGet zawiera biblioteki zbudowane przy użyciu środowiska docelowego .NET Standard 2.0 i .NET 6 . Biblioteki dla systemu .NET Standard 2.0 są dostępne w folderze `netstandard2.0` , a biblioteki dla systemu .NET 6 w folderze `net6.0` .

Istnieją trzy sposoby pobierania pakietu `IBMMQDotnetClient` :

- Za pomocą komendy Microsoft Visual Studio. Produkt NuGet jest dystrybuowany jako rozszerzenie Microsoft Visual Studio . Począwszy od wersji Microsoft Visual Studio 2012, produkt NuGet jest domyślnie wstępnie instalowany.
- Z poziomu wiersza komend za pomocą menedżera pakietów NuGet lub interfejsu CLI .NET .
- Za pomocą przeglądarki WWW.

Podobnie jak w przypadku pakietu podlegającego redystrybucji, śledzenie można włączyć za pomocą zmiennej środowiskowej **MQDOTNET_TRACE_ON**.

### Procedura

- Aby pobrać pakiet `IBMMQDotnetClient` za pomocą interfejsu użytkownika menedżera pakietów w programie Microsoft Visual Studio, wykonaj następujące kroki:
  a) Kliknij prawym przyciskiem myszy projekt .NET , a następnie kliknij opcję **Zarządzaj pakietami Nuget**.
  b) Kliknij kartę **Przeglądaj** i wyszukaj łańcuch `"IBMMQDotnetClient"`.
  c) Wybierz pakiet i kliknij przycisk **Instaluj**.
  Podczas instalacji Menedżer pakietów udostępnia informacje o postępie w postaci instrukcji konsoli.
- Aby pobrać pakiet `IBMMQDotnetClient` z wiersza komend, wybierz jedną z następujących opcji:
  - W programie NuGet Package Manager wprowadź następującą komendę:

    ```
    Install-Package IBMMQDotnetClient -Version 9.1.4.0
    ```

    Podczas instalacji Menedżer pakietów udostępnia informacje o postępie w postaci instrukcji konsoli. Dane wyjściowe można przekierować do pliku dziennika.

- W interfejsie wiersza komend .NET wprowadź następującą komendę:

```
dotnet add package IBMMQDotnetClient --version 9.1.4
```

- Za pomocą przeglądarki WWW pobierz pakiet `IBMMQDotnetClient` z serwisu https://www.nuget.org/packages/IBMMQDotnetClient.

**Pojęcia pokrewne**

Informacje licencyjne dotyczące programu IBM MQ Client for .NET

**Zadania pokrewne**

“Pobieranie produktu IBM MQ classes for XMS .NET z repozytorium platformy NuGet” na stronie 643

Produkt IBM MQ classes for XMS .NET jest dostępny do pobrania z repozytorium NuGet , dzięki czemu może być łatwo używany przez programistów .NET .

## Windows Instalowanie produktu IBM MQ classes for .NET Framework

Produkt IBM MQ classes for .NET Framework, w tym przykłady, jest instalowany razem z produktem IBM MQ. Istnieje wymaganie wstępne dla produktu Microsoft.NET Framework w systemie Windows.

Najnowsza wersja produktu IBM MQ classes for .NET Framework jest domyślnie instalowana jako część standardowej instalacji produktu IBM MQ w składniku *Przesyłanie komunikatów Java i .NET oraz usługi Web Services* . Instrukcje instalacji zawiera sekcja Instalowanie serwera IBM MQ w systemie Windows lub Instalowanie klienta IBM MQ w systemach Windows.

V 9.3.0 V 9.3.0 W systemie IBM MQ 9.3.0, aby uruchomić program IBM MQ classes for .NET Framework , należy zainstalować program Microsoft.NET Framework V4.7.2 lub nowszej. Jest to zmiana w stosunku do wersji IBM MQ 9.2 , w której minimalną wymaganą wersją było V4.6.2.

V 9.3.0 V 9.3.0 Istniejące aplikacje skompilowane przy użyciu programu Microsoft.NET Framework V3.5 można uruchamiać bez rekompilacji, dodając następujący znacznik w pliku `app.config` aplikacji:

```
<configuration>
 <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2"/>
 </startup>
</configuration>
```

**Uwaga:** Jeśli produkt Microsoft .NET Framework V4.7.2 lub nowszej nie zostanie zainstalowany przed zainstalowaniem produktu IBM MQ, instalacja produktu IBM MQ będzie kontynuowana bez błędu, ale plik IBM MQ classes for .NET nie będzie dostępny. Jeśli produkt.NET Framework został zainstalowany po zainstalowaniu produktu IBM MQ, należy zarejestrować zespoły produktu IBM MQ.NET , uruchamiając skrypt *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` , gdzie *WMQInstallDir* jest katalogiem, w którym zainstalowano produkt IBM MQ . Ten skrypt instaluje wymagane zespoły w globalnej pamięci podręcznej zespołu (GAC). Zestaw plików `amqi*.log` , które rejestrują wykonywane działania, jest tworzony w katalogu %TEMP% . Nie jest konieczne ponowne uruchamianie skryptu `amqiRegisterdotNet.cmd` , jeśli produkt .NET został zaktualizowany do wersji V4.7.2 lub nowszej z wcześniejszej wersji, na przykład z produktu .NET V3.5.

W środowisku z wieloma instalacjach, jeśli wcześniej zainstalowano produkt IBM MQ classes for .NET jako pakiet obsługi, nie można zainstalować produktu IBM MQ , chyba że najpierw zostanie zdeinstalowany pakiet obsługi. Składnik IBM MQ classes for .NET , który jest instalowany z produktem IBM MQ , zawiera te same funkcje, co pakiet obsługi.

Dostarczane są również przykładowe aplikacje, w tym pliki źródłowe; patrz sekcja “Przykładowe aplikacje dla produktu .NET” na stronie 582.

Informacje na temat używania niestandardowego kanału IBM MQ dla systemu Microsoft WCF z systemem .NETzawiera sekcja “Tworzenie aplikacji Microsoft Windows Communication Foundation przy użyciu języka IBM MQ” na stronie 1300

**Pojęcia pokrewne**

“Instalowanie produktu IBM MQ classes for .NET” na stronie 574
Produkt IBM MQ classes for .NET, w tym przykłady, jest instalowany razem z produktem IBM MQ w systemie Windows i Linux

**Zadania pokrewne**

Śledzenie aplikacji IBM MQ .NET

# Opcje łączenia programu IBM MQ classes for .NET z menedżerem kolejek

Istnieją trzy tryby nawiązywania połączenia między programem IBM MQ classes for .NET i menedżerem kolejek. Należy rozważyć, który typ połączenia najlepiej odpowiada wymaganiom.

## Połączenie powiązań klienta

Aby użyć produktu IBM MQ classes for .NET jako IBM MQ MQI client, można go zainstalować razem z produktem IBM MQ MQI clientna serwerze IBM MQ lub na osobnym komputerze. Połączenie powiązań klienta może korzystać z transakcji XA lub transakcji innych niż XA

## Połączenie powiązań serwera

W przypadku użycia w trybie powiązań serwera produkt IBM MQ classes for .NET używa interfejsu API menedżera kolejek zamiast komunikacji w sieci. Zapewnia to lepszą wydajność aplikacji IBM MQ niż używanie połączeń sieciowych.

Aby użyć połączenia powiązań, należy zainstalować produkt IBM MQ classes for .NET na serwerze IBM MQ .

## Zarządzane połączenie klienta

Połączenie nawiązane w tym trybie łączy się jako klient IBM MQ z serwerem IBM MQ działającym na komputerze lokalnym lub zdalnym.

Połączenie IBM MQ classes for .NET w tym trybie pozostaje w kodzie zarządzanym programu .NET i nie wywołuje usług rodzimych. Więcej informacji na temat kodu zarządzanego zawiera dokumentacja systemu Microsoft .

Korzystanie z klienta zarządzanego podlega pewnym ograniczeniom. Więcej informacji na ten temat zawiera sekcja “Zarządzane połączenia klienta” na stronie 598.

# Przykładowe aplikacje dla produktu .NET

Aby uruchomić własne aplikacje .NET , należy skorzystać z instrukcji dla programów weryfikujących, zastępując nazwę aplikacji nazwą przykładową.

Dostarczane są następujące aplikacje przykładowe:

- Aplikacja umieszczania komunikatów
- Aplikacja do pobierania komunikatów
- Aplikacja "hello world"
- Aplikacja publikowania/subskrypcji
- Aplikacja używająca właściwości komunikatu

Wszystkie te aplikacje przykładowe są dostarczane w języku C#, a niektóre w językach C++ i Visual Basic. Aplikacje można pisać w dowolnym języku obsługiwanym przez produkt .NET.

**"Put message" program SPUT (nmqsput.cs, mmqsput.cpp, vmqsput.vb)**

Ten program pokazuje, jak umieścić komunikat w nazwanej kolejce. Program ma trzy parametry:

- Nazwa kolejki (wymagana), na przykład SYSTEM.DEFAULT.LOCAL.QUEUE
- Nazwa menedżera kolejek (opcjonalnie)

- Definicja kanału (opcjonalna), na przykład SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Jeśli nie zostanie podana żadna nazwa menedżera kolejek, wartością domyślną menedżera kolejek będzie domyślny menedżer kolejek lokalnych. Jeśli kanał jest zdefiniowany, ma taki sam format jak zmienna środowiskowa MQSERVER.

**"Get message" program SGET (nmqsget.cs, mmqsget.cpp, vmqsget.vb)**
Ten program pokazuje, w jaki sposób pobrać komunikat z kolejki nazwanej. Program ma trzy parametry:

- Nazwa kolejki (wymagana), na przykład SYSTEM.DEFAULT.LOCAL.QUEUE
- Nazwa menedżera kolejek (opcjonalnie)
- Definicja kanału (opcjonalna), na przykład SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Jeśli nie zostanie podana żadna nazwa menedżera kolejek, wartością domyślną menedżera kolejek będzie domyślny menedżer kolejek lokalnych. Jeśli kanał jest zdefiniowany, ma taki sam format jak zmienna środowiskowa MQSERVER.

**Program "Hello World" (nmqwrld.cs, mmqwrld.cpp, vmqwrld.vb)**
Ten program pokazuje, jak umieścić i otrzymać komunikat. Program ma trzy parametry:

- Nazwa kolejki (opcjonalna), na przykład SYSTEM.DEFAULT.LOCAL.QUEUE lub SYSTEM.DEFAULT.MODEL.QUEUE
- Nazwa menedżera kolejek (opcjonalnie)
- Definicja kanału (opcjonalna), na przykład SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Jeśli nazwa kolejki nie zostanie podana, zostanie użyta nazwa domyślna SYSTEM.DEFAULT.LOCAL.QUEUE. Jeśli nie zostanie podana żadna nazwa menedżera kolejek, wartością domyślną menedżera kolejek będzie domyślny menedżer kolejek lokalnych.

**Program "Publish/subscribe" (MQPubSubSample.cs)**
Ten program pokazuje, w jaki sposób można używać funkcji publikowania/subskrybowania programu IBM MQ . Jest on dostarczany tylko w języku C#. Program ma dwa parametry:

- Nazwa menedżera kolejek (opcjonalnie)
- Definicja kanału (opcjonalnie)

**Program "Właściwości komunikatu" (MQMessagePropertiesSample.cs)**
W tym programie przedstawiono sposób użycia właściwości komunikatu. Jest on dostarczany tylko w języku C#. Program ma dwa parametry:

- Nazwa menedżera kolejek (opcjonalnie)
- Definicja kanału (opcjonalnie)

Instalację można zweryfikować, kompilując i uruchamiając te aplikacje.

## Miejsca instalacji

Przykładowe aplikacje są instalowane w następujących miejscach zgodnie z językiem, w którym zostały napisane. *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

**C#**

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\nmqswrld.cs

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\nmqsput.cs

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\nmqsget.cs

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\MQPubSubSample.cs

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\MQMessagePropertiesSample.cs

**Zarządzane C++**

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\mcp\mmqswrld.cpp

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\mcp\mmqsput.cpp

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\mcp\mmqsget.cpp

**Visual Basic**

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\vb\vmqswrld.vb

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\vb\vmqsput.vb

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\vb\vmqsget.vb

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\vb\xmqswrld.vb

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\vb\xmqsput.vb

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\vb\xmqsget.vb

## Budowanie przykładowych aplikacji

W celu zbudowania przykładowych aplikacji dostarczany jest plik wsadowy dla każdego języka.

**C#**

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\bldcssamp.bat

Plik bldcssamp.bat zawiera wiersz dla każdego przykładu, który jest wszystkim, co jest niezbędne do zbudowania tego programu przykładowego:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin
/out:nmqwrld.exe nmqwrld.cs
```

**Zarządzane C++**

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\mcp\bldmcpsamp.bat

Plik bldmcpsamp.bat zawiera wiersz dla każdego przykładu, który jest wszystkim, co jest niezbędne do zbudowania tego programu przykładowego:

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Jeśli chcesz skompilować te aplikacje w programie Microsoft Visual Studio 2003/.NET SDKv1.1, zastąp komendę kompilacji:

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

z

```
cl /clr MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

**Visual Basic**

*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\vb\bldvbsamp.bat

Plik bldvbsamp.bat zawiera wiersz dla każdego przykładu, który jest wszystkim, co jest niezbędne do zbudowania tego programu przykładowego:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:vmqwrld.exe vmqwrld.vb
```

### Przykłady użycia komendy IBM MQ z produktem Microsoft .NET Core

Począwszy od wersji IBM MQ 9.2.0, IBM MQ obsługuje aplikacje .NET Core for IBM MQ .NET w środowiskach Windows . Produkt IBM MQ classes for .NET Standard, w tym przykłady, jest instalowany domyślnie w ramach standardowej instalacji produktu IBM MQ .

Aplikacje przykładowe dla produktu IBM MQ .NET są instalowane w katalogu `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/base`. Dostępny jest również skrypt, którego można użyć do skompilowania przykładów.

Przykłady można zbudować przy użyciu dostarczonych plików `build.bat` . Dla każdego przykładu istnieje jeden plik `build.bat` w następującym położeniu w systemie Windows:

- `MQ\tools\dotnet\samples\cs\core\base\SimpleGet`
- `MQ\tools\dotnet\samples\cs\core\base\SimplePut`

Linux Od wersji IBM MQ 9.2.0produkt IBM MQ obsługuje również moduł podstawowy dla aplikacji w środowiskach Linux .

Więcej informacji na temat używania języka IBM MQ z produktem Microsoft .NET Corezawiera sekcja “Instalowanie produktu IBM MQ classes for .NET” na stronie 574.

## Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienta TCP/IP

Skonfiguruj menedżer kolejek w celu akceptowania żądań połączeń przychodzących od klientów.

### O tym zadaniu

W tym zadaniu opisano podstawowe kroki konfigurowania menedżera kolejek w celu akceptowania połączeń klienckich TCP/IP. W przypadku systemu produkcyjnego należy również wziąć pod uwagę wpływ na zabezpieczenia podczas konfigurowania menedżerów kolejek.

### Procedura

1. Zdefiniuj kanał połączenia z serwerem:
   a. Uruchom menedżer kolejek.
   b. Zdefiniuj przykładowy kanał o nazwie NET.CHANNEL:

```
DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
DESCR('Sample channel for IBM MQ classes for .NET')
```

   **Ważne:** Ten przykład jest przeznaczony tylko do użytku w środowisku testowym, ponieważ nie uwzględnia on wpływu na bezpieczeństwo. W przypadku systemu produkcyjnego należy rozważyć użycie protokołu TLS lub wyjścia zabezpieczeń. Więcej informacji na ten temat zawiera sekcja Zabezpieczanie produktu IBM MQ.
2. Uruchom program nasłuchujący:

```
runmqlsr -t tcp [-m qmnqme ] [-p portnum ]
```

   **Uwaga:** Nawiasy kwadratowe wskazują parametry opcjonalne. Wartość *qmname* nie jest wymagana dla domyślnego menedżera kolejek, a numer portu *portnum* nie jest wymagany, jeśli używana jest wartość domyślna (1414).

## Rozproszone transakcje w programie .NET

Rozproszone transakcje lub transakcje globalne umożliwiają aplikacjom klienckim uwzględnienie w jednej transakcji kilku różnych źródeł danych w dwóch lub większej liczbie systemów sieciowych.

W transakcjach rozproszonych menedżer transakcji koordynuje i zarządza transakcją między co najmniej dwoma menedżerami zasobów.

Transakcje mogą być jednofazowym lub dwufazowym procesem zatwierdzania. Zatwierdzanie jednofazowe jest procesem, w którym tylko jeden menedżer zasobów uczestniczy w transakcji, a proces zatwierdzania dwufazowego jest procesem, w którym w transakcji uczestniczy więcej niż jeden menedżer zasobów. W procesie zatwierdzania dwufazowego menedżer transakcji wysyła wywołanie przygotowania w celu sprawdzenia, czy wszystkie menedżery zasobów są przygotowane do zatwierdzenia. Po odebraniu potwierdzenia od wszystkich menedżerów zasobów jest wysyłane wywołanie zatwierdzenia. W przeciwnym razie nastąpi wycofanie całej transakcji. Więcej informacji na ten temat zawiera sekcja Zarządzanie transakcjami i wsparcie . Menedżerowie zasobów powinni informować menedżerów transakcji o swoim udziale w transakcji. Gdy menedżer zasobów informuje menedżera transakcji o swoim uczestnictwie, menedżer zasobów otrzymuje wywołania zwrotne od menedżera transakcji, gdy transakcja ma zostać zatwierdzona lub wycofana.

Klasy IBM MQ .NET obsługują już rozproszone transakcje w połączeniach w trybie powiązań niezarządzanych i w trybie powiązań serwera. W tych trybach klasy IBM MQ .NET delegują wszystkie swoje wywołania do rozszerzonego klienta transakcji C, który zarządza przetwarzaniem transakcji w imieniu .NET.

Klasy IBM MQ.NET obsługują obecnie rozproszone transakcje w trybie zarządzanym, w którym klasy IBM MQ .NET używają przestrzeni nazw System.Transactions do obsługi rozproszonych transakcji. Infrastruktura System.Transactions ułatwia i usprawnia programowanie transakcyjne, obsługując transakcje zainicjowane we wszystkich menedżerach zasobów, w tym w produkcie IBM MQ. Aplikacja IBM MQ .NET może umieszczać i wysyłać komunikaty przy użyciu .NET niejawnego modelu programowania transakcji lub jawnego modelu programowania transakcji. W transakcjach niejawnych granice transakcji są tworzone przez aplikację, która decyduje, kiedy zatwierdzić, wycofać (dla transakcji jawnych) lub zakończyć transakcję. W przypadku transakcji jawnych należy jawnie określić, czy transakcja ma zostać zatwierdzona, wycofana i zakończona.

Produkt IBM MQ.NET używa koordynatora rozproszonych transakcji Microsoft (MS DTC) jako menedżera transakcji, który koordynuje i zarządza transakcją między wieloma menedżerami zasobów. IBM MQ jest używany jako menedżer zasobów. Należy zauważyć, że nie można używać protokołu TLS z transakcjami XA. Należy użyć tabeli CCDT. Więcej informacji na ten temat zawiera sekcja Używanie rozszerzonego klienta transakcyjnego z kanałami TLS.

Produkt IBM MQ.NET jest zgodny z modelem X/Open Distributed Transaction Processing (DTP). Model X/Open Distributed Transaction Processing jest modelem rozproszonego przetwarzania transakcji zaproponowanym przez Open Group, konsorcjum dostawców. Ten model jest standardem wśród większości komercyjnych dostawców w zakresie przetwarzania transakcji i domen baz danych. Większość komercyjnych produktów do zarządzania transakcjami obsługuje model X/DTP.

## Tryby transakcji

- “Rozproszone transakcje w trybie zarządzanym .NET” na stronie 587
- Transakcje rozproszone w trybie niezarządzanym

## Koordynowanie transakcji w różnych scenariuszach

- Połączenie może uczestniczyć w kilku transakcjach, ale w danym momencie aktywna jest tylko jedna transakcja.
- Podczas transakcji MQQueueManager.Połączenie odłączone jest honorowane. W takim przypadku transakcja jest wycofywana.
- Podczas transakcji honorowane jest wywołanie MQQueue.Close lub MQTopic.Close . W tym przypadku transakcja jest proszona o wycofanie zmian.
- Granice transakcji są tworzone przez aplikację, która decyduje o tym, kiedy zatwierdzić, wycofać (dla transakcji jawnych) lub zakończyć (dla transakcji niejawnych) transakcję.

- Jeśli aplikacja kliencka zostanie przerwana podczas transakcji z nieoczekiwanym błędem przed wywołaniem metody Put lub Get w wywołaniu kolejki lub tematu, transakcja zostanie wycofana i zgłoszony zostanie wyjątek MQException.
- Jeśli kod przyczyny MQCC_FAILED zostanie zwrócony podczas wywołania Put lub Get w kolejce lub wywołaniu Topic, zgłaszany jest wyjątek MQException z kodem przyczyny i transakcja jest wycofywana. Jeśli wywołanie metody prepare zostało już wydane przez menedżer transakcji, program IBM MQ .NET zwraca żądanie metody prepare, wymuszając wycofanie transakcji. Następnie DTC menedżera transakcji powoduje wycofanie bieżącej pracy ze wszystkimi menedżerami zasobów w bieżących transakcjach otoczenia.
- Podczas transakcji obejmującej wiele menedżerów zasobów, jeśli z jakiegoś powodu środowiskowego wywołanie Put lub Get zawiesi się w nieskończoność, menedżer transakcji czeka do określonego czasu. Po upływie tego czasu powoduje on wycofanie całej bieżącej pracy ze wszystkimi menedżerami zasobów w bieżących transakcjach otoczenia. Jeśli w fazie przygotowania wystąpi nieokreślony czas oczekiwania, menedżer transakcji może przekroczyć limit czasu lub wywołać wątpliwe wywołanie zasobu, w którym to przypadku transakcja zostanie wycofana.
- Aplikacje używające transakcji muszą umieścić lub pobrać komunikaty w SYNC_POINT. Jeśli wywołanie metody Put lub Get zostanie wysłane w kontekście transakcyjnym, który nie jest w sekcji SYNC_POINT, wywołanie nie powiedzie się i zostanie zwrócony kod przyczyny MQRC_UNIT_OF_WORK_NOT_STARTED.

### Różnice w zachowaniu między obsługą transakcji klienta zarządzanego i niezarządzanego przy użyciu przestrzeni nazw Microsoft.NET System.Transactions

Zagnieżdżone transakcje mają TransactionScope wewnątrz innego TransactionScope

- W pełni zarządzany klient IBM MQ .NET obsługuje zagnieżdżony TransactionScope
- Niezarządzany klient IBM MQ .NET nie obsługuje zagnieżdżonego TransactionScope

Transakcje zależne od strumienia System.Transactions

- W pełni zarządzany klient IBM MQ .NET obsługuje zależne narzędzie transakcji udostępniane przez serwer System.Transactions.
- Niezarządzany klient IBM MQ .NET nie obsługuje narzędzia transakcji zależnych udostępnianego przez serwer System.Transactions.

### Przykłady produktów

Przykłady produktów `SimpleXAPut`i `SimpleXAGet` są dostępne w katalogu `WebSphere MQ\tools\dotnet\samples\cs\base`. Przykłady są aplikacjami C#, które demonstrują użycie komend MQPUT i MQGET w obszarze transakcji rozproszonych przy użyciu przestrzeni nazw SystemTransactions . Więcej informacji na temat tych przykładów zawiera sekcja “Tworzenie prostych operacji put i get dla komunikatów w zasięgu TransactionScope” na stronie 590.

## Rozproszone transakcje w trybie zarządzanym .NET

Klasy serwera IBM MQ .NET używają przestrzeni nazw System.Transactions do obsługi rozproszonych transakcji w trybie zarządzanym. W trybie zarządzanym MS DTC koordynuje i zarządza rozproszonymi transakcjami na wszystkich serwerach wpisanych do listy transakcji.

Klasy serwera IBM MQ .NET udostępniają jawny model programistyczny oparty na klasie System.Transactions.Transaction oraz niejawny model programistyczny korzystający z klasy System.Transactions.TransactionScope, w której transakcje są automatycznie zarządzane przez infrastrukturę.

**Transakcja niejawna**

Poniższy fragment kodu opisuje sposób, w jaki aplikacja IBM MQ .NET umieszcza komunikat przy użyciu niejawnego programowania transakcyjnego .NET .

```
Using (TransactionScope scope = new TransactionScope ())
```

```
{
    Q.Put (putMsg,pmo);
    scope.Complete ();
}

Q.close();
qMgr.Disconect();}
```

**Wyjaśnienie przepływu kodu transakcji niejawnej**

Kod tworzy *TransactionScope* i umieszcza komunikat w zasięgu. Następnie wywołuje funkcję *Zakończone* , aby poinformować koordynatora transakcji o zakończeniu transakcji. Koordynator transakcji wydaje teraz komendy *prepare* i *commit* w celu zakończenia transakcji. W przypadku wykrycia problemu wywoływane jest *wycofanie zmian* .

**Transakcja jawna**

Poniższy kod opisuje, w jaki sposób aplikacja IBM MQ .NET umieszcza komunikaty przy użyciu jawnego modelu programowania transakcji .NET .

```
MQQueueManager qMgr = new MQQueuemanager ("MQQM);
MQQueue Q = QMGR.AccessQueue("Q", MQC.MQOO_OUTPUT+MQC.MQOO_INPUT_SHARED);
MQPutMessageOptions pmo = new MQPutMessageOptions();
pmo.Options = MQC.MQPMO_SYNCPOINT;
MQMessage putMsg1 = new MQMessage();
Using(CommittableTransaction tx = new CommittableTransaction()){
Transaction.Current = tx;
    try
    {
    Q.Put(MSG,pmo);
    tx.commit();
    }
    catch(Exception)
    {tx.rollback();}
    }

Q.close();
qMgr.Disconnect();
}
```

**Wyjaśnienie przepływu kodu jawnej transakcji**

Fragment kodu tworzy transakcję przy użyciu klasy *CommitableTransaction* . Umieszcza komunikat w tym zasięgu, a następnie jawnie wywołuje funkcję *commit* w celu zakończenia transakcji. Jeśli wystąpią jakiekolwiek problemy, wywoływane jest *wycofanie zmian* .

## Rozproszone transakcje w trybie niezarządzanym ( .NET )

Klasy IBM MQ.NET obsługują połączenia niezarządzane (klient) przy użyciu rozszerzonego klienta transakcji i modelu COM + /MTS jako koordynatora transakcji przy użyciu niejawnego lub jawnego modelu programowania transakcji. W trybie niezarządzanym klasy IBM MQ .NET delegują wszystkie wywołania do rozszerzonego klienta transakcji C, który zarządza przetwarzaniem transakcji w imieniu .NET.

Przetwarzanie transakcji jest sterowane przez zewnętrznego menedżera transakcji, który koordynuje globalną jednostkę pracy pod kontrolą interfejsu API menedżera transakcji. Komendy MQBEGIN, MQCMIT i MQBACK są niedostępne. IBM MQ Klasy .NET prezentują tę obsługę w trybie transportu niezarządzanego (klient C). Więcej informacji na ten temat zawiera sekcja Konfigurowanie menedżerów transakcji zgodnych z interfejsem XA .

MTS jest systemem przetwarzania transakcyjnego (TP), który udostępnia takie same funkcje w systemie Windows NT , jakie są dostępne w systemach CICS, Tuxedo i na innych platformach. Po zainstalowaniu serwera MTS do produktu Windows NT dodawana jest oddzielna usługa o nazwie Microsoft Distributed Transaction Coordinator (MSDTC). MSDTC koordynuje transakcje obejmujące osobne składnice danych lub zasoby. Do działania wymagane jest, aby każda składnica danych implementowała własnego menedżera zasobów.

Produkt IBM MQ jest kompatybilny z produktem MSDTC przez zaimplementowanie interfejsu (własnego interfejsu menedżera zasobów), w którym zarządza odwzorowaniem wywołań DTC XA na wywołania IBM MQ(X/Open). IBM MQ pełni rolę menedżera zasobów.

Gdy komponent, taki jak COM +, żąda dostępu do IBM MQ, zwykle sprawdza przy użyciu odpowiedniego obiektu kontekstu MTS, czy transakcja jest wymagana. Jeśli transakcja jest wymagana, komunikat COM informuje DTC i automatycznie uruchamia integralną transakcję IBM MQ dla tej operacji. Następnie COM współpracuje z danymi za pośrednictwem oprogramowania MQMTS, umieszczając i pobierając komunikaty zgodnie z wymaganiami. Instancja obiektu uzyskana z modelu COM wywołuje metodę SetComplete lub SetAbort po zakończeniu wszystkich działań na danych. Po wywołaniu przez aplikację komendy SetCompletewywołanie sygnalizuje DTC, że aplikacja zakończyła transakcję, a DTC może kontynuować proces zatwierdzania dwufazowego. Następnie usługa DTC wywołuje usługę MQMTS, która z kolei wywołuje funkcję IBM MQ w celu zatwierdzenia lub wycofania transakcji.

## Pisanie aplikacji IBM MQ .NET przy użyciu klienta niezarządzanego

Aby uruchomić w kontekście COM +, klasa .NET musi dziedziczyć z systemu.EnterpriseServices.ServicedComponent. Reguły i zalecenia dotyczące tworzenia zespołów korzystających z komponentów obsługiwanych są następujące:

**Uwaga:** Poniższe kroki mają zastosowanie tylko wtedy, gdy używany jest tryb System.EnterpriseServices .

- Zarówno klasa, jak i metoda uruchamiana w modelu COM + muszą być publiczne (bez klas wewnętrznych i bez metod chronionych lub statycznych).
- Atrybuty klasy i metody: atrybut TransactionOption określa poziom transakcji klasy, tzn. określa, czy transakcje są wyłączone, obsługiwane, czy wymagane. Atrybut AutoComplete metody ExecuteUOW() instruuje COM +, aby zatwierdził transakcję, jeśli nie został zgłoszony żaden nieobsługiwany wyjątek.
- Silne-nazywanie zespołu: zespół musi mieć silną nazwę i musi być zarejestrowany w globalnej pamięci podręcznej zespołu (Global Assembly Cache-GAC). Zespół jest rejestrowany w modelu COM + w sposób jawny lub przez opóźnione rejestrowanie po zarejestrowaniu w systemie GAC.
- Rejestrowanie zespołu w modelu COM +: przygotowanie zespołu do ujawnienia klientom COM. Następnie należy utworzyć bibliotekę typów przy użyciu narzędzia Assembly Registration, regasm.exe.

```
regasm UnmanagedToManagedXa.dll
```

- Zarejestruj zespół w GAC `gacutil /i UnmanagedToManagedXa.dll`.
- Zarejestruj zespół w modelu COM +, korzystając z programu instalacyjnego usług .NET , regsvcs.exe. Patrz biblioteka typów utworzona przez regasm.exe:

```
Regsvcs /appname:UnmanagedToManagedXa /tlb:UnmanagedToManagedXa.tlb  UnmanagedToManagedXa.dll
```

- Zespół jest wdrażany w pamięci GAC, a później jest rejestrowany w COM + przez leniwą rejestrację. Środowisko .NET zajmuje się rejestracją po pierwszym uruchomieniu kodu.

Przykładowy przepływ kodu korzystający z modeli System.EnterpriseServices i System.Transactions z modelem COM + został opisany w następujących sekcjach:

**Przykładowy przepływ kodu przy użyciu modelu System.EnterpriseServices**

```
using System;
using IBM.WMQ;
using IBM.WMQ.Nmqi;
using System.Transactions;
using System.EnterpriseServices;

namespace UnmanagedToManagedXa
{

[ComVisible(true)]
[System.EnterpriseServices.Transaction(System.EnterpriseServices.TransactionOption.Required)]
    public class MyXa : System.EnterpriseServices.ServicedComponent
    {

        public MQQueueManager QMGR = null;
        public MQQueueManager QMGR1 = null;
        public MQQueue QUEUE = null;
        public MQQueue QUEUE1 = null;
        public MQPutMessageOptions pmo = null;
        public MQMessage MSG = null;
```

```
        public MyXa()
        {
        }

        [System.EnterpriseServices.AutoComplete()]
        public void ExecuteUOW()
        {
            QMGR = new MQQueueManager("usemq");

            QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                     MQC.MQOO_INPUT_SHARED +
                                     MQC.MQOO_OUTPUT +
                                     MQC.MQOO_BROWSE);

            pmo = new MQPutMessageOptions();
            pmo.Options = MQC.MQPMO_SYNCPOINT;
            MSG = new MQMessage();
            QUEUE.Put(MSG, pmo);
            QMGR.Disconnect();
        }
    }
}


public void RunNow()
{
   MyXa xa = new MyXa();
   xa.ExecuteUOW();
}
```

**Przykładowy przepływ kodu przy użyciu strumienia System.Transactions dla interakcji z COM +**

```
[STAThread]
public void ExecuteUOW()
{
Hashtable t1 = new Hashtable();
t1.Add(MQC.CHANNEL_PROPERTY, "SYSTEM.DEF.SVRCONN");
t1.Add(MQC.HOST_NAME_PROPERTY, "localhost");
t1.Add(MQC.PORT_PROPERTY, 1414);
t1.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_CLIENT);
TransactionOptions opts = new TransactionOptions();

using(TransactionScope scope = new  TransactionScope(TransactionScopeOption.RequiresNew,
                                opts,     EnterpriseServicesInteropOption.Full)
  {

        QMGR = new MQQueueManager("usemq", t1);
        QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                          MQC.MQOO_INPUT_SHARED +
                                          MQC.MQOO_OUTPUT +
                                          MQC.MQOO_BROWSE);

         pmo = new MQPutMessageOptions();
         pmo.Options = MQC.MQPMO_SYNCPOINT;
         MSG = new MQMessage();
         QUEUE.Put(MSG, pmo);
         scope.Complete();
  }
 QMGR.Disconnect();
}
```

## Tworzenie prostych operacji put i get dla komunikatów w zasięgu TransactionScope

Przykładowe aplikacje C# produktu są dostępne w produkcie IBM MQ. Te proste aplikacje demonstrują umieszczanie i pobieranie komunikatów w obrębie klasy TransactionScope. Po zakończeniu zadania możliwe będzie umieszczanie i pobieranie komunikatów z kolejki lub tematu.

### Zanim rozpoczniesz

Usługa systemu MSDTC musi być uruchomiona i włączona dla transakcji XA.

### O tym zadaniu

Przykład to prosta aplikacja, `SimpleXAPut` i `SimpleXAGet`. Programy `SimpleXAPut` i `SimpleXAGet` są aplikacjami C# dostępnymi w produkcie IBM MQ. `SimpleXAPut` demonstruje użycie MQPUT w obszarze

transakcji rozproszonych z użyciem przestrzeni nazw SystemTransactions . `SimpleXAGet` demonstruje użycie MQGET w obszarze Transakcje rozproszone przy użyciu przestrzeni nazw SystemTransactions .

Plik `SimpleXAPut` znajduje się w katalogu `MQ\tools\dotnet\samples\cs\base`

### Procedura

Aplikacje można uruchamiać z parametrami wiersza komend z katalogu `tools\dotnet\samples\cs\base\bin` .

```
SimpleXAPut.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n
numberOfMsgs]
```

```
SimpleXAGet.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n
numberOfMsgs]
```

gdzie parametry są następujące:

**-destinationURI**

Może to być kolejka lub temat. W przypadku kolejki należy określić wartość `queue://queueName` , a w przypadku tematu-wartość `topic://topicName`.

**-host**

Może to być nazwa hosta, na przykład localhost lub adres IP.

**-port**

Port, na którym działa menedżer kolejek.

**-channel**

Używany kanał połączenia. Wartością domyślną jest SYSTEM.DEF.SVRCONN DEF.SVRCONN

**-transaction**

Wynik transakcji, na przykład zatwierdzenie lub wycofanie.

**-mode**

Tryb transportu, na przykład zarządzany lub niezarządzany.

**-numberOfMsgs**

Liczba komunikatów. Wartość domyślna to 1.

#### Przykład

```
SimpleXAPut -d topic://T01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

```
SimpleXAGet -d queue://Q01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

## Odtwarzanie transakcji w programie IBM MQ .NET

W tej sekcji opisano proces odtwarzania transakcji interfejsu XA produktu IBM MQ .NET w trybie zarządzanym.

### O tym zadaniu

W przypadku rozproszonego przetwarzania transakcji transakcje mogą zostać pomyślnie zakończone, ale mogą istnieć scenariusze, w których transakcja może zakończyć się niepowodzeniem z wielu powodów. Przyczyny te mogą obejmować awarię systemu, awarię sprzętu, błąd sieci, niepoprawne lub niepoprawne

dane, błędy aplikacji lub klęski żywiołowe lub katastrofy spowodowane przez człowieka. Nie jest możliwe zapobieganie niepowodzeniom transakcji. Rozproszony system transakcji musi być w stanie obsłużyć te błędy. Musi być w stanie wykryć i poprawić błędy, gdy wystąpią. Ten proces jest nazywany odtwarzaniem transakcji.

Ważnym aspektem rozproszonego przetwarzania transakcji jest odtworzenie niekompletnych lub wątpliwych transakcji. Konieczne jest uruchomienie odtwarzania, ponieważ część jednostki pracy danej transakcji jest zablokowana do czasu jej odzyskania. Produkt Microsoft.NET z biblioteki klas System.Transactions udostępnia opcję odtwarzania niekompletnych/wątpliwych transakcji. Ta obsługa odtwarzania oczekuje, że Resource Manager będzie obsługiwać dzienniki transakcji i uruchamiać odtwarzanie w razie potrzeby.

W modelu odtwarzania transakcji serwera Microsoft .NET menedżer transakcji (System.Transactionslub Microsoft Distributed Transaction coordinator (MS DTC)) inicjuje, koordynuje i steruje odtwarzaniem transakcji. Menedżery zasobów oparte na protokole OLE Tx (protokół Microsoft XA) udostępniają opcje konfigurowania DTC do sterowania, koordynowania i sterowania odtwarzaniem dla nich. W tym celu menedżerowie zasobów muszą zarejestrować przełącznik XA_Switch w usłudze MS DTC przy użyciu interfejsu rodzimego.

Przełącznik XA_Switch udostępnia punkty wejścia funkcji XA, takich jak xa_start, xa_end i xa_recover w Resource Manager dla rozproszonego koordynatora transakcji.

**Odtwarzanie za pomocą koordynatora transakcji rozproszonych (DTC) Microsoft :**

Microsoft Koordynator rozproszonych transakcji udostępnia dwa rodzaje procesów odtwarzania.

**Zimna regeneracja**

Zimne odtwarzanie jest wykonywane, jeśli proces menedżera transakcji nie powiedzie się, gdy połączenie z menedżerem zasobów XA jest otwarte. Po zrestartowaniu menedżera transakcji odczytuje on dzienniki menedżera transakcji i ponownie nawiązuje połączenie z menedżerem zasobów XA, a następnie inicjuje odtwarzanie.

**Odtwarzanie podczas pracy**

Odtwarzanie podczas przetwarzania jest wykonywane, jeśli menedżer transakcji pozostaje uruchomiony, gdy połączenie między menedżerem transakcji a menedżerem zasobów XA nie powiedzie się z powodu awarii menedżera zasobów XA lub sieci. Po awarii menedżer transakcji okresowo próbuje ponownie nawiązać połączenie z menedżerem zasobów XA. Po ponownym nawiązaniu połączenia menedżer transakcji inicjuje odtwarzanie XA.

Przestrzeń nazw System.Transactions udostępnia zarządzaną implementację rozproszonych transakcji opartych na usłudze MS DTC jako menedżerze transakcji. Udostępnia on podobne funkcje, jak interfejs rodzimy MS DTC, ale w środowisku w pełni zarządzanym. Jedyna różnica dotyczy odtwarzania transakcji. Serwer System.Transactions oczekuje, że menedżery zasobów same będą prowadzić odtwarzanie, a następnie będą koordynować je z menedżerami transakcji (MS DTC). Resource Manager musi poprosić o odtworzenie konkretnej niekompletnej transakcji, a następnie menedżer transakcji akceptuje ją i koordynuje na podstawie rzeczywistego wyniku tej transakcji.

## *Proces odtwarzania transakcji dla produktu IBM MQ .NET*

W tej sekcji opisano sposób odzyskiwania rozproszonych transakcji za pomocą klas IBM MQ .NET .

## Przegląd

Aby odzyskać niekompletną transakcję, wymagane są informacje o odtwarzaniu. Informacje o odtwarzaniu transakcji muszą być rejestrowane w pamięci masowej przez menedżery zasobów. IBM MQ Klasy .NET mają podobną ścieżkę. Informacje o odzyskiwaniu transakcji są protokołowane w kolejce systemowej o nazwie SYSTEM.DOTNET.XARECOVERY.QUEUE.

Odtwarzanie transakcji w produkcie IBM MQ .NET jest procesem dwuetapowym:

1. Protokołowanie informacji o odtwarzaniu transakcji w systemie SYSTEM.DOTNET.XARECOVERY.QUEUE.

2. Odtwarzanie transakcji za pomocą aplikacji monitora XA WmqDotnetXAMonitor.

## SYSTEM.DOTNET.XARECOVERY.QUEUE

SYSTEM SYSTEM.DOTNET.XARECOVERY.QUEUE to kolejka systemowa, która przechowuje informacje o odtwarzaniu transakcji dla niekompletnych transakcji. Ta kolejka jest tworzona podczas tworzenia menedżera kolejek.

Dla każdej transakcji podczas fazy przygotowania trwały komunikat zawierający informacje o odtwarzaniu jest dodawany do systemu SYSTEM.DOTNET.XARECOVERY.QUEUE. Komunikat jest usuwany, jeśli wywołanie zatwierdzenia powiedzie się.

**Uwaga:** Nie wolno usuwać systemu SYSTEM.DOTNET.XARECOVERY.QUEUE .

## Aplikacja WMQDotnetXAMonitor

IBM MQ .NET Aplikacja monitora XA, WmqDotnetXAMonitor, jest aplikacją zarządzaną przez .NET , która monitoruje menedżer kolejek i przetwarza komunikaty w systemie SYSTEM.DOTNET.XARECOVERY.QUEUE i odzyskiwanie niekompletnych transakcji

Jeśli agent kanału komunikatów (MCA) nie może umieścić komunikatu w kolejce docelowej, generuje raport o wyjątku zawierający oryginalny komunikat i umieszcza go w kolejce transmisji, która ma zostać wysłana do kolejki odpowiedzi określonej w oryginalnym komunikacie. (Jeśli kolejka odpowiedzi znajduje się w tym samym menedżerze kolejek co agent MCA, komunikat jest umieszczany bezpośrednio w tej kolejce, a nie w kolejce transmisji).

Następujące transakcje uważa się za niekompletne i są one odzyskiwane:

- Jeśli transakcja jest przygotowana, ale instrukcja COMMIT nie została zakończona w limicie czasu.
- Jeśli transakcja jest przygotowana, ale menedżer kolejek IBM MQ został wyłączony.
- Jeśli transakcja jest przygotowana, ale menedżer transakcji został wyłączony.

  Aplikacja monitora XA musi być uruchamiana w tym samym systemie, w którym działa aplikacja kliencka IBM MQ .NET . Jeśli istnieją aplikacje działające w wielu systemach i nawiązujące połączenie z tym samym menedżerem kolejek, aplikacja XAMonitor WmqDotnetmusi być uruchamiana ze wszystkich systemów. Chociaż na każdym komputerze klienckim znajduje się instancja aplikacji monitora XA działająca w celu odtworzenia aplikacji, każda instancja monitora XA powinna być w stanie zidentyfikować komunikat odpowiadający transakcji, którą koordynował lokalny klient DTC bieżącego monitora XA, aby mógł on ponownie zarejestrować i zakończyć tę transakcję.

**Pojęcia pokrewne**

“Przypadki użycia odtwarzania transakcji dla produktu IBM MQ .NET” na stronie 593
Istnieje kilka różnych przypadków użycia, w których może być konieczne odtworzenie transakcji.

**Zadania pokrewne**

“Korzystanie z aplikacji WMQDotnetXAMonitor” na stronie 594
Klient IBM MQ .NET udostępnia aplikację monitora XA, WmqDotnetXAMonitor, której można użyć do odtwarzania dowolnych niekompletnych transakcji rozproszonych. Aplikacja WmqDotnetXAMonitor nawiązuje połączenie z menedżerem kolejek, w którym transakcje są wątpliwe, a następnie rozstrzyga transakcję na podstawie ustawionych parametrów.

## *Przypadki użycia odtwarzania transakcji dla produktu IBM MQ .NET*

Istnieje kilka różnych przypadków użycia, w których może być konieczne odtworzenie transakcji.

- **IBM MQ Aplikacja używająca pojedynczego DTC i pojedynczej instancji menedżera kolejek:** W tym przypadku użycia po nawiązaniu połączenia z menedżerem kolejek i uruchomieniu jednostki pracy (UoW) w ramach transakcji, jeśli transakcja zakończy się niepowodzeniem i stanie się niekompletna, aplikacja monitora XA odtworzy transakcję i ją zakończy.

  W tym przypadku użycia będzie działać pojedyncza instancja aplikacji monitora XA, ponieważ pojedynczy menedżer kolejek jest powiązany z transakcjami.

- **Wiele aplikacji IBM MQ używających pojedynczego DTC i pojedynczej instancji menedżera kolejek:** W tym przypadku użycia istnieje więcej niż jedna aplikacja IBM MQ w ramach pojedynczego DTC i wszystkie łączą się z tym samym menedżerem kolejek i uruchamiają komendę UoW w ramach transakcji.

  Jeśli transakcje nie powiodą się i staną się niekompletne, aplikacja monitora XA odtworzy je i zakończy transakcje dotyczące wszystkich aplikacji.

  W tym przypadku użycia działa pojedyncza instancja aplikacji monitora XA, ponieważ w transakcjach używany jest jeden menedżer kolejek.
- **Wiele IBM MQ aplikacji, wiele DTC, różne instancje menedżera kolejek:** W tym przypadku użycia istnieje więcej niż jedna aplikacja IBM MQ w różnych DTC (tzn. każda aplikacja działa na innym komputerze) i nawiązuje połączenie z różnymi menedżerami kolejek.

  Jeśli wystąpi niepowodzenie i transakcja stanie się niekompletna, aplikacja monitorująca sprawdza właściwość TransactionManagerw miejscu, w którym znajduje się komunikat, w celu określenia adresu DTC. Jeśli wartość miejsca pobytu TransactionManagerjest zgodna z adresem DTC, pod którym działa monitor, kończy odtwarzanie, w przeciwnym razie kontynuuje wyszukiwanie, dopóki nie zostanie znaleziony komunikat odpowiadający dTC.

  W tym przypadku użycia dla każdego klienta (użytkownika lub komputera) będzie działać tylko jedna instancja aplikacji monitora XA, ponieważ każdy klient ma własny menedżer kolejek używany w transakcjach.
- **Wiele aplikacji IBM MQ , wiele DTC, wiele instancji menedżera kolejek:** W tym przypadku użycia istnieje więcej niż jedna aplikacja IBM MQ w różnych DTC (każda aplikacja jest uruchomiona na innym komputerze) i wszystkie łączą się z tym samym menedżerem kolejek.

  Jeśli wystąpi niepowodzenie i transakcja stanie się niekompletna, aplikacja monitora sprawdza miejsce pobytu TransactionManagerw komunikacie, aby sprawdzić, czy adres DTC i wartość są zgodne z DTC, w którym działa monitor. Jeśli obie wartości są zgodne, odtwarzanie jest kontynuowane do momentu znalezienia komunikatu odpowiadającego dTC.

  W tym przypadku użycia dla każdego klienta (użytkownika lub komputera) będzie działać tylko jedna instancja aplikacji monitora XA, ponieważ każdy klient ma własne powiązanie menedżera kolejek używane w transakcjach.
- **Wiele aplikacji IBM MQ , pojedyncze DTC, różne instancje menedżera kolejek:** W tym przypadku użycia istnieje więcej niż jedna aplikacja IBM MQ w ramach jednego DTC (czyli na komputerze uruchomionych jest więcej niż jedna aplikacja IBM MQ ) i nawiązywane jest połączenie z różnymi menedżerami kolejek.

  Jeśli transakcja nie powiedzie się i stanie się niekompletna, aplikacja monitorująca odtworzy transakcję.

  W tym przypadku użycia będzie działać tak wiele instancji aplikacji monitorowania, z którymi są połączone menedżery kolejek, ponieważ każda aplikacja ma własny menedżer kolejek używany w transakcjach i każda z nich musi zostać odtworzona.

  **Uwaga:** Jeśli aplikacja monitora XA nie działa w tle, można ją uruchomić.

**Pojęcia pokrewne**

“Proces odtwarzania transakcji dla produktu IBM MQ .NET” na stronie 592
W tej sekcji opisano sposób odzyskiwania rozproszonych transakcji za pomocą klas IBM MQ .NET .

**Zadania pokrewne**

“Korzystanie z aplikacji WMQDotnetXAMonitor” na stronie 594
Klient IBM MQ .NET udostępnia aplikację monitora XA, WmqDotnetXAMonitor, której można użyć do odtwarzania dowolnych niekompletnych transakcji rozproszonych. Aplikacja WmqDotnetXAMonitor nawiązuje połączenie z menedżerem kolejek, w którym transakcje są wątpliwe, a następnie rozstrzyga transakcję na podstawie ustawionych parametrów.

## *Korzystanie z aplikacji WMQDotnetXAMonitor*

Klient IBM MQ .NET udostępnia aplikację monitora XA, WmqDotnetXAMonitor, której można użyć do odtwarzania dowolnych niekompletnych transakcji rozproszonych. Aplikacja WmqDotnetXAMonitor

nawiązuje połączenie z menedżerem kolejek, w którym transakcje są wątpliwe, a następnie rozstrzyga transakcję na podstawie ustawionych parametrów.

## O tym zadaniu

Aplikację WMQDotnetXAMonitor należy uruchomić ręcznie. Można go uruchomić w dowolnym momencie. Można go uruchomić po wyświetleniu komunikatów w systemie SYSTEM.DOTNET.XARECOVERY.QUEUE lub może działać w tle przed wykonaniem jakiejkolwiek pracy transakcyjnej z aplikacjami napisanymi za pomocą klas IBM MQ .NET .

Wartości parametrów komendy WMQDotnetXAMonitor można ustawić w wierszu komend lub przy użyciu pliku konfiguracyjnego aplikacji. Wartości podane w pliku konfiguracyjnym aplikacji mają pierwszeństwo przed wartościami ustawionymi w wierszu komend.

Przed IBM MQ 9.3.0połączenie nawiązywane przez program WMQDotnetXAMonitor jest połączeniem niezabezpieczonym.

V 9.3.0 W produkcie IBM MQ 9.3.0istnieje możliwość nawiązania bezpiecznego połączenia z menedżerem kolejek przez ustawienie dodatkowych parametrów dla programu WMQDotnetXAMonitor.

## Procedura

- Aby udostępnić dane wejściowe dla komendy WmqDotNETXAMonitor przy użyciu pliku konfiguracyjnego aplikacji, należy zapoznać się z sekcją “WmqDotUstawienia pliku konfiguracyjnego aplikacji NETXAMonitor” na stronie 597.
- Aby uruchomić aplikację WMQDotnetXAMonitor z poziomu wiersza komend, należy użyć następującej komendy z wymaganymi parametrami:

  Przed IBM MQ 9.3.0:

  ```
  WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i
  ```

  V 9.3.0 W systemie IBM MQ 9.3.0:

  ```
  WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i -k SSL Key
  Repository -s Cipher Spec
  ```

  Parametry, które można określić, są następujące:

  – **-m *QueueManagerNazwa***
  Nazwa menedżera kolejek.

  Opcjonalne

  **-n *ConnectionName***
  Nazwa połączenia w formacie hosta (portu). *ConnectionName* może zawierać więcej niż jedną nazwę połączenia. Wiele nazw połączeń musi być podanych w postaci listy rozdzielanej przecinkami, na przykład `localhost (1414), localhost (1415), localhost (1416)`. Aplikacja WMQDotnetXAMonitor uruchamia odtwarzanie dla każdej nazwy połączenia podanej na liście rozdzielanej przecinkami.

  **-c *ChannelName***
  Nazwa kanału.

  **-i**
  Zakończenie gałęzi heurystycznej.

  Opcjonalne

  V 9.3.0 **-k *Repozytorium kluczy SSL***
  Nazwa repozytorium kluczy SSL. Obsługiwane są następujące wartości:

  - *SYSTEM (jest to wartość domyślna)

- *USER (użytkownik)

Opcjonalne

V 9.3.0 **-s *specyfikacja_szyfru***

Ustawiona CipherSpec musi być jedną z CipherSpecs dla obsługiwanej wersji i najlepiej może być taka sama, jak określona w strategii grupy Windows . Więcej informacji na ten temat zawiera sekcja “Obsługa CipherSpec dla zarządzanego klienta .NET” na stronie 618.

Obowiązkowe w przypadku nawiązywania bezpiecznego połączenia z menedżerem kolejek.

V 9.3.0 **-dn *nazwa_partnera_SSL***

Nazwa węzła sieci SSL używana do sprawdzania nazwy wyróżniającej (DN) certyfikatu z menedżera kolejek węzła sieci.

Opcjonalne

V 9.3.0 **-cl *etykieta certyfikatu***

Nazwa etykiety identyfikującej certyfikat.

Opcjonalne

V 9.3.0 **-sn *OutboundSNI***

Określa, czy podczas inicjowania połączenia TLS lub nazwy hosta należy ustawić wskazanie nazwy serwera (Server Name Indication-SNI) na docelową nazwę kanału IBM MQ dla systemu zdalnego. Obsługiwane są następujące wartości tej opcji:

- CHANNEL (jest to wartość domyślna)
- HOSTNAME
- *

Jeśli nie ustawiono żadnej wartości, zostanie użyta wartość domyślna CHANNEL.

Opcjonalne

V 9.3.0 **-cr *Sprawdzanie odwołań certyfikatów***

Określa, czy ma być wykonywane sprawdzanie odwołań certyfikatów. Obsługiwane są następujące wartości tej opcji:

- Prawda
- false (jest to wartość domyślna)

Opcjonalne

V 9.3.0 **-kr *KeyResetLiczba***

Łączna liczba niezaszyfrowanych bajtów, które zostały wysłane i odebrane w kanale przed ponownym wynegocjowaniem klucza tajnego używanego do szyfrowania.

Wartość domyślna 0 wskazuje, że klucze tajne nigdy nie są renegocjowane

Opcjonalne

Aplikacja WMQDotnetXAMonitor wykonuje następujące działania:

1. Sprawdza głębokość kolejki SYSTEM.DOTNET.XARECOVERY.QUEUE w odstępie 100 sekund.
2. Jeśli zapełnienie kolejki jest większe od zera, przegląda kolejkę w poszukiwaniu komunikatów i sprawdza, czy komunikaty spełniają kryteria niekompletnej transakcji.
3. Jeśli komunikat spełnia niekompletne kryteria transakcji, pobiera go i pobiera informacje o odtwarzaniu transakcji.
4. Określa, czy informacje o odtwarzaniu odnoszą się do lokalnego systemu Microsoft Koordynator rozproszonej transakcji (MS DTC). W takim przypadku program WMQDotnetXAMonitor kontynuuje odtwarzanie transakcji, w przeciwnym razie przechodzi do następnego komunikatu.
5. Wywołuje menedżera kolejek w celu odtworzenia niekompletnej transakcji.

*WmqDotUstawienia pliku konfiguracyjnego aplikacji NETXAMonitor*
Dane wejściowe dla aplikacji monitora XA produktu IBM MQ .NET WmqDotNETXAMonitor można udostępnić przy użyciu pliku konfiguracyjnego aplikacji. Przykładowy plik konfiguracyjny aplikacji jest dostarczany z produktem IBM MQ .NET. Ten przykładowy plik można zmodyfikować zgodnie z wymaganiami.

Wartości wejściowe podane w pliku konfiguracyjnym aplikacji mają najwyższy priorytet. Jeśli wartości wejściowe zostaną podane zarówno w wierszu komend, jak i w pliku konfiguracyjnym aplikacji (“Korzystanie z aplikacji WMQDotnetXAMonitor” na stronie 594 ), to pierwszeństwo mają wartości z pliku konfiguracyjnego aplikacji.

Przykładowy plik konfiguracyjny aplikacji dla wersji wcześniejszych niż IBM MQ 9.3.0.

```
<?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>
```

V 9.3.0 Przykładowy plik konfiguracyjny aplikacji z katalogu IBM MQ 9.3.0.

```
<?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*
Aplikacja Monitor tworzy plik dziennika w katalogu aplikacji na potrzeby rejestrowania postępu monitorowania i statusu odtwarzania transakcji. Rejestrowanie rozpoczyna się od nazwy połączenia i szczegółów kanału w celu wyświetlenia bieżącego menedżera kolejek, dla którego uruchomiono odtwarzanie.

Po rozpoczęciu odtwarzania zostanie zarejestrowany identyfikator MessageId komunikatu odtwarzania transakcji, identyfikator TransactionId niekompletnej transakcji i rzeczywisty wynik transakcji zgodnie z koordynacją menedżera transakcji.

Przykładowy plik dziennika:

```
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
```

# Pisanie i wdrażanie programów IBM MQ .NET

Aby uzyskać dostęp do kolejek systemu IBM MQ za pomocą programu IBM MQ classes for .NET , należy napisać programy w dowolnym języku obsługiwanym przez program .NET , zawierające wywołania, które umieszczają komunikaty w kolejkach IBM MQ i pobierają je z tych kolejek.

Dokumentacja produktu IBM MQ zawiera tylko informacje na temat języków C#, C++ i Visual Basic.

Ta kolekcja tematów zawiera informacje pomocne przy pisaniu aplikacji przeznaczonych do interakcji z systemami IBM MQ . Szczegółowe informacje na temat poszczególnych klas zawiera sekcja Klasy i interfejsy produktu IBM MQ .NET.

## Różnice w połączeniach

Sposób, w jaki program jest uruchamiany dla produktu IBM MQ.NET , zależy od trybów połączenia, które mają być używane.

Jeśli program IBM MQ classes for .NET jest używany jako klient zarządzany, istnieje wiele różnic w porównaniu ze standardowym programem IBM MQ MQI client, ponieważ niektóre funkcje nie są dostępne dla klienta zarządzanego.

Produkt IBM MQ.NET określa typ połączenia, który ma być używany, na podstawie ustawień określonych dla nazwy połączenia, nazwy kanału, wartości dostosowania NMQ_MQ_MQ_LIB i właściwości MQC.TRANSPORT_PROPERTY.

### *Zarządzane połączenia klienta*

Jeśli produkt IBM MQ classes for .NET jest używany jako klient zarządzany, istnieje wiele różnic w porównaniu ze standardowym klientem IBM MQ MQI client.

Następujące funkcje nie są dostępne dla klienta zarządzanego:

- Kompresja kanału
- Łączenie kanału w łańcuch

Próba użycia tych opcji z klientem zarządzanym spowoduje zwrócenie wyjątku MQException. Jeśli błąd zostanie wykryty po zakończeniu połączenia przez klienta, zostanie użyty kod przyczyny MQRC_ENVIRONMENT_ERROR. Jeśli zostanie wykryty na końcu serwera, zostanie użyty kod przyczyny zwrócony przez serwer.

Wyjścia kanału zapisane dla niezarządzanego klienta nie działają. Należy napisać nowe wyjścia specjalnie dla klienta zarządzanego. Sprawdź, czy w tabeli definicji kanału klienta (CCDT) nie określono niepoprawnych wyjść kanału.

Nazwa zarządzanego wyjścia kanału może mieć długość do 999 znaków. Jeśli jednak do określenia nazwy wyjścia kanału jest używana tabela definicji kanału (CCDT), nazwa ta jest ograniczona do 128 znaków.

Komunikacja jest obsługiwana tylko przez protokół TCP/IP.

Jeśli menedżer kolejek zostanie zatrzymany za pomocą komendy **endmqm** , zamknięcie kanału połączenia serwera z klientem zarządzanym przez program .NET może zająć więcej czasu niż zamknięcie kanału połączenia serwera z innymi klientami.

Jeśli wartość właściwości *NMQ_MQ_LIB* ustawiono na wartość `managed` w celu użycia diagnostyki problemu zarządzanego produktu IBM MQ , żaden z parametrów -i, -p, -s, -b ani -c komendy **strmqtrc** nie jest obsługiwany.

Zarządzana aplikacja .NET używająca transakcji XA nie będzie działać z menedżerem kolejek produktu z/OS . Próba nawiązania połączenia zarządzanego klienta .NET z menedżerem kolejek produktu z/OS kończy się niepowodzeniem z błędem MQRC_UOW_ENLISTMENT_ERROR (mqrc=2354) w wywołaniu MQOPEN. Jednak zarządzana aplikacja .NET używająca transakcji XA będzie działać z rozproszonym menedżerem kolejek.

### *Definiowanie typu połączenia, który ma być używany*

Typ połączenia jest określany przez ustawienie nazwy połączenia, nazwy kanału, wartości dostosowania NMQ_MQ_LIB i właściwości MQC.TRANSPORT_PROPERTY.

Nazwę połączenia można określić w następujący sposób:

- Jawnie w konstruktorze MQQueueManager :

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- Przez ustawienie właściwości MQC.HOST_NAME_PROPERTY i opcjonalnie MQC.PORT_PROPERTY w pozycji tabeli mieszającej w konstruktorze MQQueueManager :

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Jako jawne wartości MQEnvironment

```
MQEnvironment.Hostname
```

  `MQEnvironment.Port` (opcjonalnie).
- Przez ustawienie właściwości MQC.HOST_NAME_PROPERTY i opcjonalnie MQC.PORT_PROPERTY w tabeli mieszającej MQEnvironment.properties .

Nazwę kanału można określić w następujący sposób:

- Jawnie w konstruktorze MQQueueManager :

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- Przez ustawienie właściwości MQC.CHANNEL_PROPERTY w pozycji tabeli mieszającej w konstruktorze MQQueueManager :

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Jako jawna wartość MQEnvironment

```
MQEnvironment.Channel
```

- Przez ustawienie właściwości MQC.CHANNEL_PROPERTY w tabeli mieszającej MQEnvironment.properties .

Właściwość transportu można określić w następujący sposób:

- Przez ustawienie właściwości MQC.TRANSPORT_PROPERTY w pozycji tabeli mieszającej w konstruktorze MQQueueManager :

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Przez ustawienie właściwości MQC.TRANSPORT_PROPERTY w tabeli mieszającej MQEnvironment.properties .

Wybierz wymagany typ połączenia, używając jednej z następujących wartości:

MQC.TRANSPORT_MQSERIES_BINDINGS -połączenie jako serwer
MQC.TRANSPORT_MQSERIES_CLIENT -nawiąż połączenie jako klient inny niż XA
MQC.TRANSPORT_MQSERIES_XACLIENT -łączenie jako klient XA
MQC.TRANSPORT_MQSERIES_MANAGED -nawiąż połączenie jako klient niezarządzany XA

Aby jawnie wybrać typ połączenia, można ustawić wartość dostosowania NMQ_MQ_MQ_LIB w sposób przedstawiony w poniższej tabeli.

| **Wartość z właściwości NMQ_MQ_LIB** | **Typ połączenia** |
|---|---|
| mqic.dll | Połącz jako klient inny niż XA |
| mqicxa.dll | Połącz jako klient XA |
| mqm.dll | Połącz jako serwer lub jako klient inny niż XA |
| usługi | Połącz jako klient niezarządzany XA |

**Uwaga:** Wartości mqic32.dll i mqic32xa.dll są akceptowane jako synonimy biblioteki mqic.dll i biblioteki mqicxa.dll w celu zapewnienia zgodności z wcześniejszymi wersjami. Jednak biblioteki mqm.dll i mqm.pdb są tylko częścią pakietu klienta począwszy od wersji IBM WebSphere MQ 7.1 .

Jeśli zostanie wybrany typ połączenia, który jest niedostępny w danym środowisku, na przykład użytkownik określi plik mqic32xa.dll i nie będzie obsługiwać interfejsu XA, produkt IBM MQ.NET zgłosi wyjątek.

Ustawienie właściwości NMQ_MQ_MQ_LIB na wartość managed powoduje, że klient używa zarządzanych IBM MQ testów diagnostycznych problemów, konwersji danych .NET i innych zarządzanych funkcji niskiego poziomu IBM MQ .

Wszystkie pozostałe wartości właściwości NMQ_MQ_LIB powodują, że proces produktu .NET używa niezarządzanych testów diagnostycznych i konwersji danych produktu IBM MQ oraz innych niezarządzanych funkcji języka IBM MQ niskiego poziomu (przy założeniu, że w systemie jest zainstalowany produkt IBM MQ MQI client lub serwer).

Program IBM MQ.NET wybiera typ połączenia w następujący sposób:

1. Jeśli MQC.TRANSPORT_PROPERTY jest określona. Łączy się ona zgodnie z wartością MQC.TRANSPORT_PROPERTY.

   Należy jednak zauważyć, że ustawienie MQC.TRANSPORT_PROPERTY na MQC.TRANSPORT_MQSERIES_MANAGED nie gwarantuje, że proces klienta będzie uruchamiany jako zarządzany. Nawet przy takim ustawieniu klient nie jest zarządzany w następujących przypadkach:

   - Jeśli inny wątek w procesie nawiązał połączenie z produktem MQC.TRANSPORT_PROPERTY została ustawiona na wartość inną niż MQC.TRANSPORT_MQSERIES_MANAGED.
   - Jeśli wartość właściwości NMQ_MQ_MQ_LIB nie jest ustawiona na wartość managed, testy diagnostyczne problemów, konwersja danych i inne funkcje niskiego poziomu nie są w pełni zarządzane (przy założeniu, że w systemie jest zainstalowany produkt IBM MQ MQI client lub serwer).
2. Jeśli nazwa połączenia została określona bez nazwy kanału lub nazwa kanału została określona bez nazwy połączenia, zgłaszany jest błąd.
3. Jeśli określono zarówno nazwę połączenia, jak i nazwę kanału:

- Jeśli właściwość NMQ_MQ_MQ_LIB ma wartość mqic32xa.dll, nawiązywane jest połączenie jako klient XA.
- Jeśli wartość właściwości NMQ_MQ_MQ_LIB jest ustawiona na wartość managed, łączy się ona jako klient zarządzany.
- W przeciwnym razie łączy się jako klient inny niż XA.

4. Jeśli określono wartość NMQ_MQ_MQ_LIB, połączenie jest nawiązywane zgodnie z wartością właściwości NMQ_MQ_LIB.
5. Jeśli zainstalowany jest serwer IBM MQ , łączy się on jako serwer.
6. Jeśli zainstalowany jest serwer IBM MQ MQI client , łączy się on jako klient inny niż XA.
7. W przeciwnym razie łączy się jako klient zarządzany.

## Windows Korzystanie z szablonu projektu produktu IBM MQ .NET

Klient produktu IBM MQ .NET umożliwia korzystanie z szablonu projektu w celu ułatwienia tworzenia aplikacji produktu .NET Core .

### Zanim rozpoczniesz

W systemie musi być zainstalowany system Microsoft Visual Studio 2017lub nowszy i .NET Core 2.1 .

Należy skopiować szablon .NET z

```
&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\base\ProjectTemplates\IBMMQ.NETClientApp.zip
```

do katalogu

```
&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates
```

katalog, gdzie:

- *& MQ_INSTALL_ROOT* to katalog główny instalacji.
- *& USER_HOME_DIRECTORY* jest katalogiem osobistym.

Aby pobrać szablon, należy zatrzymać i ponownie uruchomić program Microsoft Visual Studio .

### O tym zadaniu

Szablon projektu .NET zawiera pewien wspólny kod, którego można użyć do tworzenia aplikacji. Wbudowany kod umożliwia nawiązanie połączenia z menedżerem kolejek produktu IBM MQ i wykonanie operacji umieszczania (put) lub pobierania (get) przez zmodyfikowanie właściwości w wbudowanym kodzie.

### Procedura

1. Otwórz program Microsoft Visual Studio.
2. Kliknij opcję **Plik**, a następnie opcję **Nowy** i opcję **Projekt**.
3. W oknie *Tworzenie nowego projektu*wybierz opcję `IBM MQ .NET Client App (.NET Core)` i kliknij przycisk **Dalej**.
4. W oknie *Konfigurowanie nowego projektu* zmień nazwę projektu w polu *Nazwa projektu* i kliknij przycisk **Utwórz** , aby utworzyć projekt .NET .

   `MQDotnetApp.cs` to plik, który jest tworzony razem z plikiem projektu. Ten plik zawiera kod, który nawiązuje połączenie z menedżerem kolejek i wykonuje operację umieszczania i pobierania.

   Właściwości połączenia są ustawione na wartości domyślne:

   - MQC.CONNECTION_NAME_PROPERTY jest ustawiony na wartość *localhost (1414)*

- MQC.CHANNEL_PROPERTY jest ustawiony na wartość *DOTNET.SVRCONN*

Kolejka jest ustawiona na *Q1*i można odpowiednio zmodyfikować te właściwości.

5. Skompiluj i uruchom aplikację.

**Pojęcia pokrewne**

Komponenty i opcje produktu IBM MQ

Środowisko wykonawcze aplikacji .NET -tylko Windows

## Pliki konfiguracyjne produktu IBM MQ classes for .NET

Aplikacja kliencka .NET może korzystać z pliku konfiguracyjnego IBM MQ MQI client oraz, jeśli używany jest typ połączenia zarządzanego, z pliku konfiguracyjnego aplikacji .NET . Ustawienia w pliku konfiguracyjnym aplikacji mają priorytet.

### plik konfiguracyjny klienta

Aplikacja kliencka IBM MQ classes for .NET może korzystać z pliku konfiguracyjnego klienta w taki sam sposób, jak z dowolnego innego serwera IBM MQ MQI client. Ten plik ma zwykle nazwę `mqclient.ini`, ale można podać inną nazwę pliku. Więcej informacji na temat pliku konfiguracyjnego klienta zawiera sekcja IBM MQ MQI client , `mqclient.ini`.

Tylko następujące atrybuty w pliku konfiguracyjnym IBM MQ MQI client odnoszą się do systemu IBM MQ classes for .NET. Jeśli zostaną podane inne atrybuty, nie będzie to miało żadnego efektu.

*Tabela 77. Atrybuty pliku konfiguracyjnego klienta, które dotyczą systemu IBM MQ classes for .NET*

| **sekcja** | **Atrybut** |
|---|---|
| Kanały | CCSID |
| Kanały | Katalog ChannelDefinition |
| Kanały | Plik ChannelDefinition |
| Kanały | ReconDelay |
| Kanały | DefRecon |
| Kanały | MQReconnectTimeout |
| Kanały | ServerConnectionParametry |
| Kanały | Put1DefaultAlwaysSync |
| Kanały | PasswordProtection |
| ŚcieżkaClientExit | ExitsDefaultPath |
| ŚcieżkaClientExit | ExitsDefaultPath64 |
| MessageBuffer | MaximumSize |
| MessageBuffer | PurgeTime |
| MessageBuffer | UpdatePercentage |
| V 9.3.0 V 9.3.0 Zabezpieczenia | Plik MQIInitialKey |
| V 9.3.0 V 9.3.0 Protokół SSL | SSLKeyRepository |
| V 9.3.0 V 9.3.0 Protokół SSL | Hasło SSLKeyRepository |
| TCP | ClntRcvBufSize |
| TCP | ClntSndBufSize |

| *Tabela 77. Atrybuty pliku konfiguracyjnego klienta, które dotyczą systemu IBM MQ classes for .NET (kontynuacja)* | |
|---|---|
| **sekcja** | **Atrybut** |
| TCP | IPAddressVersion |
| TCP | KeepAlive |

Każdy z tych atrybutów można przesłonić przy użyciu odpowiedniej zmiennej środowiskowej.

### Plik konfiguracyjny aplikacji

Jeśli używany jest typ połączenia zarządzanego, można również nadpisać plik konfiguracyjny klienta IBM MQ i równoważne zmienne środowiskowe za pomocą pliku konfiguracyjnego aplikacji .NET .

Ustawienia pliku konfiguracyjnego aplikacji .NET są uwzględniane tylko podczas uruchamiania z typem połączenia zarządzanego i są ignorowane w przypadku innych typów połączeń.

Plik konfiguracyjny aplikacji .NET i jego format są definiowane przez produkt Microsoft do ogólnego użytku w środowisku .NET , ale nazwy sekcji, klucze i wartości wymienione w tej dokumentacji są specyficzne dla produktu IBM MQ.

Plik konfiguracyjny aplikacji .NET ma format kilku *sekcji*. Każda sekcja zawiera jeden lub więcej *kluczy*, a z każdym kluczem powiązana jest *wartość*. W poniższym przykładzie przedstawiono sekcje, klucze i wartości używane w pliku konfiguracyjnym aplikacji .NET do sterowania właściwością TCP/IP KeepAlive :

```
<configuration>
  <configSections>
    <section name="TCP" type="System.Configuration.NameValueSectionHandler"/>
  </configSections>
  <TCP>
    <add key="KeepAlive" value="true"></add>
  </TCP>
<configuration>
```

Słowa kluczowe używane w nazwach i kluczach sekcji pliku konfiguracyjnego aplikacji .NET są dokładnie zgodne ze słowami kluczowymi dla sekcji i atrybutów zdefiniowanych w pliku konfiguracyjnym klienta.

Sekcja `<configSections>` musi być pierwszym elementem potomnym elementu `<configuration>` .

Więcej informacji na ten temat zawiera dokumentacja produktu Microsoft .

## Przykładowy fragment kodu C# do użycia z opcją .NET

Fragment kodu C# wskazujący, że aplikacja nawiązuje połączenie z menedżerem kolejek, umieszcza komunikat w kolejce i odbiera odpowiedź.

Poniższy fragment kodu C# przedstawia aplikację, która wykonuje trzy działania:

1. Nawiązywanie połączenia z menedżerem kolejek
2. Umieść komunikat w systemie SYSTEM.DEFAULT.LOCAL.QUEUE
3. Pobierz komunikat z powrotem

Przedstawiono również sposób zmiany typu połączenia.

```
// ============================================================================
// Licensed Materials - Property of IBM
// 5724-H72
// (c) Copyright IBM Corp. 2003, 2024
// ============================================================================
using System;
using System.Collections;

using IBM.WMQ;
```

```
class MQSample
{
  // The type of connection to use, this can be:-
  // MQC.TRANSPORT_MQSERIES_BINDINGS for a server connection.
  // MQC.TRANSPORT_MQSERIES_CLIENT for a non-XA client connection
  // MQC.TRANSPORT_MQSERIES_XACLIENT for an XA client connection
  // MQC.TRANSPORT_MQSERIES_MANAGED for a managed client connection
  const String connectionType = MQC.TRANSPORT_MQSERIES_CLIENT;

  // Define the name of the queue manager to use (applies to all connections)
  const String qManager = "your_Q_manager";

  // Define the name of your host connection (applies to client connections only)
  const String hostName = "your_hostname";

  // Define the name of the channel to use (applies to client connections only)
  const String channel = "your_channelname";



  /// <summary>
  /// Initialise the connection properties for the connection type requested
  /// </summary>
  /// <param name="connectionType">One of the MQC.TRANSPORT_MQSERIES_ values</param>
  static Hashtable init(String connectionType)
  {
    Hashtable connectionProperties = new Hashtable();

    // Add the connection type
    connectionProperties.Add(MQC.TRANSPORT_PROPERTY, connectionType);

    // Set up the rest of the connection properties, based on the
    // connection type requested
    switch(connectionType)
    {
      case MQC.TRANSPORT_MQSERIES_BINDINGS:
        break;
      case MQC.TRANSPORT_MQSERIES_CLIENT:
      case MQC.TRANSPORT_MQSERIES_XACLIENT:
      case MQC.TRANSPORT_MQSERIES_MANAGED:
        connectionProperties.Add(MQC.HOST_NAME_PROPERTY, hostName);
        connectionProperties.Add(MQC.CHANNEL_PROPERTY, channel);
        break;
    }

    return connectionProperties;
  }
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static int Main(string[] args)
  {
    try
    {
      Hashtable connectionProperties = init(connectionType);

      // Create a connection to the queue manager using the connection
      // properties just defined
      MQQueueManager qMgr = new MQQueueManager(qManager, connectionProperties);

      // Set up the options on the queue we want to open
      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;

      // Now specify the queue that we want to open,and the open options
      MQQueue system_default_local_queue =
        qMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", openOptions);

      // Define 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
```

## Konfigurowanie środowiska IBM MQ

Przed użyciem połączenia klienta do nawiązania połączenia z menedżerem kolejek należy skonfigurować środowisko produktu IBM MQ .

**Uwaga:** Ten krok nie jest wymagany w przypadku korzystania z produktu IBM MQ classes for .NET w trybie powiązań serwera.

Interfejs programistyczny produktu .NET umożliwia korzystanie z wartości dostosowania NMQ_MQ_MQ_LIB, ale także z klasy MQEnvironment. Ta klasa umożliwia określenie szczegółów, które mają być używane podczas próby nawiązania połączenia, takich jak wymienione na poniższej liście:

- Nazwa kanału
- Nazwa hosta
- Numer portu
- Wyjścia kanału
- Parametry SSL
- Nazwa i hasło użytkownika

Pełne informacje na temat klasy MQEnvironment zawiera sekcja Klasa produktu MQEnvironment.NET .

Aby określić nazwę kanału i nazwę hosta, należy użyć następującego kodu:

```
MQEnvironment.Hostname = "host.domain.com";
MQEnvironment.Channel  = "client.channel";
```

Domyślnie klienty próbują połączyć się z programem nasłuchującym IBM MQ na porcie 1414. Aby określić inny port, użyj kodu:

```
MQEnvironment.Port = nnnn;
```

## Nawiązywanie i rozłączanie połączenia z menedżerem kolejek

Po skonfigurowaniu środowiska IBM MQ można nawiązać połączenie z menedżerem kolejek.

Aby nawiązać połączenie z menedżerem kolejek, należy utworzyć nową instancję klasy MQQueueManager :

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

Aby rozłączyć się z menedżerem kolejek, należy wywołać metodę `Disconnect` w menedżerze kolejek:

```
queueManager.Disconnect();
```

Podczas próby nawiązania połączenia z menedżerem kolejek wymagane jest uprawnienie do wykonywania zapytania ( `inq`) w menedżerze kolejek. Próba nawiązania połączenia nie powiedzie się bez uprawnienia do uzyskiwania informacji.

Jeśli zostanie wywołana metoda `Disconnect` , wszystkie otwarte kolejki i procesy, do których uzyskano dostęp za pośrednictwem tego menedżera kolejek, zostaną zamknięte. Dobrą praktyką programowania jest jednak jawne zamykanie tych zasobów po zakończeniu ich używania. Aby zamknąć zasoby, należy użyć metody `Close` dla obiektu powiązanego z każdym zasobem.

Metody `Commit` i `Backout` w menedżerze kolejek zastępują wywołania MQCMIT i MQBACK używane z interfejsem proceduralnym.

## Uzyskiwanie dostępu do kolejek i tematów

Dostęp do kolejek i tematów można uzyskać za pomocą metod menedżera MQQueueManager lub odpowiednich konstruktorów.

Aby uzyskać dostęp do kolejek, należy użyć metod klasy MQQueueManager . Struktura deskryptora obiektu (object descriptor structure-MQOD) jest zwijana w parametry tych metod. Aby na przykład otworzyć kolejkę w menedżerze kolejek reprezentowanym przez obiekt MQQueueManager o nazwie queueManager, należy użyć następującego kodu:

```
MQQueue queue = queueManager.AccessQueue("qName",
                                         MQC.MQOO_OUTPUT,
                                         "qMgrName",
                                         "dynamicQName",
                                         "altUserId");
```

Parametr *options* jest taki sam jak parametr Options w wywołaniu MQOPEN.

Metoda AccessQueue zwraca nowy obiekt klasy MQQueue.

Po zakończeniu korzystania z kolejki użyj metody Close (), aby ją zamknąć, tak jak w poniższym przykładzie:

```
queue.Close();
```

W produkcie IBM MQ .NETmożna również utworzyć kolejkę za pomocą konstruktora MQQueue. Parametry są dokładnie takie same, jak w przypadku metody accessQueue , z dodanym parametrem menedżera kolejek określającym instancję obiektu MQQueueManager , który ma być używany. Na przykład:

```
MQQueue queue = new MQQueue(queueManager,
```

```
                                "qName",
                                MQC.MQOO_OUTPUT,
                                "qMgrName",
                                "dynamicQName",
                                "altUserId");
```

Skonstruowanie obiektu kolejki w ten sposób umożliwia napisanie własnych podklas kolejki MQQueue.

Podobnie można uzyskać dostęp do tematów za pomocą metod klasy MQQueueManager . Użyj metody AccessTopic(), aby otworzyć temat. Zwraca nowy obiekt klasy MQTopic. Po zakończeniu korzystania z tematu użyj metody Close () tematu MQTopic, aby go zamknąć.

Temat można również utworzyć przy użyciu konstruktora MQTopic. Istnieje wiele konstruktorów tematów; więcej informacji na ten temat zawiera sekcja Klasa MQTopic.NET.

## Obsługa komunikatów

Komunikaty są obsługiwane przy użyciu metod klasy kolejki lub klasy tematu. Aby zbudować nowy komunikat, utwórz nowy obiekt MQMessageobject.

Umieszczanie komunikatów w kolejkach lub tematach przy użyciu metody Put () klasy MQQueue lub MQTopic. Pobieranie komunikatów z kolejek lub tematów przy użyciu metody Get () klasy MQQueue lub MQTopic. W przeciwieństwie do interfejsu proceduralnego, w którym komendy MQPUT i MQGET umieszczają i pobierają tablice bajtów, komenda IBM MQ classes for .NET umieszczają i pobierają instancje klasy MQMessage. Klasa MQMessage hermetyzuje bufor danych zawierający rzeczywiste dane komunikatu wraz ze wszystkimi parametrami deskryptora komunikatu (MQMD) opisującymi ten komunikat.

Aby zbudować nowy komunikat, należy utworzyć nową instancję klasy MQMessage i użyć metod WriteXXX w celu umieszczenia danych w buforze komunikatów.

Podczas tworzenia nowej instancji komunikatu wszystkie parametry MQMD są automatycznie ustawiane na wartości domyślne zgodnie z definicją w sekcji Wartości początkowe i deklaracje języka dla deskryptora MQMD. Metoda Put () klasy MQQueue przyjmuje również jako parametr instancję klasy opcji MQPutMessage. Ta klasa reprezentuje strukturę MQPMO. Poniższy przykład tworzy komunikat i umieszcza go w kolejce:

```
// 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);
```

Metoda Get () obiektu MQQueue zwraca nową instancję obiektu MQMessage, która reprezentuje komunikat właśnie pobierany z kolejki. Jako parametr przyjmuje również instancję klasy opcji MQGetMessage. Ta klasa reprezentuje strukturę MQGMO.

Nie ma potrzeby określania maksymalnej wielkości komunikatu, ponieważ metoda Get () automatycznie dopasowuje wielkość swojego wewnętrznego buforu, aby dopasować ją do komunikatu przychodzącego. Użyj metod ReadXXX klasy MQMessage, aby uzyskać dostęp do danych w zwróconym komunikacie.

W poniższym przykładzie przedstawiono sposób pobierania komunikatu z kolejki:

```
// 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();
```

Format liczb używany przez metody odczytu i zapisu można zmienić, ustawiając zmienną elementu *encoding* .

Zestaw znaków, który ma być używany do odczytywania i zapisywania łańcuchów, można zmienić, ustawiając zmienną elementu *characterSet* .

Więcej informacji na ten temat zawiera sekcja Klasa MQMessage.NET .

**Uwaga:** Metoda WriteUTF() obiektu MQMessage automatycznie koduje długość łańcucha oraz zawarte w nim bajty Unicode. Gdy komunikat zostanie odczytany przez inny program .NET (przy użyciu metody ReadUTF()), jest to najprostszy sposób wysyłania informacji o łańcuchu.

### *Obsługa właściwości komunikatu*

Właściwości komunikatu umożliwiają wybieranie komunikatów lub pobieranie informacji o komunikacie bez uzyskiwania dostępu do jego nagłówków. Klasa MQMessage zawiera metody służące do pobierania i ustawiania właściwości.

Właściwości komunikatu umożliwiają aplikacji wybieranie komunikatów do przetworzenia lub pobieranie informacji o komunikacie bez uzyskiwania dostępu do nagłówków MQMD lub MQRFH2 . Ułatwiają również komunikację między aplikacjami IBM MQ i JMS . Więcej informacji na temat właściwości komunikatu w sekcji IBM MQzawiera sekcja Właściwości komunikatu.

Klasa MQMessage udostępnia wiele metod służących do pobierania i ustawiania właściwości zgodnie z typem danych właściwości. Metody get mają nazwy w formacie Get * Property, a metody set mają nazwy w formacie Set * Property, gdzie gwiazdka (*) reprezentuje jeden z następujących łańcuchów:

- Boolowski
- Bajt
- Bajty
- Podwójny
- Zmiennopozycyjna
- Wew.
- Int2
- Int4
- Int8
- Długa liczba całkowita
- Obiekt
- Krótki
- Łańcuch

Na przykład, aby uzyskać właściwość IBM MQ myproperty (łańcuch znaków), należy użyć wywołania `message.GetStringProperty('myproperty')`. Opcjonalnie można przekazać deskryptor właściwości, który zostanie zakończony ( IBM MQ ).

## Obsługa błędów

Obsługa błędów wynikających z użycia bloków `try` i `catch` w systemie IBM MQ classes for .NET .

Metody w interfejsie .NET nie zwracają kodu zakończenia ani kodu przyczyny. Zamiast tego zgłaszają wyjątek za każdym razem, gdy kod zakończenia i kod przyczyny wynikające z wywołania IBM MQ nie są zerowe. Upraszcza to logikę programu, dzięki czemu nie trzeba sprawdzać kodów powrotu po każdym wywołaniu funkcji IBM MQ. Możesz zdecydować, w którym punkcie programu chcesz poradzić sobie

z możliwością wystąpienia awarii. W tych punktach można otoczyć kod blokami `try` i `catch` , tak jak w poniższym przykładzie:

```
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 );
}
```

## Pobieranie i ustawianie wartości atrybutów

Klasy MQManagedObject, MQQueue i MQQueueManager zawierają metody umożliwiające pobieranie i ustawianie ich wartości atrybutów. Należy zauważyć, że w przypadku kolejki MQQueue metody działają tylko wtedy, gdy podczas otwierania kolejki zostaną podane odpowiednie opcje zapytania i ustawienia.

W przypadku wspólnych atrybutów klasy MQQueueManager i MQQueue dziedziczą z klasy o nazwie MQManagedObject. Ta klasa definiuje interfejsy Inquire () i Set ().

Podczas tworzenia nowego obiektu menedżera kolejek przy użyciu operatora *nowy* jest on automatycznie otwierany do odpytywania. Jeśli do uzyskania dostępu do obiektu kolejki używana jest metoda AccessQueue(), obiekt ten nie jest automatycznie otwierany dla operacji zapytania lub ustawiania, może to powodować problemy z niektórymi typami kolejek zdalnych. Aby użyć metod Inquire i Set oraz ustawić właściwości w kolejce, należy podać odpowiednie opcje inquire i set w parametrze openOptions metody AccessQueue().

Metody inquire i set przyjmują trzy parametry:

- tablica selektorów
- Tablica intAttrs
- Tablica charAttrs

Parametry SelectorCount, IntAttri CharAttrLength znalezione w MQINQ nie są potrzebne, ponieważ długość tablicy jest zawsze znana. Poniższy przykład przedstawia sposób tworzenia zapytania w kolejce:

```
//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);
```

Można zapytać o wszystkie atrybuty tych obiektów. Podzbiór atrybutów jest prezentowany jako właściwości obiektu. Listę atrybutów obiektu zawiera sekcja Atrybuty obiektów. Informacje o właściwościach obiektu zawiera opis odpowiedniej klasy.

## Programy wielowątkowe

Środowisko wykonawcze .NET jest z natury wielowątkowe. Program IBM MQ classes for .NET umożliwia współużytkowanie obiektu menedżera kolejek przez wiele wątków, ale zapewnia synchronizację całego dostępu do docelowego menedżera kolejek.

Rozważmy prosty program łączący się z menedżerem kolejek i otwierający kolejkę podczas uruchamiania. Program wyświetli pojedynczy przycisk na ekranie. Gdy użytkownik kliknie ten przycisk, program pobiera komunikat z kolejki. W takiej sytuacji inicjowanie aplikacji odbywa się w jednym wątku, a kod wykonywany w odpowiedzi na naciśnięcie przycisku jest wykonywany w osobnym wątku (wątku interfejsu użytkownika).

Implementacja produktu IBM MQ .NET zapewnia, że dla konkretnego połączenia (instancja obiektuMQQueueManager ) wszystkie dostępy do docelowego menedżera kolejek produktu IBM MQ są zsynchronizowane. Zachowanie domyślne polega na tym, że wątek, który chce wywołać menedżera kolejek, jest blokowany do czasu zakończenia wszystkich innych wywołań w toku dla tego połączenia. Jeśli wymagany jest jednoczesny dostęp do tego samego menedżera kolejek z wielu wątków w programie, należy utworzyć nowy obiekt MQQueueManager dla każdego wątku, który wymaga dostępu współbieżnego. (Jest to równoważne z wywołaniem osobnego wywołania MQCONN dla każdego wątku).

Jeśli domyślne opcje połączenia są przesłonięte przez produkt MQC.MQCNO_HANDLE_SHARE_NONE lub MQC.MQCNO_SHARE_NO_BLOCK oznacza, że menedżer kolejek nie jest już zsynchronizowany.

## Używanie tabeli definicji kanału klienta z produktem .NET

Tabela definicji kanału klienta (CCDT) może być używana z tabelą IBM MQ classes for .NET. Położenie tabeli definicji kanału klienta można określić na różne sposoby, w zależności od tego, czy używane jest połączenie zarządzane, czy niezarządzane.

### Typ połączenia niezarządzanego klienta innego niż XA lub niezarządzanego klienta XA

W przypadku połączenia niezarządzanego można określić położenie tabeli definicji kanału klienta na dwa sposoby:

- Używając zmiennych środowiskowych MQCHLLIB do określenia katalogu, w którym znajduje się tabela, i MQCHLTAB do określenia nazwy pliku tabeli.
- Używany jest plik konfiguracyjny klienta. W sekcji CHANNELS użyj atrybutów **ChannelDefinitionDirectory** , aby określić katalog, w którym znajduje się tabela, oraz **ChannelDefinitionFile** , aby określić nazwę pliku.

Jeśli położenie jest określone zarówno w pliku konfiguracyjnym klienta, jak i za pomocą zmiennych środowiskowych, pierwszeństwo mają zmienne środowiskowe. Za pomocą tej funkcji można określić standardowe położenie w pliku konfiguracyjnym klienta i w razie potrzeby nadpisać je za pomocą zmiennych środowiskowych.

### Typ połączenia zarządzanego klienta

W przypadku połączenia zarządzanego można określić położenie tabeli definicji kanału klienta na trzy sposoby:

- Przy użyciu pliku konfiguracyjnego aplikacji .NET . W sekcji CHANNELS użyj kluczy **ChannelDefinitionDirectory** , aby określić katalog, w którym znajduje się tabela, oraz **ChannelDefinitionFile** , aby określić nazwę pliku.
- Używając zmiennych środowiskowych MQCHLLIB do określenia katalogu, w którym znajduje się tabela, i MQCHLTAB do określenia nazwy pliku tabeli.
- Używany jest plik konfiguracyjny klienta. W sekcji CHANNELS użyj atrybutów **ChannelDefinitionDirectory** , aby określić katalog, w którym znajduje się tabela, oraz **ChannelDefinitionFile** , aby określić nazwę pliku.

Jeśli położenie jest określone w więcej niż jeden z tych sposobów, zmienne środowiskowe mają priorytet nad plikiem konfiguracyjnym klienta, a plik konfiguracyjny aplikacji .NET ma priorytet nad obydwiema innymi metodami. Za pomocą tej funkcji można określić standardowe położenie w pliku konfiguracyjnym klienta i w razie potrzeby nadpisać je przy użyciu zmiennych środowiskowych lub pliku konfiguracyjnego aplikacji.

W przypadku używania tabeli definicji kanału (CCDT) z grupowaniem menedżera kolejek w przypadku wersji wcześniejszych niż IBM MQ 9.3.0występuje różnica w zachowaniu zarządzanego klienta .NET i klientów C produktu IBM MQ Java . Jeśli plik CCDT zawiera grupę menedżerów kolejek z trzema menedżerami kolejek i trzema jawnymi CLNTCONNs dla tych samych trzech menedżerów kolejek, a aplikacja udostępnia "*" jako menedżer kolejek, klienty C i Java zwracają wartość MQRC_Q_MGR_NAME_ERROR. Jednak zarządzany klient .NET używa pierwszego dostępnego CLNTCONN, a jeśli żaden nie jest dostępny, używa zgrupowanego CLNTCONN menedżera kolejek.

V 9.3.0 V 9.3.0 W produkcie IBM MQ 9.3.0klient .NET zachowuje się tak samo jak klienty C i Java i zwraca błąd MQRC_Q_MGR_NAME_ERROR w przypadku używania tabeli CCDT z grupowaniem menedżera kolejek.

## W jaki sposób aplikacja .NET określa, która definicja kanału ma być używana

W środowisku klienta IBM MQ .NET definicję kanału, która ma być używana, można określić na wiele różnych sposobów. Może istnieć wiele specyfikacji definicji kanału. Aplikacja pobiera definicję kanału z jednego lub większej liczby źródeł.

Jeśli istnieje więcej niż jedna definicja kanału, używana definicja kanału jest wybierana w następującej kolejności priorytetów:

1. Właściwości określone w konstruktorze MQQueueManager , jawnie lub przez dołączenie programu *MQC.CHANNEL_PROPERTY* w tabeli mieszającej właściwości
2. Właściwość *MQC.CHANNEL_PROPERTY* w tabeli mieszającej MQEnvironment.properties
3. Właściwość *Kanał* w środowisku MQEnvironment
4. Plik konfiguracyjny aplikacji .NET , nazwa sekcji CHANNELS, klucz ServerConnectionParms (dotyczy tylko połączeń zarządzanych)
5. Zmienna środowiskowa *MQSERVER*
6. Plik konfiguracyjny klienta, sekcja CHANNELS, atrybut ServerConnection
7. Tabela definicji kanału klienta (CCDT). Położenie tabeli definicji kanału klienta jest określone w pliku konfiguracyjnym aplikacji .NET (dotyczy tylko połączeń zarządzanych).
8. Tabela definicji kanału klienta (CCDT). Położenie tabeli CCDT jest określane przy użyciu zmiennych środowiskowych *MQCHLIB* i *MQCHLTAB* .
9. Tabela definicji kanału klienta (CCDT). Położenie tabeli definicji kanału klienta jest określane przy użyciu pliku konfiguracyjnego klienta.

W przypadku pozycji 1-3, pole definicji kanału jest budowane według pola na podstawie wartości podanych przez aplikację. Wartości te można podać przy użyciu różnych interfejsów i dla każdego z nich może istnieć wiele wartości. Wartości pól są dodawane do definicji kanału zgodnie z podaną kolejnością priorytetów:

1. Wartość parametru *connName* w konstruktorze MQQueueManager .
2. Wartości właściwości z tabeli mieszającej MQQueueManager.properties
3. Wartości właściwości z tabeli mieszającej MQEnvironment.properties
4. Wartości ustawione jako pola MQEnvironment (na przykład MQEnvironment.Hostname, MQEnvironment.Port)

W przypadku pozycji 4-6 jako wartość podawana jest cała definicja kanału. Nieokreślone pola w definicji kanału przyjmują systemowe wartości domyślne. Z tymi specyfikacjami nie są łączone żadne wartości z innych metod definiowania kanałów i ich pól.

W przypadku pozycji 7-9 cała definicja kanału jest pobierana z tabeli definicji kanału (CCDT). Pola, które nie zostały jawnie określone podczas definiowania kanału, przyjmują systemowe wartości domyślne. Z tymi specyfikacjami nie są łączone żadne wartości z innych metod definiowania kanałów i ich pól.

## Korzystanie z wyjść kanałów w programie IBM MQ .NET

Jeśli używane są powiązania klienta, można użyć wyjść kanałów, tak jak w przypadku każdego innego połączenia klienta. Jeśli używane są powiązania zarządzane, należy napisać program obsługi wyjścia, który implementuje odpowiedni interfejs.

### Powiązania klienta

Jeśli używane są powiązania klienta, można użyć wyjść kanałów zgodnie z opisem w sekcji Wyjścia kanałów. Nie można używać wyjść kanału napisanych dla powiązań zarządzanych.

### Powiązania zarządzane

Jeśli używane jest połączenie zarządzane, w celu zaimplementowania wyjścia należy zdefiniować nową klasę .NET , która implementuje odpowiedni interfejs. W pakiecie IBM MQ zdefiniowano trzy interfejsy wyjścia:

- MQSendExit
- MQReceiveExit
- MQSecurityExit

**Uwaga:** Programy zewnętrzne napisane za pomocą tych interfejsów nie są obsługiwane jako programy zewnętrzne kanału w środowisku niezarządzanym.

W poniższym przykładzie zdefiniowano klasę, która implementuje wszystkie trzy elementy:

```
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
  }

}
```

Do każdego wyjścia jest przekazywana instancja obiektu MQChannelExit i MQChannelDefinition . Te obiekty reprezentują struktury MQCXP i MQCD zdefiniowane w interfejsie proceduralnym.

Dane, które mają zostać wysłane przez wyjście wysyłania, oraz dane odebrane w wyjściu zabezpieczeń lub wyjścia odbierania, są określane przy użyciu parametrów wyjścia.

W pozycji dane na pozycji *dataOffset* o długości *dataLength* w tablicy bajtów *dataBuffer* są danymi, które mają zostać wysłane przez wyjście wysyłania, a dane odebrane przez wyjście zabezpieczeń lub wyjście odbierania. Parametr *dataMaxLength* określa maksymalną długość (od *dataOffset* ). dostępne dla wyjścia w *dataBuffer*. Uwaga: W przypadku wyjścia zabezpieczeń możliwe jest, aby parametr dataBuffer miał wartość NULL, jeśli wyjście jest wywoływane po raz pierwszy lub jeśli partnerski koniec nie wysyła żadnych danych.

Po zwróceniu wartości *dataOffset* i *dataLength* powinny być ustawione tak, aby wskazywały na przesunięcie i długość zwróconej tablicy bajtów, której powinny używać klasy .NET . W przypadku wyjścia wysyłania wskazuje on dane, które powinny zostać wysłane, a w przypadku wyjścia zabezpieczeń lub wyjścia odbierania dane, które powinny zostać zinterpretowane. Wyjście powinno zwykle zwracać tablicę bajtów; wyjątki są wyjściem zabezpieczeń, które może wybrać, aby nie wysyłać żadnych danych, i każde wyjście wywołane z powodów INIT lub TERM. Najprostszą formą wyjścia, jaką można zapisać, jest wyjście, które nie zwraca więcej niż dataBuffer:

Najprostszą możliwą treścią wyjścia jest:

```
{
  return dataBuffer;
}
```

## Klasa MQChannelDefinition

Identyfikator użytkownika i hasło określone w zarządzanej aplikacji klienckiej .NET są ustawiane w klasie MQChannelDefinition produktu IBM MQ .NET , która jest przekazywana do wyjścia zabezpieczeń klienta. Wyjście zabezpieczeń kopiuje identyfikator użytkownika i hasło do produktu MQCD.RemoteUserIdentifier i MQCD.RemotePassword (patrz “Zapisywanie wyjścia zabezpieczeń” na stronie 1003).

### *Określanie wyjść kanału (klient zarządzany)*

Jeśli podczas tworzenia obiektu MQQueueManager (w środowisku MQEnvironment lub w konstruktorze MQQueueManager ) zostanie określona nazwa kanału i nazwa połączenia, można określić wyjścia kanału na dwa sposoby.

W kolejności pierwszeństwa są to:

1. Przekazywanie właściwości tabeli mieszającej MQC.SECURITY_EXIT_PROPERTY, MQC.SEND_EXIT_PROPERTY lub MQC.RECEIVE_EXIT_PROPERTY w konstruktorze MQQueueManager .
2. Ustawianie właściwości SecurityExit, SendExit lub ReceiveExit środowiska MQEnvironment.

Jeśli nazwa kanału i nazwa połączenia nie zostaną określone, używane wyjścia kanału będą pochodzić z definicji kanału pobieranej z tabeli definicji kanału klienta (CCDT). Nie można przesłonić wartości zapisanych w definicji kanału. Więcej informacji na temat tabel definicji kanału zawiera sekcja Tabela definicji kanału klienta i sekcja “Używanie tabeli definicji kanału klienta z produktem .NET” na stronie 610 .

W każdym przypadku specyfikacja ma postać łańcucha o następującym formacie:

```
Assembly_name(Class_name)
```

*nazwa_klasy* jest pełną nazwą, w tym specyfikacją przestrzeni nazw, klasy .NET , która implementuje IBM IBM.WMQ.MQSecurityExit, IBM.WMQ.MQSendExit lub IBM.WMQ.MQReceiveExit (odpowiednio). *nazwa_zespołu* to pełne położenie, w tym rozszerzenie nazwy pliku, zespołu, w którym znajduje się klasa. Długość łańcucha jest ograniczona do 999 znaków, jeśli używane są właściwości MQEnvironment lub MQQueueManager. Jeśli jednak nazwa wyjścia kanału jest określona w tabeli definicji kanału (CCDT), jej długość jest ograniczona do 128 znaków. W razie potrzeby kod klienta .NET ładuje i tworzy instancję określonej klasy, analizując specyfikację łańcucha.

### *Określanie danych użytkownika wyjścia kanału (klient zarządzany)*

Z wyjściami kanału mogą być powiązane dane użytkownika. Jeśli podczas tworzenia obiektu MQQueueManager (w środowisku MQEnvironment lub w konstruktorze MQQueueManager ) zostanie podana nazwa kanału i nazwa połączenia, dane użytkownika można określić na dwa sposoby.

W kolejności pierwszeństwa są to:

1. Przekazywanie właściwości tabeli mieszającej MQC.SECURITY_USERDATA_PROPERTY, MQC.SEND_USERDATA_PROPERTY lub MQC.RECEIVE_USERDATA_PROPERTY w konstruktorze MQQueueManager .
2. Ustawianie właściwości danych SecurityUser, SendUserlub ReceiveUserśrodowiska MQEnvironment.

Jeśli nazwa kanału i nazwa połączenia nie zostaną określone, używane wartości danych użytkownika wyjścia będą pochodzić z definicji kanału pobieranej z tabeli definicji kanału klienta (CCDT). Nie można przesłonić wartości zapisanych w definicji kanału. Więcej informacji na temat tabel definicji kanału zawiera sekcja Tabela definicji kanału klienta i sekcja “Używanie tabeli definicji kanału klienta z produktem .NET” na stronie 610 .

W każdym przypadku specyfikacja jest łańcuchem, którego długość jest ograniczona do 32 znaków.

## Automatyczne ponowne nawiązywanie połączenia z klientem w produkcie .NET

Istnieje możliwość automatycznego ponownego nawiązania połączenia klienta z menedżerem kolejek podczas nieoczekiwanego przerwania połączenia.

Klient może nieoczekiwanie zostać odłączony od menedżera kolejek, jeśli na przykład menedżer kolejek zostanie zatrzymany lub nastąpi awaria sieci lub serwera.

Bez automatycznego ponownego nawiązywania połączenia przez klienta w przypadku niepowodzenia połączenia generowany jest błąd. Kod błędu może być pomocna przy ponownym nawiązywaniu połączenia.

Klient, który korzysta z narzędzia do automatycznego ponownego nawiązywania połączeń, jest nazywany klientem z możliwością ponownego połączenia. Aby utworzyć klienta z możliwością ponownego połączenia, należy określić pewne opcje nazywane opcjami ponownego połączenia podczas nawiązywania połączenia z menedżerem kolejek.

Jeśli aplikacja kliencka jest klientem programu IBM MQ .NET , może zdecydować się na automatyczne ponowne nawiązywanie połączenia z klientem, podając odpowiednią wartość właściwości CONNECT_OPTIONS_PROPERTY podczas tworzenia menedżera kolejek za pomocą klasy `MQQueueManager` . Szczegółowe informacje na temat wartości CONNECT_OPTIONS_PROPERTY zawiera sekcja Opcje ponownego połączenia .

Można wybrać, czy aplikacja kliencka ma zawsze nawiązywać połączenie i wznawiać połączenie z menedżerem kolejek o tej samej nazwie, z tym samym menedżerem kolejek, czy z dowolnym zestawem menedżerów kolejek zdefiniowanych za pomocą tej samej wartości `QMNAME` w tabeli połączeń klienta (szczegółowe informacje można znaleźć w sekcji Grupy menedżerów kolejek w tabeli CCDT ).

## Obsługa protokołu TLS (Transport Layer Security) w produkcie .NET

Aplikacje klienckie IBM MQ classes for .NET obsługują szyfrowanie TLS (Transport Layer Security). Protokół TLS zapewnia bezpieczeństwo komunikacji przez Internet i umożliwia aplikacjom typu klient/serwer komunikację w sposób poufny i niezawodny.

**Pojęcia pokrewne**

Obsługa protokołu TLS zarządzanego klienta IBM MQ.NET

Szyfrujące protokoły bezpieczeństwa: TLS

## *Obsługa protokołu TLS dla niezarządzanego klienta .NET*

Obsługa protokołu TLS dla niezarządzanego klienta .NET jest oparta na MQI języka C i IBM Global Security Kit (GSKit). Interfejs MQI języka C obsługuje operacje TLS i GSKit implementuje protokoły TLS Secure Socket.

*Włączanie protokołu TLS dla niezarządzanego klienta .NET*

Protokół TLS jest obsługiwany tylko dla połączeń klienckich. Aby włączyć obsługę protokołu TLS, należy określić parametr CipherSpec , który ma być używany podczas komunikacji z menedżerem kolejek. Musi on być zgodny z wartością CipherSpec ustawioną w kanale docelowym.

Aby włączyć obsługę protokołu TLS, należy określić parametr CipherSpec za pomocą zmiennej składowej static SSLCipherSpec środowiska MQEnvironment. Poniższy przykład przyłącza się do kanału SVRCONN o nazwie SECURE.SVRCONN.CHANNEL, który został skonfigurowany do obsługi protokołu TLS z wartością CipherSpec ustawioną na TLS_RSA_WITH_AES_128_CBC_SHA:

V 9.3.0 V 9.3.0

```
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");
```

Listę CipherSpecszawiera sekcja Określanie specyfikacji szyfrowania CipherSpecs .

Właściwość SSLCipherSpec można również ustawić przy użyciu właściwości MQC.SSL_CIPHER_SPEC_PROPERTY w tabeli mieszającej właściwości połączenia.

Aby pomyślnie nawiązać połączenie przy użyciu protokołu TLS, magazyn kluczy klienta musi być skonfigurowany z głównym łańcuchem certyfikatów ośrodka certyfikacji, z którego można uwierzytelnić certyfikat prezentowany przez menedżer kolejek. Podobnie, jeśli opcja SSLClientAuth w kanale SVRCONN ma wartość MQSSL_CLIENT_AUTH_REQUIRED, magazyn kluczy klienta musi zawierać identyfikujący certyfikat osobisty, który jest zaufany przez menedżer kolejek.

*Używanie nazwy wyróżniającej menedżera kolejek*

Menedżer kolejek identyfikuje się za pomocą certyfikatu TLS, który zawiera *nazwę wyróżniającą* (DN).

Aplikacja kliencka IBM MQ .NET może użyć tej nazwy wyróżniającej, aby upewnić się, że komunikuje się z poprawnym menedżerem kolejek. Wzorzec nazwy wyróżniającej jest określany za pomocą zmiennej nazwy sslPeerśrodowiska MQEnvironment. Na przykład ustawienie:

```
MQEnvironment.SSLPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

umożliwia pomyślne nawiązanie połączenia tylko wtedy, gdy menedżer kolejek przedstawi certyfikat o nazwie zwykłej rozpoczynającej się od łańcucha QMGR., i co najmniej dwie nazwy jednostek organizacyjnych, z których pierwsza musi być IBM , a druga WEBSPHERE.

Właściwość SSLPeerName można również ustawić przy użyciu właściwości MQC.SSL_PEER_NAME_PROPERTY w tabeli mieszającej właściwości połączenia. Więcej informacji na temat nazw wyróżniających i reguł ustawiania nazw węzłów sieci zawiera sekcja Zabezpieczanie produktu IBM MQ.

Jeśli parametr SSLPeerName jest ustawiony, połączenia powiodą się tylko wtedy, gdy zostanie ustawiony poprawny wzorzec, a menedżer kolejek przedstawi zgodny certyfikat.

*Obsługa błędów podczas używania protokołu TLS*

Program IBM MQ classes for .NET może użyć następujących kodów przyczyny podczas nawiązywania połączenia z menedżerem kolejek przy użyciu protokołu TLS:

**MQRC_SSL_NOT_ALLOWED (NIEWŁĄCZONY)**
Właściwość SSLCipherSpec została ustawiona, ale używane było połączenie powiązań. Tylko połączenie klienta obsługuje protokół TLS.

**MQRC_SSL_PEER_NAME_MISMATCH**
Wzorzec nazwy wyróżniającej podany we właściwości SSLPeerName nie jest zgodny z nazwą wyróżniającą podaną przez menedżer kolejek.

**MQRC_SSL_PEER_NAME_ERROR BŁĄD**
Wzorzec nazwy wyróżniającej podany we właściwości SSLPeerName jest niepoprawny.

V 9.3.0 V 9.3.0 **BŁĄD MQRC_KEY_REPOSITORY_ERROR**
Położenie repozytorium kluczy nie zostało określone, jest niepoprawne lub nie można uzyskać do niego dostępu.

## *Obsługa protokołu TLS dla zarządzanego klienta .NET*

Zarządzany klient .NET używa bibliotek produktu Microsoft .NET Framework do zaimplementowania protokołów SSL TLS. Klasa Microsoft System.Net.SecuritySslStream działa jako strumień przez połączone gniazda TCP oraz wysyła i odbiera dane za pośrednictwem tego połączenia gniazda.

Minimalny wymagany poziom .NET Framework to .NET Framework v3.5. Poziom obsługi algorytmu szyfrowania jest określany na podstawie poziomu .NET Framework używanego przez aplikację:

- W przypadku aplikacji opartych na .NET Framework poziomach 3.5 i 4.0dostępne są protokoły SSL 3.0 i TSL 1.0.
- W przypadku aplikacji, które są oparte na .NET Framework poziomie 4.5, dostępne są protokoły SSL 3.0, TLS 1.1 i TLS 1.2.

Może być konieczne przeniesienie aplikacji, które oczekują wyższej obsługi protokołu TLS, do nowszej wersji środowiska zgodnie z definicją obsługi zabezpieczeń Microsoft w pliku .NET Framework.

Główne funkcje obsługi protokołu TLS dla zarządzanego klienta .NET są następujące:

**Obsługa protokołu TLS**
Obsługa protokołu TLS dla klienta zarządzanego przez .NET jest definiowana za pomocą klasy .NET SSLStream i zależy od środowiska .NET używanego przez aplikację. Więcej informacji na ten temat zawiera sekcja “Obsługa protokołu TLS dla zarządzanego klienta .NET” na stronie 618.

**Obsługa CipherSpec**
Ustawienia TLS dla klienta zarządzanego przez .NET są takie same, jak dla połączeń TLS produktu Microsoft.NET . Więcej informacji na ten temat zawierają sekcje “Obsługa CipherSpec dla zarządzanego klienta .NET” na stronie 618 oraz “Odwzorowania CipherSpec dla zarządzanego klienta .NET” na stronie 620.

**Repozytoria kluczy**
Repozytorium kluczy po stronie klienta jest magazynem kluczy Windows . Repozytorium po stronie serwera jest repozytorium typu Cryptographic Message Syntax (CMS). Więcej informacji na ten temat zawiera sekcja “Repozytoria kluczy dla zarządzanego klienta .NET” na stronie 622.

**Certyfikaty**
Do zaimplementowania wzajemnego uwierzytelniania między klientem a menedżerem kolejek można użyć samopodpisanych certyfikatów TLS. Więcej informacji na ten temat zawiera sekcja “Korzystanie z certyfikatów dla zarządzanego klienta .NET” na stronie 622.

**SSLPEERNAME**
W systemie .NETaplikacje mogą używać opcjonalnego atrybutu SSLPEERNAME do określenia wzorca nazwy wyróżniającej (DN). Więcej informacji na ten temat zawiera sekcja “SSLPEERNAME” na stronie 622.

**Zgodność ze standardami FIPS**
Programowe włączenie trybu FIPS nie jest obsługiwane przez bibliotekę zabezpieczeń produktu Microsoft.NET . Włączenie standardu FIPS jest kontrolowane przez ustawienie Windows Group Policy.

**Zgodność z NSA Suite B**
IBM MQ implementuje RFC 6460. Implementacja Microsoft.NET dla pakietu NSA Suite B to 5430. Jest to obsługiwane od wersji 3.5 środowiska .NET Framework.

**Resetowanie lub renegocjowanie klucza tajnego**
Chociaż klasa SSLStream nie obsługuje resetowania ani renegocjacji klucza tajnego, w celu zachowania spójności z innymi klientami IBM MQ zarządzany klient .NET zezwala aplikacjom na ustawianie wartości `SSLKeyResetCount`. Więcej informacji na ten temat zawiera sekcja “Resetowanie lub renegocjowanie klucza tajnego dla zarządzanego klienta .NET” na stronie 623.

**Sprawdzenie odwołania**
Klasa SSLStream obsługuje sprawdzanie odwołań certyfikatów, które jest automatycznie wykonywane przez mechanizm łączenia certyfikatów w łańcuch. Więcej informacji na ten temat zawiera sekcja “Sprawdzenie odwołania” na stronie 623.

**Obsługa wyjścia zabezpieczeń systemu IBM MQ**
Klasa `SSLStream` zapewnia ograniczoną obsługę wyjść zabezpieczeń IBM MQ . Możliwe jest wysyłanie zapytań do certyfikatów lokalnych i zdalnych w celu uzyskania `SSLPeerNamePtr`(Nazwa wyróżniająca podmiotu) i `SSLRemCertIssNamePtr` (Nazwa wyróżniająca wystawcy), ponieważ jest to obsługiwane w produkcie Microsoft.NET. Jednak nie jest obsługiwane pobieranie atrybutów, takich jak DNQ, UNSTRUCTUREDNAME i UNSTRUCTUREDADDRESS, dlatego tych wartości nie można pobrać przy użyciu wyjść.

**Obsługa sprzętu szyfrującego**
Sprzęt szyfrujący nie jest obsługiwany przez zarządzanego klienta .NET .

## Obsługa protokołu TLS1.3 w zarządzanych klientach IBM MQ .NET i XMS .NET

V 9.3.2

W systemie IBM MQ 9.3.2klienty IBM MQ .NET i XMS .NET obsługują protokół TLS1.3 , pod warunkiem, że system operacyjny obsługuje protokół TLS1.3.

Zarządzany klient .NET używa bibliotek produktu Microsoft .NET Framework do zaimplementowania protokołów SSL TLS. Klasa Microsoft `System.Net.SecuritySslStream` działa jako strumień przez połączone gniazda TCP oraz wysyła i odbiera dane za pośrednictwem tego połączenia gniazda.

W systemie Windowsprodukt .NET używa parametru SCHANNEL, a w systemie Linux .NET używa parametru OpenSSL do komunikacji SSL.

### Windows Dla aplikacji klienckich IBM MQ .NET działających w systemie Windows

Microsoft ogłosił, że Windows 11 i Windows Server 2022 domyślnie obsługują szyfry TSL1.3 .

Zestawy algorytmów szyfrowania TLS_AES_128_GCM_SHA256 i TLS_AES_256_GCM_SHA384 są domyślnie włączone w obu wersjach systemu Windows.

**Ostrzeżenie:**

- TLS_CHACHA20_POLY1305_SHA256 Zestaw algorytmów szyfrowania nie jest domyślnie włączony, ale jest obsługiwany.
- W przypadku klienta IBM MQ .NET z włączonym TLS1.3 , aby pomyślnie nawiązać połączenie z menedżerem kolejek, IBM Global Security Kit (GSKit) wersja 8.0.55.29 jest minimalną wersją, która jest wymagana po stronie menedżera kolejek.

### Linux Dla aplikacji klienckich IBM MQ .NET działających w systemie Linux

Ponieważ system .NET używa protokołu OpenSSL w systemie Linux do komunikacji SSL, aby używać protokołu TLS1.3, minimalnym wymaganiem jest OpenSSL v1.1.1 .

Ponadto, ponieważ .NET używa OpenSSL w systemie Linux, wszystkie szyfry obsługiwane przez OpenSSL powinny działać również w systemie .NET .

OpenSSL obsługuje następujące CipherSpecs dla protokołu 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

**Pojęcia pokrewne**

“Odwzorowania CipherSpec dla zarządzanego klienta .NET” na stronie 620
Interfejs programu IBM MQ.NET obsługuje tabelę odwzorowań typu IBM MQ na Microsoft.NET , która jest używana do określenia wersji protokołu TLS, której klient zarządzany potrzebuje do nawiązania bezpiecznego połączenia z menedżerem kolejek.

*Obsługa protokołu TLS dla zarządzanego klienta .NET*
Obsługa protokołu TLS w produkcie IBM MQ.NET jest oparta na klasie .NET SSLStream.

**Uwaga:** Obsługa protokołu TLS dla zarządzanego klienta .NET zależy od poziomu środowiska .NET , z którego korzysta aplikacja. Więcej informacji na ten temat zawiera sekcja “Obsługa protokołu TLS dla zarządzanego klienta .NET” na stronie 616.

Dla klasy SSLStream w systemie Microsoft.NET , aby zainicjować protokół TLS i przeprowadzić uzgadnianie z menedżerem kolejek, jednym z wymaganych parametrów, które należy ustawić, jest **SSLProtocol**, gdzie należy podać numer wersji protokołu TLS, który musi mieć jedną z następujących wartości:

- SSL3.0
- TLS1.0
- TLS1.2

Wartość tego parametru jest ściśle powiązana z rodziną protokołów, do której należy preferowana specyfikacja CipherSpec . Gdy strumień SSL rozpoczyna uzgadnianie TLS z serwerem (menedżerem kolejek), używa wersji protokołu TLS określonej w parametrze **SSLProtocol** do identyfikowania listy CipherSpecs , które mają być używane do negocjacji.

Produkt IBM MQ.NET nie udostępnia żadnych właściwości, które mogą być używane przez aplikacje do ustawiania tej wartości. Zamiast tego produkt IBM MQ używa tabeli odwzorowań w celu wewnętrznego odwzorowania CipherSpec ustawionej na rodzinę protokołów i identyfikuje wersję protokołu SSLProtocol, która ma być używana. W tej tabeli przedstawiono odwzorowanie każdej obsługiwanej CipherSpec między produktem Microsoft.NET i produktem IBM MQoraz wersję protokołu, do której one należą. Więcej informacji na ten temat zawiera sekcja “Odwzorowania CipherSpec dla zarządzanego klienta .NET” na stronie 620.

*Obsługa CipherSpec dla zarządzanego klienta .NET*
Ustawienia CipherSpec dla aplikacji są używane podczas uzgadniania z serwerem.

Klienty IBM MQ umożliwiają ustawienie wartości CipherSpec , która jest używana podczas uzgadniania z menedżerem kolejek. Klienty IBM MQ powinny ustawić poprawną wartość atrybutu CipherSpec , aby nawiązać bezpieczne połączenie, najlepiej wartość CipherSpec określoną w strategii grupy Windows . Pozostawienie tego pola pustego oznacza kanał jawnego tekstu bez ochrony gniazd.

W przypadku klienta zarządzanego przez IBM MQ.NET ustawienia TLS dotyczą klasy SSLStream w systemie Microsoft.NET . W przypadku strumienia SSL parametr CipherSpeclub listy preferencji CipherSpecsmożna ustawić tylko w strategii grupy Windows , która jest ustawieniem dla całego komputera. Następnie strumień SSL używa określonej CipherSpec lub listy preferencji podczas uzgadniania z serwerem. W przypadku innych klientów IBM MQ właściwość CipherSpec można ustawić w aplikacji w definicji kanału produktu IBM MQ . To samo ustawienie jest używane na potrzeby negocjacji TLS. Z powodu tego ograniczenia uzgadnianie TLS może negocjować dowolną obsługiwaną CipherSpec niezależnie od tego, co jest określone w konfiguracji kanału produktu IBM MQ . Z tego powodu prawdopodobnie spowoduje to wystąpienie błędu AMQ9631 w menedżerze kolejek. Aby uniknąć tego błędu, należy ustawić tę samą wartość CipherSpec , która została ustawiona w aplikacji, co konfiguracja TLS w strategii grupy Windows .

Nowy kod klienta TLS produktu IBM MQ.NET sprawdza tylko, czy wynegocjowana została poprawna wersja protokołu. Wersja protokołu TLS jest określana na podstawie CipherSpec ustawionej przez

aplikację i używanej do uzgadniania TLS z serwerem (menedżerem kolejek). Dlatego jest ona wymagana z projektu do ustawienia parametru CipherSpec w aplikacji klienta zarządzanego przez produkt IBM MQ.NET . Jeśli parametr CipherSpec ustawiony przez klienta IBM MQ jest inny niż protokół SSL 3.0, TLS 1.0 i TLS 1.2 , IBM MQ zarządzany .NET klient będzie domyślnie negocjować z dowolnym szyfram z protokołu SSL 3.0 lub TLS 1.0 i nie zgłosi błędu.

**Uwaga:** Jeśli wartość CipherSpec podana przez aplikację nie jest wartością CipherSpec znaną produktowi IBM MQ, IBM MQ zarządzany .NET klient zlekceważy ją i negocjuje połączenie w oparciu o strategię grupy systemu Windows .

## Ustawianie atrybutu CipherSpec

Istnieją trzy sposoby ustawiania parametru CipherSpec:

**Klasa .NET MQEnvironment**
W poniższym przykładzie przedstawiono sposób ustawienia atrybutu CipherSpec z klasą MQEnvironment.

```
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";
```

**Właściwość CipherSpec protokołu TLS**
W poniższym przykładzie przedstawiono sposób ustawienia atrybutu CipherSpec przez dodanie parametru hashtable do konstruktora MQQueueManager .

```
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 zasady grupy**
Jeśli lista zestawów algorytmów szyfrowania jest konfigurowana za pomocą konsoli zarządzania strategii grupy Windows , definicja kanału SVRCONN musi określać zgodną CipherSpec. Zgodna CipherSpec może być wartością ogólną, taką jak "ANY_TLS12_OR_HIGHER", lub konkretną wartością, która jest odwzorowywana na najwyższy zestaw algorytmów szyfrowania negocjowany z listy uporządkowanej. Użycie ogólnych wartości CipherSpec jest zalecane w przypadku klientów .NET, ponieważ pozwala uniknąć konieczności zmiany konfiguracji SVRCONN CipherSpec w przypadku zmiany kolejności na liście klientów.

## Składnia CCDT

Produkt IBM MQ.NET obsługuje tylko tabele definicji kanału klienta (pliki .TAB) znajdujące się na komputerze lokalnym. Istniejące pliki CCDT, które mają ustawioną wartość CipherSpec , mogą być używane na potrzeby połączeń produktu IBM MQ.NET . Jednak wartość CipherSpec ustawiona w kanale połączenia klienckiego określa wersję protokołu TLS i musi być zgodna z wartością CipherSpec ustawioną w strategii grupy Windows .

**Pojęcia pokrewne**
“Konfigurowanie środowiska IBM MQ” na stronie 605
Przed użyciem połączenia klienta do nawiązania połączenia z menedżerem kolejek należy skonfigurować środowisko produktu IBM MQ .

“Obsługa protokołu TLS dla zarządzanego klienta .NET” na stronie 616

Zarządzany klient .NET używa bibliotek produktu Microsoft .NET Framework do zaimplementowania protokołów SSL TLS. Klasa Microsoft System.Net.SecuritySslStream działa jako strumień przez połączone gniazda TCP oraz wysyła i odbiera dane za pośrednictwem tego połączenia gniazda.

**Zadania pokrewne**

Określanie CipherSpecs

**Odsyłacze pokrewne**

Klasa .NET MQEnvironment

*Odwzorowania CipherSpec dla zarządzanego klienta .NET*

Interfejs programu IBM MQ.NET obsługuje tabelę odwzorowań typu IBM MQ na Microsoft.NET , która jest używana do określenia wersji protokołu TLS, której klient zarządzany potrzebuje do nawiązania bezpiecznego połączenia z menedżerem kolejek.

Jeśli w kanale SVRCONN określono wartość CipherSpec , po zakończeniu uzgadniania TLS menedżer kolejek próbuje uzgodnić wartość CipherSpec z wynegocjowaną wartością CipherSpec , która jest używana przez aplikację kliencką. Jeśli menedżer kolejek nie może znaleźć zgodnej specyfikacji szyfrowania CipherSpec, komunikacja kończy się niepowodzeniem z błędem AMQ9631.

Interfejs produktu IBM MQ.NET obsługuje tabelę odwzorowań CipherSpec IBM MQ na produkt Microsoft.NET . Ta tabela służy do określenia wersji protokołu TLS, której klient chce użyć do nawiązania bezpiecznego połączenia przez gniazdo z menedżerem kolejek. W zależności od wartości SSLCipherSpec wersją protokołu SSLProtocol może być TLS 1.0lub TLS 1.2(w zależności od używanej wersji środowiska Microsoft.NET ).

Upewnij się, że podano poprawną wartość SSLCipherSpec , ponieważ podanie niepoprawnej wartości może spowodować użycie protokołów SSL 3.0 lub TLS 1.0 .

*Tabela 78. Tabela odwzorowań produktów IBM MQ i Microsoft.NET*

| **IBM MQ CipherSpec** | **Microsoft.NET CipherSpec** | **Wersja TLS** |
|---|---|---|
| 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 |
| ECDHE_ECDSA_AES_128_CBC_SHA256 | TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P521 | TLS 1.2 |

| *Tabela 78. Tabela odwzorowań produktów IBM MQ i Microsoft.NET (kontynuacja)* | | |
|---|---|---|
| **IBM MQ CipherSpec** | **Microsoft.NET CipherSpec** | **Wersja TLS** |
| 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 |
| V 9.3.2 TLS_AES_256_GCM_SHA384 | TLS_AES_256_GCM_SHA384 | TLS 1.3 |
| V 9.3.2 TLS_CHACHA20_POLY1305_SHA256 | TLS_CHACHA20_POLY1305_SHA256 | TLS 1.3 |
| V 9.3.2 TLS_AES_128_GCM_SHA256 | TLS_AES_128_GCM_SHA256 | TLS 1.3 |
| V 9.3.2 TLS_AES_128_CCM_8_SHA256 | TLS_AES_128_CCM_8_SHA256 | TLS 1.3 |
| V 9.3.2 TLS_AES_128_CCM_SHA256 | TLS_AES_128_CCM_SHA256 | TLS 1.3 |

**Uwagi:**

1. Deprecated Ta CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA jest nieaktualna. Jednak nadal może być używany do przesyłania do 32 GB danych, zanim połączenie zostanie przerwane i zostanie zgłoszony błąd AMQ9288. Aby uniknąć tego błędu, należy unikać używania potrójnego algorytmu szyfrowania DES lub włączyć resetowanie klucza tajnego podczas korzystania z tej CipherSpec.

**Pojęcia pokrewne**

“Obsługa protokołu TLS dla zarządzanego klienta .NET” na stronie 616

Zarządzany klient .NET używa bibliotek produktu Microsoft .NET Framework do zaimplementowania protokołów SSL TLS. Klasa Microsoft System.Net.SecuritySslStream działa jako strumień przez połączone gniazda TCP oraz wysyła i odbiera dane za pośrednictwem tego połączenia gniazda.

*Repozytoria kluczy dla zarządzanego klienta .NET*
Repozytorium kluczy używane przez zarządzane klienty .NET to magazyn kluczy Windows . Certyfikaty i klucze prywatne muszą być dostępne w pliku kluczy użytkownika lub systemu, aby aplikacja kliencka mogła korzystać z nich zarówno w przypadku tożsamości, jak i zaufania podczas uzgadniania TLS.

## Po stronie klienta

W aplikacji można ustawić jedną z następujących wartości dla repozytorium kluczy:

- "*USER": produkt IBM MQ.NET uzyskuje dostęp do bazy certyfikatów bieżącego użytkownika w celu pobrania certyfikatów klienta.
- "*SYSTEM": IBM MQ.NET uzyskuje dostęp do konta komputera lokalnego w celu pobrania certyfikatów.

Certyfikaty klienta muszą być przechowywane w bazie Moje certyfikaty konta użytkownika lub komputera. Wszystkie certyfikaty serwera (CA) muszą być przechowywane w katalogu głównym bazy certyfikatów.

**Uwaga:** W jednym pliku można przechowywać więcej niż jeden certyfikat w następujących formatach:

- Wymiana informacji osobistych-PKCS #12 (.PFX, .P12).
- Cryptographic Message Syntax Standard-certyfikaty PKCS #7 (.P7B)
- Serializowana baza certyfikatów Microsoft (.SST)

*Korzystanie z certyfikatów dla zarządzanego klienta .NET*
W przypadku certyfikatów klienta klient IBM MQ zarządzany .NET uzyskuje dostęp do magazynu kluczy Windows i ładuje wszystkie certyfikaty klienta, które są zgodne z etykietą certyfikatu lub są zgodne z łańcuchem.

Podczas wybierania certyfikatu, który ma być używany, IBM MQ zarządzany .NET klient zawsze używa pierwszego zgodnego certyfikatu dla uzgadniania SSLStream TLS.

## Uzgadnianie certyfikatów według etykiety certyfikatu

Jeśli zostanie ustawiona etykieta certyfikatu, klient IBM MQ zarządzany .NET przeszukuje bazę certyfikatów Windows o podanej nazwie, aby zidentyfikować certyfikat klienta. Ładuje wszystkie zgodne certyfikaty i używa pierwszego certyfikatu z listy. Istnieją dwie opcje ustawiania etykiety certyfikatu:

- Etykietę certyfikatu można ustawić w klasie MQEnvironment uzyskującej dostęp do MQEnvironment.CertificateLabel.
- Etykietę certyfikatu można również ustawić we właściwościach tabeli mieszającej, podając ją jako parametr wejściowy w konstruktorze MQQueueManager , jak pokazano w poniższym przykładzie.

```
Hashtable properties = new Hashtable();
properties.Add("CertificateLabel", "mycert");
```

Nazwa ("CertificateLabel") a w wartości rozróżniana jest wielkość liter.

## Uzgadnianie certyfikatów według łańcucha

Jeśli etykieta certyfikatu nie jest ustawiona, wyszukiwany i używany jest certyfikat zgodny z łańcuchem "ibmwebspheremq" i aktualnie zalogowanym użytkownikiem (małymi literami).

**Zadania pokrewne**

Bezpieczne łączenie klienta z menedżerem kolejek

**Odsyłacze pokrewne**

Klasa .NET MQEnvironment

*SSLPEERNAME*
Atrybut SSLPEERNAME jest używany do sprawdzania nazwy wyróżniającej (DN) certyfikatu z menedżera kolejek węzła sieci.

W produkcie IBM MQ.NETaplikacje mogą używać parametru SSLPEERNAME do określenia wzorca nazwy wyróżniającej, jak pokazano w poniższym przykładzie.

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

Podobnie jak w przypadku innych klientów IBM MQ , parametr SSLPEERNAME jest parametrem opcjonalnym.

Jeśli wartość SSLPEERNAME nie jest ustawiona, zarządzany klient IBM MQ.NET nie sprawdza poprawności certyfikatu zdalnego (serwera), a klient zarządzany po prostu akceptuje certyfikat zdalny (/serwer) w nieokreślonej postaci.

Sposób ustawienia parametru SSLPEERNAME zależy od tego, która z ofert stosu IBM MQ jest używana.

**IBM MQ classes for .NET**
Istnieją trzy następujące opcje.

1. Ustaw wartość MQEnvironment.SSLPeerName w klasie MQEnvironment.
2. `MQEnvironment.properties.Add(MQC.SSL_PEER_NAME_PROPERTY, value)`
3. Użyj konstruktora menedżera kolejek `MQQueueManager (String queueManagerName, Hashtable properties)`. Podaj parametr SSLPEERNAME w pliku `Hashtable properties` , tak jak w przypadku opcji 2.

**XMS .NET**
Ustaw nazwę węzła sieci SSL w fabryce połączeń:

```
ConnectionFactory.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, value);
```

**WCF,**
Dołącz do identyfikatora URI nazwę SslPeerjako pole rozdzielone średnikiem.

**Odsyłacze pokrewne**
Klasa .NET MQEnvironment

*Resetowanie lub renegocjowanie klucza tajnego dla zarządzanego klienta .NET*
Klasa SSLStream nie obsługuje resetowania/renegocjacji klucza tajnego. Jednak w celu zachowania spójności z innymi klientami IBM MQ IBM MQ zarządzany klient .NET umożliwia aplikacjom ustawianie wartości parametru **SSLKeyResetCount**.

Po osiągnięciu limitu program IBM MQ.NET rozłącza się z menedżerem kolejek, a aplikacja jest powiadamiana o tym jako o wyjątku z kodem przyczyny MQRC_CONNECTION_BROKEN. Aplikacje mogą obsłużyć wyjątek i ponownie nawiązać połączenia lub włączyć opcję MQCNO_RECONNECT, aby program IBM MQ.NET automatycznie ponownie nawiązywał połączenie z menedżerem kolejek.

Włączenie narzędzia do automatycznego ponownego nawiązywania połączeń oznacza, że po osiągnięciu licznika resetowania klucza wszystkie istniejące połączenia są wyłączane, a klient IBM MQ.NET ponownie tworzy wszystkie połączenia na nowo. Więcej informacji na temat automatycznego ponownego połączenia klienta zawiera sekcja Automatyczne ponowne połączenie klienta.

**Pojęcia pokrewne**
Resetowanie kluczy tajnych SSL i TLS

*Sprawdzenie odwołania*
Klasa SSLStream obsługuje sprawdzanie odwołań certyfikatów.

Sprawdzanie odwołań jest automatycznie wykonywane przez mechanizm łańcucha certyfikatów. Dotyczy to zarówno protokołu OCSP (Online Certificate Status Protocol), jak i list odwołań certyfikatów (Certificate Revocation List-CRL). Klasa SSLStream używa odwołania certyfikatu, które wykorzystuje tylko serwer określony w certyfikacie, czyli serwer jest narzucony przez sam certyfikat. Jest możliwe, że żądania HTTP CDP extensions i OCSP HTTP będą kierowane do serwera proxy za pośrednictwem serwera proxy HTTP .

Sposób ustawiania sprawdzania odwołań zależy od tego, która z ofert stosu IBM MQ jest używana.

**IBM MQ.NET**
Sprawdzenie odwołania można ustawić, uzyskując dostęp do właściwości **MQEnvironment.SSLCertRevocationCheck** w pliku klasy `MQEnvironment.cs` .

**XMS .NET**
Sprawdzanie odwołań można ustawić w kontekście właściwości fabryki połączeń, jak pokazano w poniższym przykładzie.

```
ConnectionFactory.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

**WCF,**
Sprawdzanie odwołań można ustawić dla identyfikatora URI przy użyciu następującej konwencji nazewnictwa.

```
"SslCertRevocationCheck=true"
```

*Konfigurowanie protokołu TLS dla zarządzanego serwera IBM MQ .NET*
Konfigurowanie protokołu TLS dla zarządzanego serwera IBM MQ .NET obejmuje utworzenie certyfikatów osoby podpisującej, a następnie skonfigurowanie po stronie serwera, po stronie klienta i aplikacji.

## O tym zadaniu

Aby skonfigurować protokół TLS, należy najpierw utworzyć odpowiednie certyfikaty osoby podpisującej. Certyfikaty osoby podpisującej mogą być samopodpisane lub mogą być udostępniane przez ośrodek certyfikacji. Chociaż certyfikaty samopodpisane mogą być używane w systemie programistycznym, testowym lub przedprodukcyjnym, nie należy ich używać w systemie produkcyjnym. W systemie produkcyjnym należy użyć certyfikatów uzyskanych z zaufanego zewnętrznego ośrodka certyfikacji (CA).

## Procedura

1. Utwórz certyfikaty osoby podpisującej.
   a) Aby utworzyć samopodpisane certyfikaty, należy użyć jednego z następujących narzędzi dostarczanych z produktem IBM MQ :

      Użyj interfejsu GUI programu **strmqikm** lub użyj komendy **runmqckm** lub **runmqakm** z poziomu wiersza komend. Więcej informacji na temat korzystania z tych narzędzi zawiera sekcja Używanie programów **runmqckm**, **runmqakm**i **strmqikm** do zarządzania certyfikatami cyfrowymi.
   b) Aby uzyskać certyfikaty dla menedżera kolejek i klientów z ośrodka certyfikacji (CA), należy postępować zgodnie z instrukcjami w sekcji Uzyskiwanie certyfikatów osobistych z ośrodka certyfikacji.
2. Skonfiguruj po stronie serwera.
   a) Skonfiguruj protokół TLS w menedżerze kolejek przy użyciu programu IBM Global Security Kit (GSKit)zgodnie z opisem w sekcji Nawiązywanie bezpiecznego połączenia klienta z menedżerem kolejek.
   b) Ustaw atrybuty TLS kanału SVRCONN:
      - Ustaw parametr **SSLCAUTH** na wartość `"REQUIRED/OPTIONAL"`.
      - Ustaw parametr **SSLCIPH** na odpowiednią wartość CipherSpec.

      Więcej informacji na ten temat zawiera sekcja“Włączanie protokołu TLS dla niezarządzanego klienta .NET” na stronie 615.
3. Skonfiguruj stronę klienta.
   a) Zaimportuj certyfikaty klienta do bazy certyfikatów Windows (w ramach konta użytkownika/komputera).

      Program IBM MQ .NET uzyskuje dostęp do certyfikatów klienta z bazy certyfikatów Windows , dlatego należy zaimportować certyfikaty do bazy certyfikatów Windows , aby nawiązać bezpieczne

połączenie przez gniazdo z serwerem IBM MQ . Więcej informacji na temat uzyskiwania dostępu do magazynu kluczy Windows i importowania certyfikatów po stronie klienta zawiera sekcja Importowanie lub eksportowanie certyfikatów i kluczy prywatnych.

b) Podaj etykietę CertificateLabel zgodnie z opisem w sekcji Bezpieczne łączenie klienta z menedżerem kolejek.

c) W razie potrzeby zmodyfikuj Windows Zasady grupy, aby ustawić CipherSpec, a następnie zrestartuj komputer, aby aktualizacje zasad grupy Windows zostały zastosowane.

4. Skonfiguruj aplikację.

a) Ustaw wartość MQEnvironment lub SSLCipherSpec , aby oznaczyć połączenie jako zabezpieczone.

Podana wartość jest używana do identyfikacji używanego protokołu (TLS). Zestaw CipherSpec powinien być jednym z CipherSpecs obsługiwanej wersji protokołu SSL i najlepiej może być taki sam, jak określony w strategii grupy Windows . Obsługiwana wersja protokołu SSLProtocol zależy od używanego środowiska .NET . Wersją protokołu SSL może być TLS 1.0lub TLS 1.2, w zależności od używanej wersji środowiska Microsoft .NET ).

**Uwaga:** Jeśli wartość CipherSpec podana przez aplikację nie jest wartością CipherSpec znaną produktowi IBM MQ, IBM MQ zarządzany .NET klient zlekceważy ją i negocjuje połączenie w oparciu o strategię grupy systemu Windows .

b) Ustaw właściwość SSLKeyRepository na wartość "*SYSTEM" lub "*USER".

c) Opcjonalne: Ustaw wartość SSLPEERNAME na nazwę wyróżniającą (DN) certyfikatu serwera.

d) Podaj etykietę CertificateLabel zgodnie z opisem w sekcji Bezpieczne łączenie klienta z menedżerem kolejek.

e) Ustaw dodatkowe parametry opcjonalne, które są wymagane, takie jak KeyResetCount, CertificationRevocationCheck i włącz FIPS.

#### Przykłady ustawiania protokołu TLS i repozytorium kluczy TLS

W przypadku bazy danych .NETmożna ustawić protokół TLS i repozytorium kluczy TLS za pomocą klasy MQEnvironment, jak pokazano w poniższym przykładzie:

```
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")
```

Alternatywnie można ustawić protokół TLS i repozytorium kluczy TLS, podając tabelę mieszającą jako część konstruktora MQQueueManager , jak pokazano w poniższym przykładzie.

```
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")
```

## Co dalej

Więcej informacji na temat rozpoczynania pracy z tworzeniem aplikacji TLS zarządzanych przez produkt IBM MQ .NET zawiera sekcja“Pisanie prostej aplikacji” na stronie 625.

**Odsyłacze pokrewne**

Klasa .NET MQEnvironment

Liczba operacji KeyReset(MQLONG)

Standardy FIPS (Federal Information Processing Standards) dla AIX, Linux, and Windows

*Pisanie prostej aplikacji*

Wskazówki dotyczące tworzenia prostej aplikacji TLS produktu .NET zarządzanej przez IBM MQ , w tym przykłady ustawiania właściwości SSL dla fabryk połączeń, tworzenia instancji menedżera kolejek, połączenia, sesji i miejsca docelowego oraz wysyłania komunikatu testowego.

## Zanim rozpoczniesz

Najpierw należy skonfigurować protokół TLS dla zarządzanego serwera IBM MQ.NET zgodnie z opisem w sekcji “Konfigurowanie protokołu TLS dla zarządzanego serwera IBM MQ .NET” na stronie 624.

W przypadku konfiguracji aplikacji w podstawowym produkcie .NETnależy ustawić właściwości protokołu SSL przy użyciu klasy MQEnvironment lub przez podanie tabeli mieszającej jako części konstruktora MQQueueManager .

W przypadku konfiguracji aplikacji w systemie XMS .NETnależy ustawić właściwości SSL w kontekście właściwości fabryk połączeń.

## Procedura

1. Ustaw właściwości SSL dla fabryk połączeń, jak pokazano w poniższych przykładach.

   **Przykład dla IBM MQ.NET**

   ```
   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;
   ```

   **Przykład dla XMS .NET**

   ```
   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. Utwórz instancję menedżera kolejek, połączenia, sesję i miejsce docelowe, jak pokazano w poniższych przykładach.

   **Przykład dla produktu MQ .NET**

   ```
   queueManager = new MQQueueManager(queueManagerName, properties);
   Console.WriteLine("done");

   // accessing queue
   Console.Write("Accessing queue " + queueName + ".. ");
   queue = queueManager.AccessQueue(queueName, MQC.MQOO_OUTPUT +
   MQC.MQOO_FAIL_IF_QUIESCING);
   Console.WriteLine("done");
   ```

   **Przykład dla XMS .NET**

   ```
   connectionWMQ = cf.CreateConnection();
   // Create session
   sessionWMQ = connectionWMQ.CreateSession(false, AcknowledgeMode.AutoAcknowledge);

   // Create destination
   destination = sessionWMQ.CreateQueue(destinationName);

   // Create producer
   producer = sessionWMQ.CreateProducer(destination);
   ```

3. Wyślij komunikat w sposób przedstawiony w poniższych przykładach.

**Przykład dla produktu MQ .NET**

```
// creating a message object
message = new MQMessage();
message.WriteString(messageString);

// putting messages continuously
for (int i = 1; i <= numberOfMsgs; i++)
{
Console.Write("Message " + i + " <" + messageString + ">.. ");
queue.Put(message);
Console.WriteLine("put");
}
```

**Przykład dla XMS .NET**

```
textMessage = sessionWMQ.CreateTextMessage();
textMessage.Text = simpleMessage;
producer.Send(textMessage);
```

4. Sprawdź połączenie TLS.

   Sprawdź status kanału, aby upewnić się, że połączenie TLS zostało nawiązane i działa poprawnie.

*Konfigurowanie śledzenia dla strumienia SSL*
Aby przechwytywać zdarzenia śledzenia i komunikaty dotyczące klasy SSLStream, należy dodać sekcję konfiguracji dla diagnostyki systemu do pliku konfiguracyjnego aplikacji.

## O tym zadaniu

**Uwaga:**

To zadanie dotyczy tylko systemu IBM MQ classes for .NET Framework . Plik konfiguracyjny aplikacji nie jest obsługiwany w IBM MQ classes for .NET (biblioteki.NET Standard i .NET 6 ).

Jeśli do pliku konfiguracyjnego aplikacji nie zostanie dodana sekcja konfiguracji na potrzeby diagnostyki systemu, IBM MQ zarządzany .NET klient nie będzie przechwytywać żadnych zdarzeń, śladów ani punktów debugowania związanych z protokołem TLS i klasą SSLStream.

**Uwaga:** Uruchomienie IBM MQ śledzenia za pomocą **strmqtrc** nie przechwytuje wszystkich wymaganych danych śledzenia TLS.

## Procedura

1. Utwórz plik konfiguracyjny aplikacji (App.Config) dla projektu aplikacji.
2. Dodaj sekcję konfiguracji diagnostyki systemu, jak pokazano w poniższym przykładzie.

```
<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>
```

**Ostrzeżenie:** Pole `Version` we wpisie `add name` musi mieć wartość odpowiadającą używanej wersji pliku `amqmdnet.dll` w serwisie .net.

**Zadania pokrewne**

Śledzenie klientów IBM MQ classes for .NET Framework przy użyciu pliku konfiguracyjnego aplikacji

*Przykładowe aplikacje do implementowania protokołu TLS w zarządzanym systemie .NET*

Udostępniono przykładowe aplikacje, które przedstawiają implementację protokołu TLS dla zarządzanego systemu .NET w produkcie IBM MQ classes for .NET, w produkcie XMS .NET i w niestandardowym kanale produktu IBM MQ dla produktu WCF.

W poniższej tabeli przedstawiono położenie przykładowych aplikacji. *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

*Tabela 79. Położenie przykładowych aplikacji do zaimplementowania protokołu TLS w zarządzanym systemie .NET*

| **Oferta stosu IBM MQ.NET** | **Położenie próbek** |
|---|---|
| podstawowe.NET | *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimplePut\SimplePut.cs<br>*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<br>*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleConsumer\SimpleConsumer.cs |
| Kanał niestandardowy IBM MQ dla WCF | *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\MQMessagingOneWayService.cs |

## Windows Korzystanie z programu .NET Monitor

Monitor .NET jest aplikacją podobną do monitora wyzwalacza IBM MQ .

**Ważne:** Informacje na ten temat zawiera sekcja Składniki, które można używać tylko w ramach instalacji podstawowej w systemie Windows .

Można utworzyć komponenty .NET , których instancje są tworzone za każdym razem, gdy komunikat jest odbierany w monitorowanej kolejce i które następnie przetwarzają ten komunikat. Monitor .NET jest uruchamiany za pomocą komendy **runmqdnm** i zatrzymywany za pomocą komendy **endmqdnm** . Szczegółowe informacje na temat tych komend zawierają opisy komend runmqdnm i endmqdnm.

Aby korzystać z programu .NET Monitor, należy napisać komponent implementujący interfejs IMQObjectTrigger , który jest zdefiniowany w pliku amqmdnm.dll.

Komponenty mogą być transakcyjne lub nietransakcyjne. Komponent transakcyjny musi dziedziczyć z pliku System.EnterpriseServices.ServicedComponent i być zarejestrowany jako RequiresTransaction lub SupportsTransaction. Nie może być zarejestrowana jako RequiresNew , ponieważ program .NET Monitor już zainicjował transakcję.

Komponent odbiera obiekty MQQueueManager, MQQueue i MQMessage z produktu **runmqdnm**. Może również otrzymać łańcuch parametru użytkownika, jeśli został określony przy użyciu opcji wiersza komend *-u* podczas uruchamiania komendy `runmqdnm` . Należy zauważyć, że komponent odbiera treść komunikatu, który dotarł do monitorowanej kolejki w obiekcie MQMessage. Nie musi łączyć się z menedżerem kolejek, otwierać kolejki ani sam komunikat. Następnie komponent musi przetworzyć komunikat zgodnie z potrzebami i zwrócić sterowanie do programu .NET Monitor.

Jeśli komponent został zapisany jako komponent transakcyjny, rejestruje się w celu zatwierdzenia lub wycofania transakcji przy użyciu narzędzi udostępnianych przez System.EnterpriseServices.ServicedComponent.

Ponieważ komponent odbiera obiekty MQQueueManager i MQQueue, a także komunikat, posiada pełne informacje o kontekście dla tego komunikatu i może na przykład otworzyć inną kolejkę w tym samym menedżerze kolejek bez konieczności oddzielnego nawiązywania połączenia z produktem IBM MQ.

## Windows *Przykładowe fragmenty kodu*

Ten temat zawiera dwa przykłady komponentów, które uzyskują komunikat z programu .NET Monitor i drukują go, z których jeden korzysta z przetwarzania transakcyjnego, a drugi z przetwarzania nietransakcyjnego. Trzeci przykład przedstawia wspólne procedury narzędziowe, które mają zastosowanie do dwóch pierwszych przykładów. Wszystkie przykłady znajdują się w C#.

### Przykład 1: Przetwarzanie transakcyjne

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/
using System;
using System.EnterpriseServices;

using IBM.WMQ;
using IBM.WMQMonitor;

[assembly: ApplicationName("dnmsamp")]

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll TranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m QMNAME -q QNAME -a dnmsamp.dll -c Tran

namespace dnmsamp
{
  [TransactionAttribute(TransactionOption.Required)]
  public class Tran : ServicedComponent, IMQObjectTrigger
  {
    Util util = null;

    [AutoComplete(true)]
    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("Tran");

      if (param != null)
        util.Print("PARAM: '" +param.ToString() + "'");

      util.PrintMessage(message);
```

```
      //System.Console.WriteLine("SETTING ABORT");
      //ContextUtil.MyTransactionVote = TransactionVote.Abort;

      System.Console.WriteLine("SETTING COMMIT");
      ContextUtil.SetComplete();
      //ContextUtil.MyTransactionVote = TransactionVote.Commit;
    }
  }
}
```

**Przykład 2: Przetwarzanie nietransakcyjne**

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/

using System;

using IBM.WMQ;
using IBM.WMQMonitor;

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll NonTranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m QMNAME -q QNAME -a dnmsamp.dll -c NonTran
namespace dnmsamp
{
  public class NonTran : IMQObjectTrigger
  {
    Util util = null;

    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("NonTran");

      try
      {
        util.PrintMessage(message);
      }

      catch (Exception ex)
      {
        System.Console.WriteLine(">>> NonTran\n{0}", ex.ToString());
      }
    }
  }
}
```

**Przykład 3: procedury wspólne**

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/

using System;

using IBM.WMQ;

namespace dnmsamp
{
    /// <summary>
    /// Summary description for Util.
    /// </summary>
    public class Util
    {
    /* ---------------------------------------------------------------- */
    /* Default prefix string of the namespace.                           */
```

```
    /* ------------------------------------------------------------------ */
    private string prefixText = "dnmsamp";

    /* ------------------------------------------------------------------ */
    /* Constructor that takes the replacement prefix string to use.       */
    /* ------------------------------------------------------------------ */
    public Util(String text)
    {
      prefixText = text;
    }

    /* ------------------------------------------------------------------ */
    /* Display an arbitrary string to the console.                        */
    /* ------------------------------------------------------------------ */
    public void Print(String text)
    {
      System.Console.WriteLine("{0} {1}\n", prefixText, text);
    }



 /* ------------------------------------------------------------------ */
    /* Display the content of the message passed to the console.          */
    /* ------------------------------------------------------------------ */
    public void PrintMessage(MQMessage message)
    {
      if (message.Format.CompareTo(MQC.MQFMT_STRING) == 0)
      {
        try
        {
          string messageText = message.ReadString(message.MessageLength);

          Print(messageText);
        }

        catch(Exception ex)
        {
          Print(ex.ToString());
        }
      }
      else
      {
        Print("UNRECOGNISED FORMAT");
      }
    }

    /* ------------------------------------------------------------------ */
    /* Convert the byte array into a hex string.                          */
    /* ------------------------------------------------------------------ */
    static public string ToHexString(byte[] byteArray)
    {
      string hex = "0123456789ABCDEF";

      string retString = "";

      for(int i = 0; i < byteArray.Length; i++)
      {
        int h = (byteArray[i] & 0xF0)>>4;
        int l = (byteArray[i] & 0x0F);

        retString += hex.Substring(h,1) + hex.Substring(l,1);
      }

      return retString;
    }
  }
}
```

## Kompilowanie programów IBM MQ .NET

Komendy próbne służące do kompilowania aplikacji .NET napisanych w różnych językach.

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Aby zbudować aplikację C# przy użyciu programu IBM MQ classes for .NET, należy użyć następującej komendy:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin /out:MyProg.exe
MyProg.cs
```

Aby zbudować aplikację Visual Basic za pomocą programu IBM MQ classes for .NET, należy użyć następującej komendy:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:MyProg.exe MyProg.vb
```

Aby zbudować zarządzaną aplikację C++ za pomocą programu IBM MQ classes for .NET, należy użyć następującej komendy:

```
cl /clr MQ_INSTALLATION_PATH\bin Myprog.cpp
```

W przypadku innych języków należy zapoznać się z dokumentacją dostarczoną przez dostawcę języka.

## Korzystanie z autonomicznego klienta IBM MQ .NET

Klient IBM MQ .NET umożliwia tworzenie pakietów i wdrażanie zespołu produktu IBM MQ .NET bez konieczności używania pełnej instalacji klienta IBM MQ w systemach produkcyjnych do uruchamiania aplikacji.

### Zanim rozpoczniesz

V 9.3.1 W katalogu IBM MQ 9.3.1biblioteka klienta `amqmdnetstd.dll` zainstalowana w położeniu domyślnym jest oparta na katalogu .NET 6. Biblioteka klienta `amqmdnetstd.dll` o nazwie .NET Standard została przeniesiona do nowego położenia w pakiecie instalacyjnym klienta IBM MQ i jest teraz dostępna w następujących miejscach:

- Windows W systemie Windows: *MQ_INSTALLATION_PATH*`\bin\netstandard2.0`
- Linux W systemie Linux: *MQ_INSTALLATION_PATH*`\lib64\netstandard2.0`

LTS W przypadku systemu IBM MQ 9.3.0 Long Term Supportbiblioteka `amqmdnetstd.dll` jest dostępna w następujących miejscach:

- Windows W systemie Windows: *MQ_INSTALLATION_PATH*`\bin`. Aplikacje przykładowe są instalowane w produkcie *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.
- Linux W systemie Linux: *MQ_INSTALLATION_PATH*`/lib64 path`. Przykłady .NET znajdują się w sekcji *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.

LTS Stabilized Biblioteka `amqmdnet.dll` jest nadal dostarczana, ale ta biblioteka jest ustabilizowana, co oznacza, że nie zostaną do niej wprowadzone żadne nowe funkcje. W przypadku wszystkich najnowszych składników należy przeprowadzić migrację do biblioteki `amqmdnetstd.dll` . Można jednak nadal korzystać z biblioteki `amqmdnet.dll` w systemie IBM MQ 9.1 Long Term Support lub Continuous Delivery .

### O tym zadaniu

Aplikacje produktu IBM MQ .NET można zbudować na komputerze, na którym jest zainstalowany pełny klient IBM MQ , a następnie można utworzyć pakiet zespołu produktu IBM MQ .NET ( `amqmdnetstd.dll`) razem z aplikacją i wdrożyć go w systemach produkcyjnych.

Tworzone i wdrażane aplikacje mogą być tradycyjnymi aplikacjami .NET , usługami lub aplikacjami Microsoft Azure Web/Worker.

W takich wdrożeniach klient IBM MQ .NET obsługuje tylko tryb zarządzany połączenia z menedżerem kolejek. Powiązania serwera i połączenia w trybie klienta niezarządzanego nie są dostępne, ponieważ te dwa tryby wymagają pełnej instalacji klienta IBM MQ . Każda próba użycia tych dwóch innych trybów powoduje wystąpienie wyjątku aplikacji.

## Procedura

Odwołanie do zespołu klienta IBM MQ .NET w aplikacjach

- Odwołaj się do zespołu `amqmdnetstd.dll` w aplikacji w taki sam sposób, jak w przypadku wcześniejszych wersji.

  Ustaw właściwość **CopyLocal** zespołu `amqmdnetstd.dll` na wartość `True` , aby upewnić się, że zespół `amqmdnetstd.dll` jest kopiowany do katalogu `bin` aplikacji. Ustawienie tej właściwości pomaga również narzędziu do tworzenia pakietów aplikacji w spakowaniu wymaganych plików binarnych na potrzeby wdrożenia w systemach produkcyjnych oraz w środowiskach chmurowych Microsoft Azure PaaS .

Dodawanie obsługi transakcji globalnych

- Upewnij się, że aplikacja wdraża aplikację monitorującą WMQDotnetXAMonitor na komputerze razem z aplikacją.

  Jeśli aplikacja używa funkcji globalnej transakcji zarządzanej przez IBM MQ .NET , musi również wdrożyć program WMQDotnetXAMonitor na komputerze razem z samą aplikacją. Ten program narzędziowy jest wymagany do odzyskiwania wszystkich wątpliwych transakcji.

Uruchamianie i zatrzymywanie śledzenia

- Tylko w przypadku systemu IBM MQ classes for .NET Framework , aby uruchomić i zatrzymać śledzenie za pomocą pliku konfiguracyjnego aplikacji i pliku konfiguracyjnego śledzenia specyficznego dla systemu IBM MQ , należy zapoznać się z sekcją Śledzenie klas IBM MQ dla klienta .NET Framework za pomocą pliku konfiguracyjnego aplikacji.

  Należy użyć pliku konfiguracyjnego aplikacji i pliku konfiguracyjnego śledzenia specyficznego dla systemu IBM MQ , ponieważ ponieważ nie ma pełnej instalacji klienta IBM MQ , standardowe narzędzia używane do uruchamiania i zatrzymywania śledzenia, **strmqtrc** i **endmqtrc**, nie są dostępne.

  **Uwagi:**

  - Ta metoda generowania danych śledzenia ma zastosowanie do klienta zarządzanego podlegającego redystrybucji .NET , a także do autonomicznego klienta .NET . Patrz środowisko wykonawcze aplikacji.NET -tylko Windows.
  - Plik konfiguracyjny aplikacji nie jest obsługiwany w IBM MQ classes for .NET (biblioteki.NET Standard i .NET 6 ). Aby włączyć śledzenie dla bibliotek IBM MQ classes for .NET (.NET Standard i .NET 6 ), należy użyć zmiennej środowiskowej **MQDOTNET_TRACE_ON** . Patrz Śledzenie aplikacji IBM MQ .NET przy użyciu zmiennych środowiskowych.

- V 9.3.3

  Uruchom i zatrzymaj śledzenie, korzystając z pliku `mqclient.ini` i ustawiając odpowiednie właściwości w sekcji Trace.

  Patrz sekcja Śledzenie aplikacji produktu IBM MQ .NET za pomocą pliku mqclient.ini.

  Z poziomu systemu IBM MQ 9.3.3można skonfigurować śledzenie za pomocą pliku `mqclient.ini` i ustawić odpowiednie właściwości sekcji Trace. Można również dynamicznie włączać i wyłączać śledzenie za pomocą pliku `mqclient.ini` .

Włączanie przekierowania powiązania w pliku konfiguracyjnym aplikacji

- Aby włączyć odwołanie do powiązania czasu kompilacji zespołu produktu IBM MQ .NET z późniejszą wersją zespołu, należy dodać właściwość <dependentAssembly> do pliku konfiguracyjnego aplikacji.

  Poniższy przykładowy fragment kodu w pliku `app.config` przekierowuje aplikację, która została skompilowana przy użyciu wersji IBM MQ 8.0.0 Fix Pack 2 (`8.0.0.2`) zespołu produktu IBM MQ .NET ,

ale później zastosowano pakiet poprawek IBM MQ 8.0.0 Fix Pack 3, który zaktualizował zespół produktu IBM MQ.NET do wersji 8.0.0.3.

```
<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>
```

**Pojęcia pokrewne**

“Instalowanie produktu IBM MQ classes for .NET” na stronie 574
Produkt IBM MQ classes for .NET, w tym przykłady, jest instalowany razem z produktem IBM MQ w systemie Windows i Linux

Klienty podlegające redystrybucji

Środowisko wykonawcze aplikacji .NET -tylko Windows

**Zadania pokrewne**

“Korzystanie z aplikacji WMQDotnetXAMonitor” na stronie 594
Klient IBM MQ .NET udostępnia aplikację monitora XA, WmqDotnetXAMonitor, której można użyć do odtwarzania dowolnych niekompletnych transakcji rozproszonych. Aplikacja WmqDotnetXAMonitor nawiązuje połączenie z menedżerem kolejek, w którym transakcje są wątpliwe, a następnie rozstrzyga transakcję na podstawie ustawionych parametrów.

Śledzenie aplikacji IBM MQ .NET

## V 9.3.0 OutboundSNI właściwość

Właściwość **OutboundSNI** można ustawić w aplikacji za pomocą właściwości lub zmiennej środowiskowej.

W programie IBM MQ 9.3.0można ustawić MQC.OUTBOUND_SNI_PROPERTY w aplikacji, używając tabeli mieszającej podczas łączenia się z menedżerem kolejek za pomocą klasy `MQQueueManager` .

MQC.OUTBOUND_SNI_PROPERTY przyjmuje następujące wartości:

- MQC.OUTBOUND_SNI_CHANNEL, który jest odwzorowywany na "CHANNEL"
- MQC.OUTBOUND_SNI_HOSTNAME, która jest odwzorowana na "HOSTNAME"
- MQC.OUTBOUND_SNI_ASTERISK, która jest odwzorowana na "*"

Dodatkowo można ustawić właściwość **OutboundSNI** przy użyciu zmiennej środowiskowej MQOUTBOUND_SNI, która przyjmuje następujące wartości:

- CHANNEL
- HOSTNAME
- *

i ustaw wartość **OutboundSNI** w pliku `App.config` , tak jak w przypadku każdej innej właściwości `mqclient.ini` .

**Uwaga:** Wartością domyślną tej właściwości jest MQC.OUTBOUND_SNI_CHANNEL , jeśli nie ustawiono konkretnej wartości.

Kolejność wykonywania operacji ustawiania właściwości **OutboundSNI** w węźle zarządzanym jest następująca:

1. Właściwość poziomu aplikacji
2. Zmienna środowiskowa

Dla właściwości **OutboundSNI** w węźle niezarządzanym obsługiwana jest tylko wartość `mqclient.ini` .

Właściwości ustawione w pliku `App.config` mają zastosowanie tylko do aplikacji .NET Framework .

Jeśli zostanie podana wartość, która nie jest poprawna na poziomie aplikacji lub w pliku `App.config` , zostanie wygenerowany kod powrotu MQRC_OUTBOUND_SNI_NOT_VALID.

Jeśli zostanie ustawiona niepoprawna zmienna środowiskowa lub zostanie podana wartość niepoprawna w pliku `mqclient.ini` , zostanie użyta wartość domyślna `CHANNEL` .

### OutboundSNI i wiele certyfikatów

Produkt IBM MQ korzysta z nagłówka SNI w celu udostępnienia wielu funkcji certyfikatów. Jeśli aplikacja łączy się z kanałem IBM MQ skonfigurowanym do używania innego certyfikatu za pomocą pola CERTLABL, aplikacja musi połączyć się z ustawieniem **OutboundSNI** o wartości CHANNEL.

Jeśli aplikacja z ustawieniem **OutboundSNI** innym niż CHANNEL nawiąże połączenie z kanałem ze skonfigurowaną etykietą certyfikatu, aplikacja zostanie odrzucona z błędem MQRC_SSL_INITIALIZATION_ERROR i w dziennikach błędów menedżera kolejek zostanie zapisany komunikat AMQ9673 .

Więcej informacji na temat sposobu, w jaki program IBM MQ udostępnia funkcje obsługi wielu certyfikatów, zawiera sekcja W jaki sposób program IBM MQ udostępnia możliwość obsługi wielu certyfikatów .

## Tworzenie aplikacji XMS .NET

IBM MQ Message Service Client (XMS) for .NET (XMS .NET) udostępnia aplikacyjny interfejs programistyczny (API) o nazwie XMS , który ma ten sam zestaw interfejsów co Java Message Service (JMS) Interfejs API. IBM MQ Message Service Client (XMS) for .NET zawiera w pełni zarządzaną implementację języka XMS, która może być używana przez dowolny język zgodny z językiem .NET .

### O tym zadaniu

XMS obsługuje:

- Przesyłanie komunikatów w modelu punkt-punkt
- Przesyłanie komunikatów w trybie publikowania/subskrypcji
- Synchroniczne dostarczanie komunikatów
- Asynchroniczne dostarczanie komunikatów

Aplikacja XMS może wymieniać komunikaty z następującymi typami aplikacji:

- Aplikacja XMS
- Aplikacja IBM MQ classes for JMS
- Rodzima aplikacja IBM MQ
- Aplikacja JMS, która używa domyślnego dostawcy przesyłania komunikatów produktu IBM MQ .

Aplikacja XMS może łączyć się z dowolnym z następujących serwerów przesyłania komunikatów i korzystać z nich:

**IBM MQ menedżer kolejek**
Aplikacja może nawiązywać połączenia w trybie powiązań lub w trybie klienta.

**WebSphere Application Server service integration bus**
Aplikacja może korzystać z bezpośredniego połączenia TCP/IP lub z protokołu HTTP przez TCP/IP.

**IBM Integration Bus**
Komunikaty są transportowane między aplikacją i brokerem przy użyciu produktu WebSphere MQ Real-Time Transport. Komunikaty mogą być dostarczane do aplikacji za pomocą programu WebSphere MQ Multicast Transport.

Po nawiązaniu połączenia z menedżerem kolejek produktu IBM MQ aplikacja produktu XMS może używać języka WebSphere MQ Enterprise Transport do komunikowania się z produktem IBM Integration Bus. Alternatywnie aplikacja XMS może publikować i subskrybować, nawiązując połączenie z produktem IBM MQ.

Od wersji IBM MQ 9.1.1 IBM MQ obsługuje .NET Core dla aplikacji w środowiskach Windows . Aby uzyskać więcej informacji, zapoznaj się z sekcją: “Instalowanie produktu IBM MQ classes for XMS .NET” na stronie 640.

Od wersji IBM MQ 9.1.2 IBM MQ obsługuje .NET Core dla aplikacji w środowiskach Linux .

Z poziomu produktu IBM MQ 9.1.4aplikacje zarządzane przez produkt XMS .NET mogą automatycznie równoważyć połączenia między menedżerami kolejek w klastrze. Obsługiwane są zarówno biblioteki .NET Framework , jak i .NET Standard . Więcej informacji na ten temat zawiera sekcja Informacje o jednolitych klastrach i sekcja Automatyczne równoważenie aplikacji.

**Zadania pokrewne**
Kontakt z działem wsparcia IBM
Rozwiązywanie problemów z systemem XMS .NETproblems

# Style przesyłania komunikatów obsługiwane przez produkt XMS

Produkt XMS obsługuje style przesyłania komunikatów typu punkt z punktem i publikowanie/subskrypcja.

Style przesyłania komunikatów są również nazywane domenami przesyłania komunikatów.

## Przesyłanie komunikatów w modelu punkt-punkt

Powszechną formą przesyłania komunikatów w trybie punkt z punktem jest kolejkowanie. W najprostszym przypadku aplikacja wysyła komunikat do innej aplikacji, identyfikując niejawnie lub jawnie kolejkę docelową. Bazowy system przesyłania komunikatów i kolejkowania odbiera komunikat z aplikacji wysyłającej i kieruje go do kolejki docelowej. Aplikacja odbierająca może następnie pobrać komunikat z kolejki.

Jeśli bazowy system przesyłania komunikatów i kolejkowania zawiera komunikat IBM Integration Bus, produkt IBM Integration Bus może replikować komunikat i kierować jego kopie do różnych kolejek. W rezultacie komunikat może zostać odebrany przez więcej niż jedną aplikację. Produkt IBM Integration Bus może również transformować komunikat i dodawać do niego dane.

Kluczową cechą przesyłania komunikatów w trybie punkt z punktem jest to, że aplikacja umieszcza komunikat w kolejce lokalnej podczas wysyłania komunikatu. Bazowy system przesyłania komunikatów i kolejkowania określa, do której kolejki docelowej jest wysyłany komunikat. Aplikacja odbierająca pobiera komunikat z kolejki docelowej.

## Przesyłanie komunikatów w trybie publikowania/subskrypcji

W przesyłaniu komunikatów w trybie publikowania/subskrypcji istnieją dwa typy aplikacji: publikator i subskrybent.

*Publikator* dostarcza informacje w postaci komunikatów publikacji. Gdy publikator publikuje komunikat, określa temat, który identyfikuje temat informacji w komunikacie.

*Subskrybent* jest konsumentem publikowanych informacji. Subskrybent określa tematy, którymi jest zainteresowany, tworząc subskrypcje.

System publikowania/subskrypcji odbiera publikacje od publikatorów i subskrypcje od subskrybentów. Kieruje publikacje do subskrybentów. Subskrybent otrzymuje publikacje tylko w tych tematach, które zasubskrybował.

Kluczową cechą przesyłania komunikatów w trybie publikowania/subskrypcji jest to, że publikator identyfikuje temat podczas publikowania komunikatu. Nie identyfikuje on subskrybentów. Jeśli komunikat jest publikowany w temacie, dla którego nie ma subskrybentów, żadna aplikacja nie odbiera komunikatu.

Aplikacja może być zarówno publikatorem, jak i subskrybentem.

# Model obiektu XMS

Interfejs API XMS jest interfejsem obiektowym. Model obiektu XMS jest oparty na modelu obiektu JMS 1.1 .

## Główne klasy języka XMS

Główne klasy XMS lub typy obiektów są następujące:

**ConnectionFactory**
Obiekt `ConnectionFactory` hermetyzuje zestaw parametrów połączenia. Aplikacja używa `ConnectionFactory` do utworzenia połączenia. Aplikacja może udostępnić parametry w czasie wykonywania i utworzyć obiekt `ConnectionFactory` . Alternatywnie parametry połączenia mogą być przechowywane w repozytorium obiektów administrowanych. Aplikacja może pobrać obiekt z repozytorium i utworzyć na jego podstawie obiekt `ConnectionFactory` .

**Połączenie**
Obiekt `Connection` hermetyzuje aktywne połączenie między aplikacją a serwerem przesyłania komunikatów. Aplikacja używa połączenia do tworzenia sesji.

**Miejsce docelowe**
Aplikacja wysyła lub odbiera komunikaty przy użyciu obiektu `Destination` . W domenie publikowania/subskrypcji obiekt `Destination` hermetyzuje temat, a w domenie typu punkt z punktem obiekt `Destination` hermetyzuje kolejkę. Aplikacja może udostępnić parametry w celu utworzenia obiektu `Destination` w czasie wykonywania. Alternatywnie można utworzyć obiekt `Destination` na podstawie definicji obiektu, która jest przechowywana w repozytorium obiektów administrowanych.

**Sesja**
Obiekt `Session` jest jednowątkowym kontekstem na potrzeby wysyłania i odbierania komunikatów. Aplikacja używa obiektu `Session` do tworzenia obiektów `Message`, `MessageProducer`i `MessageConsumer` .

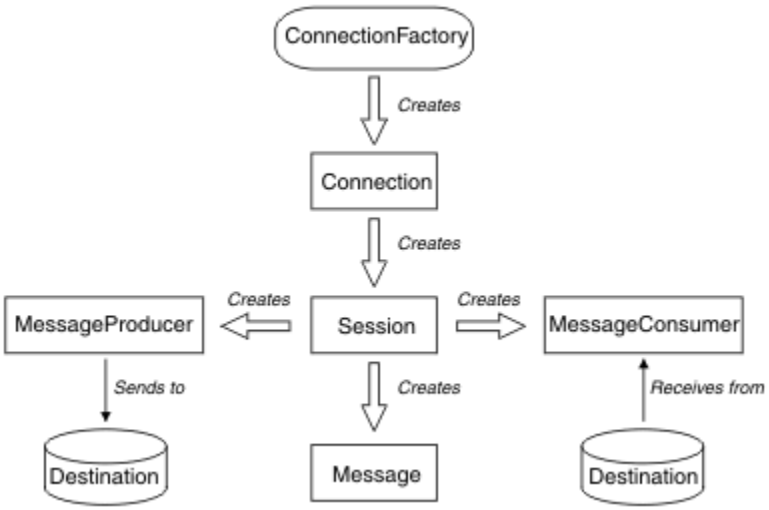
**Komunikat**
Obiekt `Message` hermetyzuje obiekt `Message` , który aplikacja wysyła za pomocą obiektu `MessageProducer` lub odbiera za pomocą obiektu `MessageConsumer` .

**MessageProducer**
Obiekt `MessageProducer` jest używany przez aplikację do wysyłania komunikatów do miejsca docelowego.

**MessageConsumer**
Obiekt `MessageConsumer` jest używany przez aplikację do odbierania komunikatów wysyłanych do miejsca docelowego.

## Obiekty XMS i ich relacje

Rysunek 52 na stronie 638 przedstawia główne typy obiektu XMS : ConnectionFactory, Connection, Session, MessageProducer, MessageConsumer, Message i Destination. Aplikacja używa fabryki połączeń do utworzenia połączenia i używa połączenia do utworzenia sesji. Aplikacja może następnie użyć sesji do utworzenia komunikatów, producentów komunikatów i konsumentów komunikatów. Aplikacja używa producenta komunikatów do wysyłania komunikatów do miejsca docelowego, a konsumenta komunikatów do odbierania komunikatów wysyłanych do miejsca docelowego.




*Rysunek 52. Obiekty XMS i ich relacje*

W systemie XMS .NETklasy XMS są zdefiniowane jako zestaw interfejsów .NET . Podczas kodowania aplikacji XMS .NET potrzebne są tylko zadeklarowane interfejsy.

Model obiektowy XMS jest oparty na niezależnych od domeny interfejsach, które zostały opisane w specyfikacji Java Message Service Specification, Version 1.1. Klasy specyficzne dla domeny, takie jak `Topic`, `TopicPublisher`i `TopicSubscriber`, nie są udostępniane.

### Atrybuty i właściwości obiektów XMS

Obiekt XMS może mieć atrybuty i właściwości, które są charakterystyką obiektu i są implementowane na różne sposoby:

**Atrybuty**
Cecha obiektu, która jest zawsze obecna i zajmuje pamięć masową, nawet jeśli atrybut nie ma wartości. Pod tym względem atrybut jest podobny do pola w strukturze danych o stałej długości. Cechą wyróżniającą atrybuty jest to, że każdy atrybut ma własne metody ustawiania i pobierania wartości.

**Właściwości**
Właściwość obiektu jest obecna i zajmuje pamięć masową dopiero po ustawieniu jej wartości. Nie można usunąć właściwości lub odzyskać jej pamięci masowej po ustawieniu jej wartości. Można zmienić jego wartość. Produkt XMS udostępnia zestaw ogólnych metod służących do ustawiania i pobierania wartości właściwości.

## Obiekty administrowane

Za pomocą obiektów administrowanych można administrować ustawieniami połączenia używanymi przez aplikacje klienckie do administrowania z centralnego repozytorium. Aplikacja pobiera definicje obiektów z centralnego repozytorium i używa ich do tworzenia obiektów `ConnectionFactory` i `Destination` . Za pomocą obiektów administrowanych można pobrać aplikacje z zasobów, które są przez nie używane w czasie wykonywania.

Na przykład aplikacje XMS mogą być napisane i przetestowane z obiektami administrowanymi, które odwołują się do zestawu połączeń i miejsc docelowych w środowisku testowym. Po wdrożeniu aplikacji można zmienić obiekty administrowane w celu skonfigurowania aplikacji tak, aby odwoływał się do połączeń i miejsc docelowych w środowisku produkcyjnym.

XMS obsługuje dwa typy obiektów administrowanych:

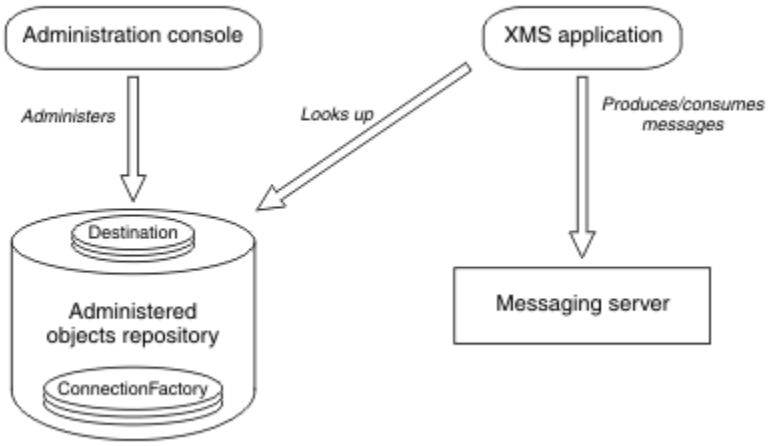
- Obiekt `ConnectionFactory` , który jest używany przez aplikacje do nawiązania początkowego połączenia z serwerem.

- Obiekt `Destination` używany przez aplikacje do określania miejsca docelowego dla wysyłanych komunikatów oraz źródła odbieranych komunikatów. Miejscem docelowym jest temat lub kolejka na serwerze, z którym łączy się aplikacja.

Narzędzie administracyjne **JMSAdmin** jest dostarczane z produktem IBM MQ. Jest on używany do tworzenia i zarządzania obiektami administrowanymi w centralnym repozytorium obiektów administrowanych.

Obiekty administrowane w repozytorium mogą być używane przez aplikacje IBM MQ classes for JMS i XMS . Aplikacje XMS mogą używać obiektów `ConnectionFactory` i `Destination` do nawiązywania połączeń z IBM MQ menedżerem kolejek. Administrator może zmienić definicje obiektów przechowywane w repozytorium bez wpływu na kod aplikacji.

Na poniższym diagramie przedstawiono, w jaki sposób aplikacja XMS zwykle używa obiektów administrowanych. Po lewej stronie diagramu znajduje się repozytorium zawierające definicje obiektów ConnectionFactory i Destination, które są administrowane za pomocą konsoli administracyjnej. Po prawej stronie diagramu znajduje się aplikacja XMS , która wyszukuje definicje obiektów w repozytorium, a następnie używa tych definicji obiektów podczas nawiązywania połączenia z serwerem przesyłania komunikatów.




*Rysunek 53. Typowe użycie obiektów administrowanych przez aplikację XMS*

## Model komunikatów XMS

Model komunikatów XMS jest taki sam jak model komunikatów IBM MQ classes for JMS .

W szczególności XMS implementuje te same pola nagłówka komunikatu i właściwości komunikatu, które są implementowane przez produkt IBM MQ classes for JMS :

- Pola nagłówka JMS . Te pola mają nazwy rozpoczynające się od przedrostka JMS.
- JMS zdefiniowanych właściwości. Te pola mają właściwości, których nazwy rozpoczynają się od przedrostka JMSX.
- IBM zdefiniowanych właściwości. Te pola mają właściwości, których nazwy rozpoczynają się od przedrostka JMS_IBM_.

W rezultacie aplikacje XMS mogą wymieniać komunikaty z aplikacjami IBM MQ classes for JMS . W każdym komunikacie niektóre pola i właściwości nagłówka są ustawiane przez aplikację, a inne przez produkt XMS lub IBM MQ classes for JMS. Niektóre pola ustawione przez XMS lub IBM MQ classes for JMS są ustawiane podczas wysyłania komunikatu, a inne podczas odbierania komunikatu. Pola nagłówka i właściwości są propagowane wraz z komunikatem przez serwer przesyłania komunikatów, jeśli jest to odpowiednie. Są one udostępniane każdej aplikacji, która odbiera komunikat.

**Pojęcia pokrewne**

IBM MQ classes for JMS

## Windows Linux Instalowanie produktu IBM MQ classes for XMS .NET

Produkt IBM MQ classes for XMS .NET, w tym przykłady, jest instalowany razem z produktem IBM MQ w systemie Windows i Linux.

Począwszy od wersji IBM MQ 9.2.0, Microsoft.NET Core 3.1 jest minimalną wersją wymaganą do uruchomienia IBM MQ classes for XMS .NET Standard.

V 9.3.0 V 9.3.0 Począwszy od wersji IBM MQ 9.3.0, IBM MQ obsługuje aplikacje .NET 6 korzystające z produktu IBM MQ classes for XMS .NET Standard. Jeśli używana jest aplikacja .NET Core 3.1 , można uruchomić tę aplikację z niewielką modyfikacją w pliku `csproj` , ustawiając parametr `targetframeworkversion` na wartość `"net6.0"`, bez konieczności ponownego kompilowania.

V 9.3.1 IBM MQ 9.3.1 udostępnia bibliotekę klienta XMS .NET zbudowaną w oparciu o środowisko .NET 6 jako środowisko docelowe. Począwszy od wersji IBM MQ 9.3.1, Microsoft .NET 6.0 jest minimalną wymaganą wersją do uruchamiania aplikacji korzystających z bibliotek IBM MQ , które zostały zbudowane przy użyciu środowiska .NET 6 jako środowiska docelowego.

V 9.3.1 W systemie IBM MQ 9.3.1biblioteka klienta XMS .NET zbudowana przy użyciu pliku .NET Standard jest dostępna w nowym folderze `netstandard2.0` , a biblioteka klienta XMS .NET zbudowana przy użyciu pliku .NET 6 jako struktura docelowa jest dostępna w produkcie *MQ_INSTALLATION_PATH*`/bin` w systemie Windows i w produkcie *MQ_INSTALLATION_PATH*`/lib64` w systemie Linux.

### `amqmxmsstd.dll` biblioteka

V 9.3.1 W systemie IBM MQ 9.3.1biblioteka `amqmxmsstd.dll` jest dostępna w następujących miejscach:

**Biblioteka zbudowana przy użyciu środowiska .NET Standard 2.0 jako struktury docelowej**

- Windows W systemie Windows: *MQ_INSTALLATION_PATH*`\bin\netstandard2.0`.
- Linux W systemie Linux: *MQ_INSTALLATION_PATH*`\lib64\netstandard2.0`.

Deprecated Te biblioteki są nieaktualne i produkt IBM zamierza je usunąć w przyszłych wersjach.

**Biblioteka zbudowana przy użyciu środowiska .NET 6 jako struktury docelowej**

- Windows W systemie Windows: *MQ_INSTALLATION_PATH*`\bin`. Aplikacje przykładowe są instalowane w produkcie *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.
- Linux W systemie Linux: *MQ_INSTALLATION_PATH*`\lib64`. Przykłady .NET znajdują się w sekcji *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.

LTS W przypadku produktu IBM MQ 9.3.0 Long Term Supportbiblioteka IBM MQ classes for XMS .NET Standard ( `amqmxmsstd.dll`) jest dostępna w następujących miejscach:

- Windows W systemie Windows: *MQ_INSTALLATION_PATH*`\bin`. Aplikacje przykładowe są instalowane w produkcie *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/xms`.
- Linux W systemie Linux: *MQ_INSTALLATION_PATH*`/lib64 path`. Przykłady .NET znajdują się w sekcji *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/xms`.

Więcej informacji na ten temat zawiera “Instalowanie produktu IBM MQ classes for .NET” na stronie 574.

**Ostrzeżenie:** Deprecated V 9.3.1 Od wersji IBM MQ 9.3.1biblioteki klienta IBM MQ .NET zbudowane przy użyciu środowiska .NET Standard 2.0 jako środowiska docelowego są

nieaktualne, a aplikacje odwołujące się do tych bibliotek zgłaszają ostrzeżenie CS0618 podczas kompilacji.

Stabilized Wszystkie biblioteki `IBM.XMS.*` są nadal dostarczane, ale te biblioteki są ustabilizowane, co oznacza, że nie zostaną do nich wprowadzone żadne nowe funkcje. W przypadku wszystkich najnowszych funkcji należy przeprowadzić migrację do biblioteki `amqmxmsstd.dll` . Można jednak nadal używać istniejących bibliotek w wersji IBM MQ 9.1 Long Term Support lub Continuous Delivery .

V 9.3.1 Jeśli aplikacja .NET Framework jest kompilowana przy użyciu pliku `amqmdnetstd.dll` lub `amqmxmsstd.dll` z wersji wcześniejszej niż IBM MQ 9.3.1 i ta sama aplikacja jest uruchamiana przy użyciu bibliotek klienta IBM MQ opartych na pliku .NET 6 , to wyjątek typu `FileLoad` jest zgłaszany przez .NET:

```
Wychwycono wyjątek: System.IO.FileLoadException: Nie można załadować pliku lub zespołu
'amqmdnetstd, Version =x.x.x.x, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e' lub
jedną z jej zależności. Odnaleziona definicja manifestu zespołu nie jest zgodna z
numer referencyjny zespołu. (Wyjątek z HRESULT: 0x80131040)

Nazwa pliku: ' amqmdnetstd, Version =x.x.x.x, Culture=neutral,
PublicKeyToken=23d6cb914eeaac0e'
```

Aby rozwiązać ten błąd, biblioteki znajdujące się w produkcie *MQ_INSTALLATION_PATH*`/bin/netstandard2.0` muszą zostać skopiowane do katalogu, w którym działa aplikacja .NET Framework .

W systemie IBM MQ 9.2.0plik IBM MQ classes for XMS .NET Standard jest dostępny do pobrania z repozytorium NuGet . Pakiet NuGet zawiera zarówno bibliotekę `amqmxmsstd.dll` , jak i bibliotekę `amqmdnetstd.dll` . Produkt `amqmxmsstd.dll` jest zależny od produktu `amqmdnetstd.dll` i podczas tworzenia pakietu aplikacji XMS .NET Core, zarówno produkt `amqmxmsstd.dll` , jak i produkt `amqmdnetstd.dll` powinny być spakowane razem z aplikacją XMS .NET Core. Więcej informacji na ten temat zawiera sekcja “Pobieranie produktu IBM MQ classes for XMS .NET z repozytorium platformy NuGet” na stronie 643.

## Komenda dspmqver

Komenda **dspmqver** umożliwia wyświetlenie informacji o wersji i kompilacji dla komponentu .NET Core .

## Porównanie bibliotek IBM MQ classes for XMS .NET Framework i IBM MQ classes for XMS .NET (.NET Standard i .NET 6 )

Poniższa tabela zawiera listę funkcji produktu IBM MQ classes for XMS .NET Framework w porównaniu z funkcjami produktu IBM MQ classes for XMS .NET (biblioteki.NET Standard i .NET 6 ).

*Tabela 80. Różnice między bibliotekami IBM MQ classes for XMS .NET Framework i IBM MQ classes for XMS .NET (biblioteki.NET Standard i .NET 6 )*

| Funkcja | IBM MQ classes for XMS .NET Framework | IBM MQ classes for XMS .NET (biblioteki.NET Standard i .NET 6 ) |
|---|---|---|
| Nazwy klas (interfejsy API) | Wszystkie klasy pozostają takie same w każdej sieci. | Wszystkie klasy pozostają takie same w każdej sieci. |
| System operacyjny | Windows | Windows<br>Kontenery Docker<br>Linux<br>macOS |
| Plik `app.config` (plik konfiguracyjny do włączania śledzenia w kliencie redystrybuowanym) | Plik `app.config` służy do włączania śledzenia dla pakietu podlegającego redystrybucji. | `app.config` nie jest obsługiwana. Użyj zmiennych środowiskowych. |

*Tabela 80. Różnice między bibliotekami IBM MQ classes for XMS .NET Framework i IBM MQ classes for XMS .NET (biblioteki.NET Standard i .NET 6 ) (kontynuacja)*

| **Funkcja** | **IBM MQ classes for XMS .NET Framework** | **IBM MQ classes for XMS .NET (biblioteki.NET Standard i .NET 6 )** |
|---|---|---|
| Śledzenie | Aby śledzić klienta XMS .NET , można użyć istniejących zmiennych środowiskowych, takich jak zmienna środowiskowa **XMS_TRACE_ON** używana do włączenia śledzenia. Więcej informacji na ten temat zawiera sekcja Konfigurowanie śledzenia produktu XMS .NET przy użyciu zmiennych środowiskowych produktu XMS.<br>W przypadku klientów podlegających redystrybucji do włączenia śledzenia można użyć pliku `app.config` . | Aby śledzić klienta XMS .NET , można użyć istniejących zmiennych środowiskowych, takich jak zmienna środowiskowa **XMS_TRACE_ON** używana do włączenia śledzenia. Więcej informacji na ten temat zawiera sekcja Konfigurowanie śledzenia produktu XMS .NET przy użyciu zmiennych środowiskowych produktu XMS. |
| Tryby transportu | Zarządzany, Niezarządzany i Powiązania. | Zarządzany |
| TLS | Magazyn kluczy systemu Windows służy do przechowywania certyfikatów. | W systemie Windows certyfikaty należy przechowywać w magazynie kluczy. Dozwolone wartości: *USER lub *SYSTEM. Zależnie od danych wejściowych klient IBM MQ .NET przeszukuje magazyn kluczy systemu Windows bieżącego użytkownika lub całego systemu.<br>W Linux zaleca się użycie klasy X509Store w celu zainstalowania certyfikatów. Program .NET Core instaluje certyfikaty w następującym położeniu: "`.dotnet/corefx/cryptography/x509stores`". |
| CCDT | Obsługiwany | Obsługiwana. Ustawienia ścieżki do tabeli CCDT są takie same jak w przypadku klas produktu .NET Framework. |
| Automatyczne ponowne łączenie klienta | Obsługiwany | Obsługiwany |
| Rozproszone transakcje | Obsługiwany | Nieobsługiwane |
| Instalowanie bibliotek dołączanych dynamicznie (bibliotek DLL) w pamięci podręcznej Global Assembly Cache (GAC) | Biblioteki DLL są instalowane w pamięci GAC jako część instalacji produktu IBM MQ. | Biblioteki DLL nie są instalowane w pamięci GAC jako część instalacji produktu IBM MQ. |
| Obsługa typów połączeń WMQ, WPM i RTT | Obsługuje typy połączeń WMQ, WPM i RTT | Obsługa tylko produktu WMQ |

| *Tabela 80. Różnice między bibliotekami IBM MQ classes for XMS .NET Framework i IBM MQ classes for XMS .NET (biblioteki.NET Standard i .NET 6 ) (kontynuacja)* | | |
|---|---|---|
| **Funkcja** | **IBM MQ classes for XMS .NET Framework** | **IBM MQ classes for XMS .NET (biblioteki.NET Standard i .NET 6 )** |
| Obiekty administrowane JNDI | Obsługuje LDAP i FileSystem | Obsługuje tylko system plików FileSystem |

V 9.3.0 V 9.3.0 W systemie IBM MQ 9.3.0, aby uruchomić program IBM MQ classes for XMS .NET Framework , należy zainstalować program Microsoft.NET Framework V4.7.2 lub nowszej.

**Zadania pokrewne**

“Korzystanie z przykładowych aplikacji XMS” na stronie 650
Przykładowe aplikacje produktu XMS .NET udostępniają przegląd wspólnych funkcji każdego interfejsu API. Można ich używać do weryfikowania instalacji i konfiguracji serwera przesyłania komunikatów oraz do tworzenia własnych aplikacji.

## Windows Linux Pobieranie produktu IBM MQ classes for XMS .NET z repozytorium platformy NuGet

Produkt IBM MQ classes for XMS .NET jest dostępny do pobrania z repozytorium NuGet , dzięki czemu może być łatwo używany przez programistów .NET .

### O tym zadaniu

NuGet jest menedżerem pakietów dla platform programistycznych Microsoft , w tym .NET. Narzędzia klienckie NuGet umożliwiają tworzenie i konsumowanie pakietów. Pakiet NuGet to pojedynczy skompresowany plik z rozszerzeniem `.nupkg` , który zawiera skompilowany kod (biblioteki DLL), inne pliki powiązane z tym kodem oraz opisowy manifest zawierający informacje, takie jak numer wersji pakietu.

Pakiet produktu `IBMXMSDotnetClient` NuGet , który zawiera zarówno bibliotekę `amqmdnetstd.dll` , jak i bibliotekę `amqmxmsstd.dll` , można pobrać z galerii NuGet , która jest centralnym repozytorium pakietów używanym przez wszystkich autorów i konsumentów pakietów.

**Uwaga:** V 9.3.1 Począwszy od wersji IBM MQ 9.3.1pakiet NuGet zawiera biblioteki zbudowane przy użyciu środowiska docelowego .NET Standard 2.0 i .NET 6 . Biblioteki dla systemu .NET Standard 2.0 są dostępne w folderze `netstandard2.0` , a biblioteki dla systemu .NET 6 w folderze `net6.0` .

Istnieją trzy sposoby pobierania pakietu `IBMXMSDotnetClient` :

- Za pomocą komendy Microsoft Visual Studio. Produkt NuGet jest dystrybuowany jako rozszerzenie Microsoft Visual Studio . Począwszy od wersji Microsoft Visual Studio 2012, produkt NuGet jest domyślnie wstępnie instalowany.
- Z poziomu wiersza komend za pomocą menedżera pakietów NuGet lub interfejsu CLI .NET .
- Za pomocą przeglądarki WWW.

Podobnie jak w przypadku pakietu podlegającego redystrybucji, śledzenie można włączyć za pomocą zmiennej środowiskowej **XMS_TRACE_ON**.

### Procedura

- Aby pobrać pakiet `IBMXMSDotnetClient` za pomocą interfejsu użytkownika menedżera pakietów w programie Microsoft Visual Studio, wykonaj następujące kroki:
  a) Kliknij prawym przyciskiem myszy projekt .NET , a następnie kliknij opcję **Zarządzaj pakietami Nuget**.
  b) Kliknij kartę **Przeglądaj** i wyszukaj łańcuch `"IBMXMSDotnetClient"`.
  c) Wybierz pakiet i kliknij przycisk **Instaluj**.

Podczas instalacji Menedżer pakietów udostępnia informacje o postępie w postaci instrukcji konsoli.

- Aby pobrać pakiet `IBMXMSDotnetClient` z wiersza komend, wybierz jedną z następujących opcji:
  - W programie NuGet Package Manager wprowadź następującą komendę:

    ```
    Install-Package IBMXMSDotnetClient -Version 9.1.4.0
    ```

    Podczas instalacji Menedżer pakietów udostępnia informacje o postępie w postaci instrukcji konsoli. Dane wyjściowe można przekierować do pliku dziennika.
  - W interfejsie wiersza komend .NET wprowadź następującą komendę:

    ```
    dotnet add package IBMXMSDotnetClient --version 9.1.4
    ```

- Za pomocą przeglądarki WWW pobierz pakiet `IBMXMSDotnetClient` z serwisu https://www.nuget.org/packages/IBMXMSDotnetClient.

**Pojęcia pokrewne**

"Instalowanie produktu IBM MQ classes for .NET" na stronie 574
Produkt IBM MQ classes for .NET, w tym przykłady, jest instalowany razem z produktem IBM MQ w systemie Windows i Linux

Informacje licencyjne dotyczące programu IBM MQ Client for .NET

**Zadania pokrewne**

"Pobieranie pliku IBM MQ classes for .NET z repozytorium platformy NuGet" na stronie 580
IBM MQ classes for .NET są dostępne do pobrania z repozytorium NuGet , dzięki czemu mogą być łatwo używane przez programistów .NET .

# Konfigurowanie środowiska serwera przesyłania komunikatów

Tematy w tej sekcji opisują sposób konfigurowania środowiska serwera przesyłania komunikatów w celu umożliwienia aplikacjom XMS nawiązywania połączeń z serwerem.

## O tym zadaniu

W przypadku aplikacji, które łączą się z menedżerem kolejek produktu IBM MQ , wymagany jest klient IBM MQ (lub menedżer kolejek w trybie powiązań).

Obecnie nie ma żadnych wymagań wstępnych dla aplikacji, które używają połączenia w czasie rzeczywistym z brokerem.

Środowisko serwera przesyłania komunikatów należy skonfigurować przed uruchomieniem dowolnej aplikacji XMS , w tym aplikacji przykładowych dostarczanych z produktem XMS.

Sekcja obejmuje następujące tematy:

- "Konfigurowanie menedżera kolejek i brokera dla aplikacji, która nawiązuje połączenie z menedżerem kolejek produktu IBM MQ" na stronie 647
- "Instalowanie produktu IBM MQ classes for XMS .NET" na stronie 640
- "Konfigurowanie brokera dla aplikacji, która używa połączenia w czasie rzeczywistym z brokerem" na stronie 648
- "Konfigurowanie magistrali integracji usług dla aplikacji, która łączy się z produktem WebSphere Application Server" na stronie 649

## Obiekty nasłuchiwania komunikatów w produkcie XMS .NET

Obiekt nasłuchiwania komunikatów jest używany do asynchronicznego odbierania komunikatów. W przeciwieństwie do wywołania `MessageConsumer.receive()` obiekt nasłuchiwania komunikatów nie blokuje wątku wywołującego, ale dostarcza komunikaty do metody wywołania zwrotnego określonej przez aplikację (zwykle jest to metoda `onMessage` ).

Dostarczanie komunikatów rozpoczyna się po wywołaniu metody `Connection.Start()` . Dostarczanie komunikatów można zatrzymać i wznowić w dowolnym momencie, używając odpowiednio metod `Connection.Stop()` i `Connection.Start()` .

Po wywołaniu metody `Connection.Start()` po ustawieniu obiektu nasłuchiwania komunikatów dla co najmniej jednego konsumenta w sesji sesja ta staje się sesją asynchroniczną. Gdy sesja staje się asynchroniczna, nie jest możliwe wywołanie żadnej synchronicznej metody XMS .NET . Na przykład: `MessageProducer.Send()`. Spowoduje to zgłoszenie wyjątku z kodem przyczyny IBM MQ MQRC_HCONN_ASYNC_ACTIVE (2500).

## Wywołania synchroniczne w sesji asynchronicznej

`Session.Close` jest jedynym wywołaniem synchronicznym, które jest dozwolone w sesji asynchronicznej. Aplikacje mogą również wykonywać wywołania synchroniczne (z wyjątkiem wywołania `Session.Close`) przy użyciu metody wywołania zwrotnego obiektu nasłuchiwania komunikatów, czyli metody `onMessage` .

Inne niż te dwie opcje wymagają zatrzymania połączenia przy użyciu metody `Connection.Stop()` dla aplikacji w celu wykonania dowolnego wywołania synchronicznego. Po wykonaniu wywołań należy wznowić połączenie przy użyciu metody `Connection.Start()` . który restartuje dostarczanie komunikatów.

## Ile asynchronicznych konsumentów komunikatów może mieć sesja?

Sesja może mieć wiele asynchronicznych konsumentów komunikatów. Jednak w dowolnym momencie komunikat jest dostarczany tylko do jednego konsumenta. Praktycznie oznacza to, że po nadejściu drugiego komunikatu, gdy produkt XMS .NET wywołał metodę `onMessage()` konsumenta w celu dostarczenia pierwszego komunikatu, drugi komunikat nie zostanie dostarczony do konsumenta w sesji, dopóki nie zostanie zwrócona metoda `onMessage()` .

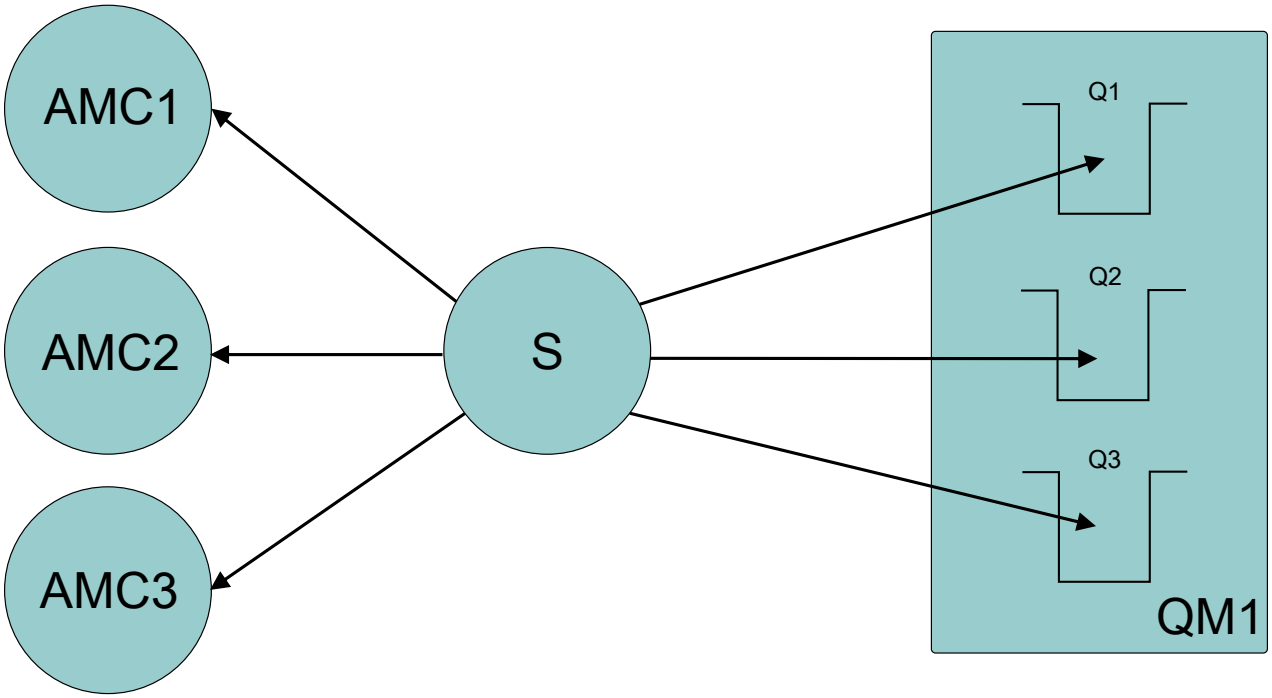
Drugi komunikat jest dostarczany do konsumenta w sesji tylko po powrocie z metody `onMessage()` . Dzieje się tak, ponieważ sesja zarządza dostarczaniem komunikatów do konsumentów przy użyciu tylko jednego wątku. Oznacza to, że w danym momencie można dostarczyć tylko jeden komunikat, a konsumentem może być dowolny komunikat.

Jeśli aplikacja wymaga współbieżnego dostarczania komunikatów, to znaczy, że wszyscy konsumenci muszą odbierać komunikaty w tym samym czasie, aplikacja musi utworzyć wiele sesji i każdy z nich musi mieć jednego asynchronicznego konsumenta komunikatów.

Poniższe przykłady bardziej wyraźnie pokazują tę funkcję.

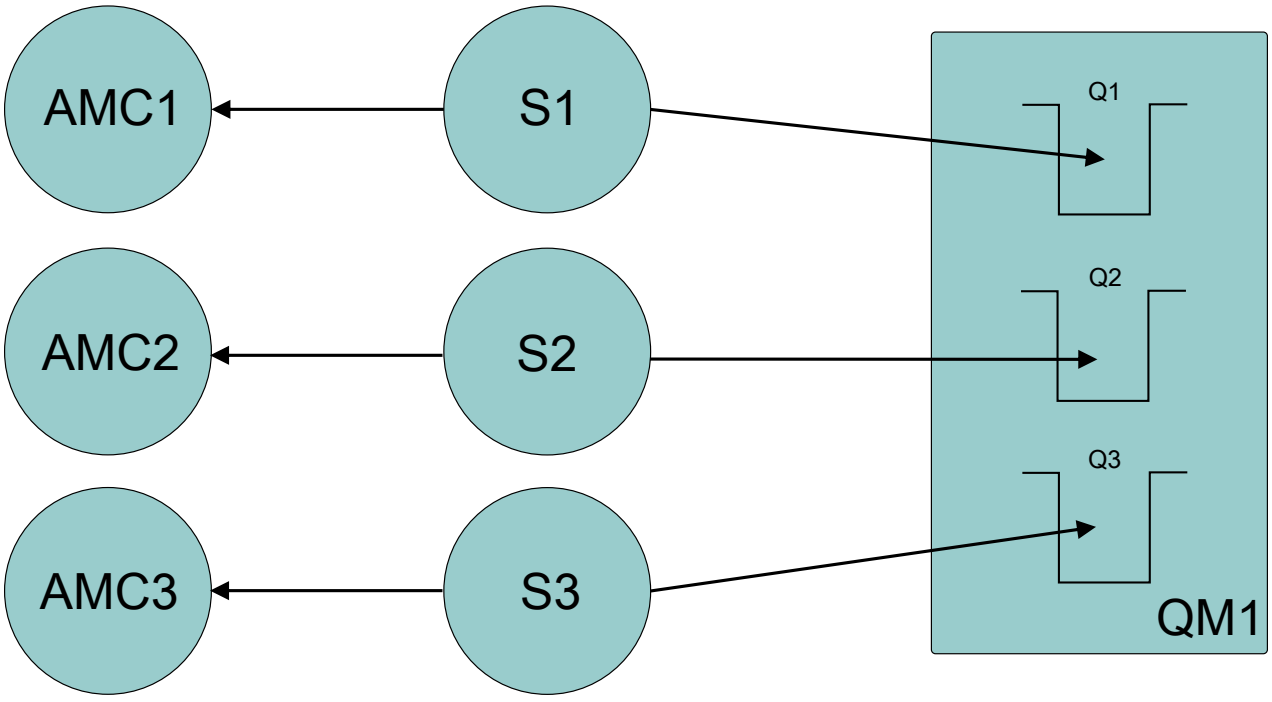
W pierwszym przykładzie w sesji istnieje wiele asynchronicznych konsumentów komunikatów. Sesja S ma trzech asynchronicznych konsumentów komunikatów: AMC1, AMC2 i AMC3 , którzy odbierają komunikaty z trzech różnych miejsc docelowych Q1, Q2 i Q3.

Ponieważ istnieje tylko jedna sesja S, istnieje tylko wątek dostarczania komunikatów, który dostarcza komunikaty do konsumentów AMC1, AMC2i AMC3. Gdy sesja dostarcza komunikat do serwera AMC1, pozostali dwaj konsumenci AMC2 i AMC3 oczekują, nawet jeśli w systemach Q2 i Q3 istnieją komunikaty gotowe do dostarczenia.




*Rysunek 54. Jedna sesja z trzema asynchronicznymi konsumentami komunikatów*

W drugim przypadku istnieje wiele sesji S1, S2i S3, z których każda ma jeden konsument komunikatów asynchronicznych `AMC1`, `AMC2`i `AMC3` . Ponieważ dla każdej sesji istnieje jeden konsument, komunikaty są dostarczane do konsumentów współbieżnie.




*Rysunek 55. Wiele sesji, każda z jednym asynchronicznym konsumentem komunikatów*

Oznacza to, że jeśli wymagane jest współbieżne dostarczanie komunikatów, wymagane jest wiele sesji.

# Konfigurowanie menedżera kolejek i brokera dla aplikacji, która nawiązuje połączenie z menedżerem kolejek produktu IBM MQ

W tej sekcji przyjęto założenie, że używana jest wersja IBM WebSphere MQ 7.0.1lub nowsza. Przed uruchomieniem aplikacji, która nawiązuje połączenie z menedżerem kolejek produktu IBM MQ , należy skonfigurować menedżer kolejek. W przypadku aplikacji publikującej/subskrybującej wymagana jest dodatkowa konfiguracja, jeśli używany jest interfejs publikowania/subskrypcji w kolejce.

## Zanim rozpoczniesz

XMS działa z systemem IBM Integration Bus lub WebSphere Message Broker 6.1 lub nowszym

Przed rozpoczęciem tej czynności wykonaj następujące kroki:

- Upewnij się, że aplikacja ma dostęp do działającego menedżera kolejek.
- Jeśli aplikacja jest aplikacją publikującą/subskrybującą i korzysta z interfejsu publikowania/subskrypcji w kolejce, należy upewnić się, że atrybut **PSMODE** w menedżerze kolejek ma wartość ENABLED .
- Upewnij się, że aplikacja używa fabryki połączeń, której właściwości są odpowiednio ustawione w celu nawiązania połączenia z menedżerem kolejek. Jeśli aplikacja jest aplikacją publikującą/subskrybującą, upewnij się, że odpowiednie właściwości fabryki połączeń zostały ustawione na potrzeby używania brokera. Więcej informacji na temat właściwości fabryki połączeń zawiera sekcja Właściwości fabryki połączeń ConnectionFactory.

## O tym zadaniu

Menedżer kolejek i broker są konfigurowane do uruchamiania aplikacji XMS w taki sam sposób, jak menedżer kolejek i interfejs publikowania/subskrypcji w kolejce na potrzeby uruchamiania aplikacji produktu IBM MQ JMS . Poniższe kroki zawierają podsumowanie czynności, które należy wykonać.

## Procedura

1. W menedżerze kolejek utwórz kolejki wymagane przez aplikację.

   Przegląd sposobu tworzenia kolejek zawiera sekcja Definiowanie kolejek.

   Jeśli aplikacja jest aplikacją publikującą/subskrybującą i korzysta z interfejsu publikowania/subskrypcji w kolejce, który wymaga dostępu do kolejek systemowych IBM MQ classes for JMS , przed utworzeniem kolejek poczekaj do kroku 4a .
2. Nadaj identyfikatorowi użytkownika powiązanemu z aplikacją uprawnienie do nawiązywania połączenia z menedżerem kolejek oraz odpowiednie uprawnienie do uzyskiwania dostępu do kolejek.

   Przegląd informacji na temat autoryzacji zawiera sekcja Zabezpieczanie. Jeśli aplikacja nawiązuje połączenie z menedżerem kolejek w trybie klienta, należy zapoznać się także z sekcją Klienty i serwery.
3. Jeśli aplikacja nawiązuje połączenie z menedżerem kolejek w trybie klienta, upewnij się, że kanał połączenia z serwerem jest zdefiniowany w menedżerze kolejek i że uruchomiono proces nasłuchujący.

   Nie trzeba wykonywać tego kroku dla każdej aplikacji, która nawiązuje połączenie z menedżerem kolejek. Jedna definicja kanału połączenia z serwerem i jeden program nasłuchujący mogą obsługiwać wszystkie aplikacje, które łączą się w trybie klienta.
4. Jeśli aplikacja jest aplikacją publikującą/subskrybującą i korzysta z interfejsu publikowania/subskrypcji w kolejce, wykonaj następujące kroki.
   a) W menedżerze kolejek utwórz kolejki systemowe IBM MQ classes for JMS , uruchamiając skrypt komend MQSC dostarczanych z produktem IBM MQ. Upewnij się, że ID użytkownika powiązany z IBM Integration Bus lub WebSphere Message Broker ma uprawnienia dostępu do kolejek.

      Informacje o tym, gdzie znaleźć skrypt i jak go uruchomić, zawiera sekcja Korzystanie z programu IBM MQ classes for Java.

Ten krok należy wykonać tylko raz dla menedżera kolejek. Ten sam zestaw kolejek systemowych IBM MQ classes for JMS może obsługiwać wszystkie aplikacje XMS i IBM MQ classes for JMS , które łączą się z menedżerem kolejek.

b) Nadaj identyfikatorowi użytkownika powiązanemu z aplikacją uprawnienie dostępu do kolejek systemu IBM MQ classes for JMS .

Informacje na temat uprawnień wymaganych przez ID użytkownika zawiera sekcja Korzystanie z programu IBM MQ classes for JMS.

c) W przypadku brokera produktu IBM Integration Bus lub WebSphere Message Brokernależy utworzyć i wdrożyć przepływ komunikatów w celu obsługi kolejki, w której aplikacje wysyłają publikowane przez siebie komunikaty.

Podstawowy przepływ komunikatów składa się z węzła przetwarzania komunikatów MQInput do odczytywania opublikowanych komunikatów oraz węzła przetwarzania komunikatów publikowania do publikowania komunikatów.

Informacje na temat tworzenia i wdrażania przepływu komunikatów zawiera dokumentacja produktu IBM Integration Bus lub WebSphere Message Broker dostępna w serwisie Strona WWW biblioteki dokumentacji produktu IBM Integration Bus.

Nie trzeba wykonywać tego kroku, jeśli odpowiedni przepływ komunikatów jest już wdrożony w brokerze.

### Wyniki

Teraz można uruchomić aplikację.

## Konfigurowanie brokera dla aplikacji, która używa połączenia w czasie rzeczywistym z brokerem

Przed uruchomieniem aplikacji, która używa połączenia w czasie rzeczywistym z brokerem, należy skonfigurować ten broker.

### Zanim rozpoczniesz

Przed rozpoczęciem tego zadania należy wykonać następujące kroki:

- Upewnij się, że aplikacja ma dostęp do działającego brokera.
- Upewnij się, że aplikacja używa fabryki połączeń, której właściwości są odpowiednio ustawione dla połączenia z brokerem w czasie rzeczywistym. Więcej informacji na temat właściwości fabryki połączeń zawiera sekcja Właściwości fabryki połączeń ConnectionFactory.

### O tym zadaniu

Broker można skonfigurować do uruchamiania aplikacji XMS w taki sam sposób, jak broker do uruchamiania aplikacji IBM MQ classes for JMS . Poniższe kroki zawierają podsumowanie czynności, które należy wykonać:

### Procedura

1. Utwórz i wdróż przepływ komunikatów, aby odczytywać komunikaty z portu TCP/IP, na którym broker nasłuchuje i publikuje komunikaty.

   Można to zrobić w jeden z następujących sposobów:

   - Utwórz przepływ komunikatów, który zawiera węzeł przetwarzania komunikatów **Real-timeOptimizedFlow** .
   - Utwórz przepływ komunikatów, który zawiera węzeł przetwarzania komunikatów **Real-timeInput** i węzeł przetwarzania komunikatów publikowania.

Należy skonfigurować węzeł **Real-timeOptimizedFlow** lub **Real-timeInput** w taki sposób, aby nasłuchiwał na porcie używanym dla połączeń w czasie rzeczywistym. W systemie XMSdomyślnym numerem portu dla połączeń w czasie rzeczywistym jest 1506.

Nie trzeba wykonywać tego kroku, jeśli odpowiedni przepływ komunikatów jest już wdrożony w brokerze.

2. Jeśli wymagane jest, aby komunikaty były dostarczane do aplikacji przy użyciu produktu IBM MQ classes for JMS, należy skonfigurować broker w taki sposób, aby włączyć rozsyłanie grupowe. Skonfiguruj tematy, dla których musi być włączone rozsyłanie grupowe, określając niezawodną jakość usługi dla tych tematów, które wymagają niezawodnego rozsyłania grupowego.
3. Jeśli podczas nawiązywania połączenia z brokerem aplikacja udostępnia identyfikator użytkownika i hasło, a broker ma uwierzytelnić aplikację przy użyciu tych informacji, należy skonfigurować serwer nazw użytkowników i broker na potrzeby prostego uwierzytelniania przy użyciu hasła typu telnet.

### Wyniki

Teraz można uruchomić aplikację.

## Konfigurowanie magistrali integracji usług dla aplikacji, która łączy się z produktem WebSphere Application Server

Przed uruchomieniem aplikacji, która nawiązuje połączenie z magistralą integracji usług produktu WebSphere Application Server service integration technologies , należy skonfigurować integrację usług w taki sam sposób, w jaki konfiguruje się magistralę integracji usług w celu uruchamiania aplikacji produktu JMS korzystających z domyślnego dostawcy przesyłania komunikatów.

### Zanim rozpoczniesz

Przed rozpoczęciem tego zadania należy wykonać następujące kroki:

- Upewnij się, że została utworzona magistrala przesyłania komunikatów i że serwer został dodany do magistrali jako element magistrali.
- Upewnij się, że aplikacja ma dostęp do magistrali integracji usług, która zawiera co najmniej jeden uruchomiony mechanizm przesyłania komunikatów.
- Jeśli operacja HTTP jest wymagana, należy zdefiniować kanał transportu przychodzącego mechanizmu przesyłania komunikatów HTTP . Domyślnie kanały dla SSL i TCP są definiowane podczas instalacji serwera.
- Upewnij się, że aplikacja używa fabryki połączeń, której właściwości są odpowiednio ustawione w celu nawiązania połączenia z magistralą integracji usług przy użyciu serwera startowego. Minimalne wymagane informacje to:
  - Punkt końcowy dostawcy, który opisuje położenie i protokół używane podczas negocjowania połączenia z serwerem przesyłania komunikatów (czyli za pośrednictwem serwera startowego). W najprostszej formie, dla serwera zainstalowanego z ustawieniami domyślnymi, punkt końcowy udostępniania może być ustawiony na nazwę hosta serwera.
  - Nazwa magistrali, przez którą są wysyłane komunikaty.

  Więcej informacji na temat właściwości fabryki połączeń zawiera sekcja Właściwości fabryki połączeń ConnectionFactory.

### O tym zadaniu

Wszystkie wymagane obszary kolejek lub tematów muszą być zdefiniowane. Domyślnie podczas instalacji serwera jest definiowany obszar tematu o nazwie Default.Topic.Space , ale jeśli wymagane są dalsze obszary tematów, należy utworzyć te obszary tematów samodzielnie. Nie ma potrzeby predefiniowania pojedynczych tematów w obszarze tematu, ponieważ serwer tworzy ich instancje dynamicznie zgodnie z wymaganiami.

Poniższe kroki zawierają podsumowanie czynności, które należy wykonać.

### Procedura

1. Utwórz kolejki wymagane przez aplikację do przesyłania komunikatów w trybie punkt z punktem.
2. Utwórz dodatkowe obszary tematów, które są wymagane przez aplikację do przesyłania komunikatów w trybie publikowania/subskrypcji.

### Wyniki

Teraz można uruchomić aplikację.

## Korzystanie z przykładowych aplikacji XMS

Przykładowe aplikacje produktu XMS .NET udostępniają przegląd wspólnych funkcji każdego interfejsu API. Można ich używać do weryfikowania instalacji i konfiguracji serwera przesyłania komunikatów oraz do tworzenia własnych aplikacji.

### O tym zadaniu

Jeśli potrzebna jest pomoc przy tworzeniu własnych aplikacji, można użyć przykładowych aplikacji jako punktu początkowego. Dla każdej aplikacji udostępniono zarówno wersję źródłową, jak i skompilowaną. Zapoznaj się z przykładowym kodem źródłowym i zidentyfikuj kluczowe kroki, które należy wykonać, aby utworzyć każdy wymagany obiekt dla aplikacji (ConnectionFactory, Connection, Session, Destination i Producer, lub Consumer), a także aby ustawić konkretne właściwości, które są wymagane do określenia sposobu działania aplikacji. Więcej informacji na ten temat zawiera "Pisanie aplikacji XMS" na stronie 653. Przykłady mogą ulec zmianie w przyszłych wersjach produktu XMS.

W poniższej tabeli przedstawiono zestawy przykładowych aplikacji (po jednej dla każdego interfejsu API), które są dostarczane z produktem XMS.

*Tabela 81. Przykładowe aplikacje dla produktu XMS .NET*

| **Nazwa próbki** | **Opis** |
|---|---|
| SampleConsumerCS | Aplikacja konsumenta komunikatów, która pobiera komunikaty z kolejki lub subskrybuje temat. |
| SampleProducerCS | Aplikacja producenta komunikatów, która generuje komunikaty do kolejki lub tematu. |
| SampleConfigCS | Aplikacja konfiguracyjna, której można użyć do utworzenia repozytorium obiektów administrowanych opartego na plikach. Aplikacja zawiera fabrykę połączeń i miejsce docelowe dla konkretnych ustawień połączenia. To administrowane repozytorium obiektów może być następnie używane z każdą przykładową aplikacją konsumenta i producenta. |

Przykłady, które obsługują te same funkcje w różnych interfejsach API, różnią się składniowo.

- Przykładowe aplikacje konsumenta i producenta komunikatów obsługują następujące funkcje:
  - Połączenia z produktem IBM MQ, IBM Integration Bus (przy użyciu połączenia z brokerem w czasie rzeczywistym) i WebSphere Application Server service integration bus
  - Wyszukiwania w administrowanym repozytorium obiektów przy użyciu początkowego interfejsu kontekstu
  - Połączenia z kolejkami (IBM MQ i WebSphere Application Server service integration bus) i tematami (IBM MQ, połączenie w czasie rzeczywistym z brokerem i WebSphere Application Server service integration bus)
  - Komunikaty bazowe, bajtowe, mapy, obiektowe, strumieniowe i tekstowe
- Przykładowa aplikacja konsumująca komunikaty obsługuje synchroniczne i asynchroniczne tryby odbierania oraz instrukcje selektora SQL.

- Przykładowa aplikacja producenta komunikatów obsługuje tryby dostarczania trwałego i nietrwałego.

Przykłady mogą działać w jednym z dwóch trybów:

**tryb prosty**
Przykłady można uruchomić z minimalną wartością wprowadzoną przez użytkownika.

**tryb zaawansowany**
Bardziej precyzyjnie można dostosować sposób działania przykładów.

Wszystkie przykłady są kompatybilne i dlatego mogą działać w różnych językach.

Windows Od wersji IBM MQ 9.1.1produkt IBM MQ obsługuje aplikacje .NET Core for XMS .NET w środowiskach Windows . Produkt IBM MQ classes for .NET Standard, w tym przykłady, jest instalowany domyślnie w ramach standardowej instalacji produktu IBM MQ .

Linux Od wersji IBM MQ 9.1.2produkt IBM MQ obsługuje również moduł podstawowy .NET dla aplikacji w środowiskach Linux .

Aplikacje przykładowe dla produktu XMS .NET są instalowane w katalogu `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/xms`.

Więcej informacji na ten temat zawiera “Instalowanie produktu IBM MQ classes for XMS .NET” na stronie 640.

## Uruchamianie przykładowych aplikacji .NET

Aplikacje przykładowe .NET można uruchomić interaktywnie w trybie prostym lub zaawansowanym albo nieinteraktywnie, używając automatycznie wygenerowanych lub dostosowanych plików odpowiedzi.

### Zanim rozpoczniesz

Przed uruchomieniem dowolnej z dostarczonych aplikacji przykładowych należy najpierw skonfigurować środowisko serwera przesyłania komunikatów, aby aplikacje mogły nawiązywać połączenia z serwerem. Patrz “Konfigurowanie środowiska serwera przesyłania komunikatów” na stronie 644.

### Procedura

Aby uruchomić przykładową aplikację .NET, wykonaj następujące kroki:

**Wskazówka:** Jeśli uruchamiasz przykładową aplikację, wpisz? w dowolnym momencie, aby uzyskać pomoc na temat tego, co zrobić dalej.

1. Wybierz tryb, w którym ma zostać uruchomiona przykładowa aplikacja.

   Wpisz `Advanced` lub `Simple`.
2. Odpowiedz na pytania.

   Aby wybrać wartość domyślną, która jest wyświetlana w nawiasach na końcu pytania, naciśnij klawisz Enter. Aby wybrać inną wartość, wpisz odpowiednią wartość i naciśnij klawisz Enter.

   Oto przykładowe pytanie:

   ```
   Enter connection type [wpm]:
   ```

   W takim przypadku wartością domyślną jest `wpm` (połączenie z WebSphere Application Server service integration bus).

### Wyniki

Po uruchomieniu przykładowych aplikacji pliki odpowiedzi są generowane automatycznie w bieżącym katalogu roboczym. Nazwy plików odpowiedzi mają format *`connection_type-sample_type`*`.rsp`, na przykład `wpm-producer.rsp`. W razie potrzeby można użyć wygenerowanego pliku odpowiedzi, aby

ponownie uruchomić przykładową aplikację z tymi samymi opcjami, dzięki czemu nie będzie konieczne ponowne wprowadzanie opcji.

**Zadania pokrewne**

Budowanie przykładowych aplikacji .NET
Podczas budowania przykładowej aplikacji .NET tworzona jest wersja wykonywalna wybranego przykładu.

Tworzenie własnych aplikacji
Użytkownik tworzy własne aplikacje, podobnie jak aplikacje przykładowe.

## Budowanie przykładowych aplikacji .NET

Podczas budowania przykładowej aplikacji .NET tworzona jest wersja wykonywalna wybranego przykładu.

### Zanim rozpoczniesz

Zainstaluj odpowiedni kompilator. W tej czynności przyjęto założenie, że zainstalowano produkt Microsoft Visual Studio 2012 i że użytkownik zna jego używanie.

### Procedura

Aby zbudować przykładową aplikację .NET , wykonaj następujące kroki:

1. Kliknij plik rozwiązania `Samples.sln` udostępniony z przykładami .NET .
2. Kliknij prawym przyciskiem myszy `Przykłady` rozwiązania w oknie Eksplorator rozwiązań i wybierz opcję **Buduj rozwiązanie**.

### Wyniki

Program wykonywalny jest tworzony w odpowiednim podfolderze przykładu, `bin/Debug` lub `bin/Release`, w zależności od wybranej konfiguracji. Ten program ma taką samą nazwę, jak folder, z przyrostkiem CS. Jeśli na przykład tworzona jest wersja C# przykładowej aplikacji producenta komunikatów, plik `SampleProducerCS.exe` jest tworzony w folderze `SampleProducer` .

**Zadania pokrewne**

Uruchamianie przykładowych aplikacji .NET
Aplikacje przykładowe .NET można uruchomić interaktywnie w trybie prostym lub zaawansowanym albo nieinteraktywnie, używając automatycznie wygenerowanych lub dostosowanych plików odpowiedzi.

Tworzenie własnych aplikacji
Użytkownik tworzy własne aplikacje, podobnie jak aplikacje przykładowe.

“Tworzenie własnych aplikacji” na stronie 652
Użytkownik tworzy własne aplikacje, podobnie jak aplikacje przykładowe.

## Tworzenie własnych aplikacji

Użytkownik tworzy własne aplikacje, podobnie jak aplikacje przykładowe.

### Zanim rozpoczniesz

Zainstaluj odpowiedni kompilator. W tej czynności przyjęto założenie, że zainstalowano produkt Microsoft Visual Studio 2012 i że użytkownik zna jego używanie.

### Procedura

- Zbuduj aplikację .NET zgodnie z opisem w sekcji “Budowanie przykładowych aplikacji .NET” na stronie 652.

  Aby uzyskać dodatkowe wskazówki dotyczące budowania własnych aplikacji, należy użyć plików makefile udostępnionych dla każdej przykładowej aplikacji.

  **Wskazówka:** Aby pomóc w diagnozowaniu problemów w przypadku awarii, pomocne może być skompilowanie aplikacji z dołączonymi symbolami.

**Zadania pokrewne**

Uruchamianie przykładowych aplikacji .NET
Aplikacje przykładowe .NET można uruchomić interaktywnie w trybie prostym lub zaawansowanym albo nieinteraktywnie, używając automatycznie wygenerowanych lub dostosowanych plików odpowiedzi.

Budowanie przykładowych aplikacji .NET
Podczas budowania przykładowej aplikacji .NET tworzona jest wersja wykonywalna wybranego przykładu.

# Pisanie aplikacji XMS

Tematy w tej sekcji zawierają informacje pomocne podczas ogólnego pisania aplikacji XMS .

## O tym zadaniu

Ta sekcja zawiera ogólne pojęcia związane z pisaniem aplikacji XMS . Więcej informacji na temat tworzenia aplikacji XMS .NET zawiera sekcja “Pisanie aplikacji XMS .NET” na stronie 672 .

W systemie IBM MQ 9.2.0jest to liczba XMS.NET zostały znacząco zredukowane do pięciu. Pięć bibliotek dołączanych dynamicznie to:

- `IBM.XMS.dll` -zawiera wszystkie komunikaty w języku narodowym
- `IBM.XMS.Comms.RMM.dll`
- Trzy biblioteki połączeń dynamicznych strategii:
  - `policy.8.0.IBM.XMS.dll`
  - `policy.9.0.IBM.XMS.dll`
  - `policy.9.1.IBM.XMS.dll`

V 9.3.0 Removed V 9.3.0 Rozsyłanie grupowe komunikatów XMS .NET (przy użyciu RMM) stało się nieaktualne od wersji IBM MQ 9.2 i zostało usunięte w wersji IBM MQ 9.3.

Sekcja obejmuje następujące tematy:

- “Model wielowątkowości” na stronie 654
- “ConnectionFactories i obiekty połączeń” na stronie 655
- “Sesje” na stronie 656
- “Miejsca docelowe” na stronie 660
- “Producenci komunikatów” na stronie 662
- “Konsumenci komunikatów” na stronie 663
- “Przeglądarki kolejek” na stronie 666
- “Żądający” na stronie 667
- “Usuwanie obiektu” na stronie 667
- “Typy podstawowe XMS” na stronie 668
- “Niejawna konwersja wartości właściwości z jednego typu danych do innego” na stronie 669
- “Iteratory” na stronie 671
- “Identyfikatory kodowanego zestawu znaków” na stronie 671
- “Kody błędów i wyjątków XMS” na stronie 671
- “Tworzenie własnych aplikacji” na stronie 652

## Windows Korzystanie z szablonu projektu produktu IBM MQ XMS .NET

Klient IBM MQ XMS .NET umożliwia korzystanie z szablonu projektu w celu ułatwienia tworzenia aplikacji XMS .NET Core .

## Zanim rozpoczniesz

W systemie musi być zainstalowany system Microsoft Visual Studio 2017lub nowszy i .NET Core 2.1 .

Należy skopiować szablon XMS .NET z

*&MQ_INSTALL_ROOT*&\tools\dotnet\samples\cs\core\xms\ProjectTemplates\IBMXMS.NETClientApp.zip

do katalogu

*&USER_HOME_DIRECTORY*&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates

katalog, gdzie:

- *& MQ_INSTALL_ROOT* to katalog główny instalacji.
- *& USER_HOME_DIRECTORY* jest katalogiem osobistym.

Aby pobrać szablon, należy zatrzymać i ponownie uruchomić program Microsoft Visual Studio .

## O tym zadaniu

Szablon projektu XMS .NET zawiera pewien wspólny kod, którego można użyć do tworzenia aplikacji. Wbudowany kod umożliwia nawiązanie połączenia z menedżerem kolejek produktu IBM MQ i wykonanie operacji umieszczania (put) lub pobierania (get) przez zmodyfikowanie właściwości w wbudowanym kodzie.

## Procedura

1. Otwórz program Microsoft Visual Studio.
2. Kliknij opcję **Plik**, a następnie opcję **Nowy** i opcję **Projekt**.
3. W oknie *Tworzenie nowego projektu*wybierz opcję `IBM XMS .NET Client App (.NET Core)` i kliknij przycisk **Dalej**.
4. W oknie *Konfigurowanie nowego projektu* zmień nazwę projektu w polu *Nazwa projektu* i kliknij przycisk **Utwórz** , aby utworzyć projekt XMS .NET .

   `XMSDotnetApp.cs` to plik, który jest tworzony razem z plikiem projektu. Ten plik zawiera kod, który nawiązuje połączenie z menedżerem kolejek i wykonuje operację wysyłania i odbierania.

   Właściwości połączenia są ustawione na wartości domyślne:

   - Parametr WMQ_CONNECTION_NAME_LIST jest ustawiony na wartość *localhost (1414)*
   - XMSC.WMQ_CHANNEL jest ustawiony na wartość *DOTNET.SVRCONN*

   Kolejka jest ustawiona na *Q1*i można odpowiednio zmodyfikować te właściwości.
5. Skompiluj i uruchom aplikację.

**Pojęcia pokrewne**

Komponenty i opcje produktu IBM MQ

Środowisko wykonawcze aplikacji .NET -tylko Windows

# Model wielowątkowości

Reguły ogólne określają, w jaki sposób aplikacja wielowątkowa może używać obiektów XMS .

- W różnych wątkach mogą być współbieżnie używane tylko obiekty następujących typów:
  - ConnectionFactory
  - Połączenie
  - Dane ConnectionMeta
  - Miejsce docelowe

- Obiekt sesji może być używany tylko w pojedynczym wątku w danym momencie.

Wyjątki od tych reguł są wskazywane przez pozycje oznaczone "Kontekst wątku" w definicjach interfejsów metod w IBM Message Service Client for .NET reference.

## ConnectionFactories i obiekty połączeń

Obiekt ConnectionFactory udostępnia szablon, który jest używany przez aplikację do tworzenia obiektu Connection. Aplikacja używa obiektu połączenia do utworzenia obiektu sesji.

W przypadku produktu .NETaplikacja XMS najpierw używa obiektu XMSFactoryFactory w celu uzyskania odwołania do obiektu ConnectionFactory , który jest odpowiedni dla wymaganego typu protokołu. Obiekt ConnectionFactory może następnie generować połączenia tylko dla tego typu protokołu.

Aplikacja XMS może tworzyć wiele połączeń, a aplikacja wielowątkowa może korzystać z pojedynczego obiektu połączenia współbieżnie w wielu wątkach. Obiekt połączenia hermetyzuje połączenie komunikacyjne między aplikacją a serwerem przesyłania komunikatów.

Połączenie służy kilku celom:

- Gdy aplikacja tworzy połączenie, można ją uwierzytelnić.
- Aplikacja może powiązać unikalny identyfikator klienta z połączeniem. Identyfikator klienta jest używany do obsługi trwałych subskrypcji w domenie publikowania/subskrypcji. Identyfikator klienta można ustawić na dwa sposoby:

  Preferowanym sposobem przypisania identyfikatora klienta połączeń jest skonfigurowanie w specyficznym dla klienta obiekcie ConnectionFactory przy użyciu właściwości i niezauważalne przypisanie go do tworzonego połączenia.

  Innym sposobem przypisania identyfikatora klienta jest użycie wartości specyficznej dla dostawcy, która jest ustawiona w obiekcie połączenia. Ta wartość nie nadpisuje identyfikatora, który został skonfigurowany administracyjnie. Jest ona podawana w przypadku, gdy nie istnieje identyfikator określony administracyjnie. Jeśli identyfikator określony administracyjnie istnieje, próba przesłonięcia go wartością specyficzną dla dostawcy spowoduje zgłoszenie wyjątku. Jeśli aplikacja jawnie ustawi identyfikator, musi to zrobić natychmiast po utworzeniu połączenia i przed wykonaniem innych działań na połączeniu. W przeciwnym razie zostanie zgłoszony wyjątek.

Aplikacja XMS zwykle tworzy połączenie, jedną lub więcej sesji oraz wiele producentów komunikatów i konsumentów komunikatów.

Tworzenie połączenia jest stosunkowo kosztowne pod względem zasobów systemowych, ponieważ wiąże się z nawiązaniem połączenia komunikacyjnego, a także z uwierzytelnieniem aplikacji.

## Tryb połączenia uruchomionego i zatrzymanego

Połączenie może działać w trybie uruchomionym lub zatrzymanym.

Gdy aplikacja tworzy połączenie, połączenie jest w trybie zatrzymania. Gdy połączenie jest w trybie zatrzymanym, aplikacja może inicjować sesje i wysyłać komunikaty, ale nie może odbierać ich synchronicznie ani asynchronicznie.

Aplikacja może uruchomić połączenie, wywołując metodę `Start Connection` . Gdy połączenie jest w trybie uruchomienia, aplikacja może wysyłać i odbierać komunikaty. Aplikacja może następnie zatrzymać i zrestartować połączenie, wywołując metody Stop Connection i `Start Connection` .

## Zamknięcie połączenia

Aplikacja zamyka połączenie, wywołując metodę Zamknij połączenie. Gdy aplikacja zamyka połączenie, program XMS wykonuje następujące działania:

- Zamyka wszystkie sesje powiązane z połączeniem i usuwa niektóre obiekty powiązane z tymi sesjami. Więcej informacji na temat usuwania obiektów zawiera sekcja “Usuwanie obiektu” na stronie 667. Jednocześnie program XMS wycofuje wszystkie transakcje, które są obecnie w toku w ramach sesji.

- Kończy on połączenie komunikacyjne z serwerem przesyłania komunikatów.
- Zwalnia pamięć i inne zasoby wewnętrzne używane przez połączenie.

Produkt XMS nie potwierdza odbioru żadnych komunikatów, których nie potwierdził podczas sesji, przed zamknięciem połączenia. Więcej informacji na temat potwierdzania odbioru komunikatów zawiera sekcja “Potwierdzenie komunikatu” na stronie 657.

## Obsługa wyjątków

Wszystkie wyjątki serwera XMS .NET są wyprowadzane z wyjątku System.Exception. Więcej informacji na ten tematzawiera sekcja “Obsługa błędów w produkcie .NET” na stronie 676.

## Połączenie z magistralą integracji usług

Aplikacja XMS może łączyć się z magistralą integracji usług WebSphere Application Server za pomocą bezpośredniego połączenia TCP/IP lub za pomocą protokołu HTTP przez TCP/IP.

Protokołu HTTP można używać w sytuacjach, gdy bezpośrednie połączenie TCP/IP nie jest możliwe. Jedną z powszechnych sytuacji jest komunikacja przez firewall, na przykład gdy dwa przedsiębiorstwa wymieniają się komunikatami. Korzystanie z programu HTTP do komunikacji przez firewall jest często nazywane *HTTP tunelowanie*. Jednak tunelowanie HTTP jest z natury wolniejsze niż bezpośrednie połączenie TCP/IP, ponieważ nagłówki HTTP znacząco uzupełniają ilość przesyłanych danych i ponieważ protokół HTTP wymaga więcej przepływów komunikacyjnych niż protokół TCP/IP.

Aby utworzyć połączenie TCP/IP, aplikacja może użyć fabryki połączeń, której właściwość XMSC_WPM_TARGET_TRANSPORT_CHAIN ma wartość XMSC_WPM_TARGET_TRANSPORT_CHAIN_BASIC. Jest to wartość domyślna właściwości. Jeśli połączenie zostało pomyślnie utworzone, właściwość XMSC_WPM_CONNECTION_PROTOCOL połączenia jest ustawiona na wartość XMSC_WPM_CP_TCP.

Aby utworzyć połączenie używające protokołu HTTP, aplikacja musi używać fabryki połączeń, której właściwość XMSC_WPM_TARGET_TRANSPORT_CHAIN jest ustawiona na nazwę łańcucha transportowego danych przychodzących, który jest skonfigurowany do używania kanału transportowego HTTP . Jeśli połączenie zostało pomyślnie utworzone, właściwość XMSC_WPM_CONNECTION_PROTOCOL połączenia jest ustawiona na wartość XMSC_WPM_CP_HTTP. Informacje na temat konfigurowania łańcuchów transportowych zawiera sekcja Konfigurowanie łańcuchów transportowych w dokumentacji produktu WebSphere Application Server .

Aplikacja ma podobny wybór protokołów komunikacyjnych podczas nawiązywania połączenia z serwerem startowym. Właściwość XMSC_WPM_PROVIDER_ENDPOINTS fabryki połączeń jest sekwencją jednego lub większej liczby adresów punktów końcowych serwerów startowych. Startowym komponentem łańcucha transportowego każdego adresu punktu końcowego może być XMSC_WPM_BOOTSTRAP_TCP dla połączenia TCP/IP z serwerem startowym lub XMSC_WPM_BOOTSTRAP_HTTP dla połączenia używającego protokołu HTTP.

## Sesje

Sesja jest jednowątkowym kontekstem do wysyłania i odbierania komunikatów.

Aplikacja może używać sesji do tworzenia komunikatów, producentów komunikatów, konsumentów komunikatów, przeglądarek kolejek i tymczasowych miejsc docelowych. Aplikacja może również używać sesji do uruchamiania transakcji lokalnych.

Aplikacja może tworzyć wiele sesji, w których każda sesja generuje i konsumuje komunikaty niezależnie od innych sesji. Jeśli dwa konsumenty komunikatów w oddzielnych sesjach (lub nawet w tej samej sesji) subskrybują ten sam temat, każdy z nich otrzymuje kopię dowolnego komunikatu opublikowanego w tym temacie.

W przeciwieństwie do obiektu połączenia, obiekt sesji nie może być używany współbieżnie w różnych wątkach. Tylko metoda Close Session obiektu Session może być wywołana z wątku innego niż ten, który jest używany w tym czasie przez obiekt Session. Metoda Zamknij sesję kończy sesję i zwalnia wszystkie zasoby systemowe przydzielone do sesji.

Jeśli aplikacja musi przetwarzać komunikaty współbieżnie w więcej niż jednym wątku, musi utworzyć sesję w każdym wątku, a następnie użyć tej sesji dla każdej operacji wysyłania lub odbierania w tym wątku.

## *Sesje transakcyjne*

Aplikacje XMS mogą uruchamiać transakcje lokalne. *Transakcja lokalna* to transakcja, która obejmuje tylko zmiany zasobów menedżera kolejek lub magistrali integracji usług, z którymi połączona jest aplikacja.

Informacje zawarte w tym temacie mają zastosowanie tylko wtedy, gdy aplikacja nawiązuje połączenie z menedżerem kolejek produktu IBM MQ lub magistralą integracji usług produktu WebSphere Application Server . Informacje te nie dotyczą połączenia z brokerem w czasie rzeczywistym.

Aby uruchomić transakcje lokalne, aplikacja musi najpierw utworzyć sesję transakcją, wywołując metodę Create Session obiektu połączenia, określając jako parametr, że sesja jest transakcją. Następnie wszystkie komunikaty wysyłane i odbierane w ramach sesji są grupowane w sekwencję transakcji. Transakcja kończy się, gdy aplikacja zatwierdza lub wycofuje wysłane i odebrane komunikaty od momentu rozpoczęcia transakcji.

Aby zatwierdzić transakcję, aplikacja wywołuje metodę zatwierdzania obiektu Session. Po zatwierdzeniu transakcji wszystkie komunikaty wysłane w ramach transakcji stają się dostępne do dostarczenia do innych aplikacji, a wszystkie komunikaty odebrane w ramach transakcji są potwierdzane, aby serwer przesyłania komunikatów nie próbował ponownie dostarczyć ich do aplikacji. W domenie typu punkt z punktem serwer przesyłania komunikatów usuwa również odebrane komunikaty z ich kolejek.

Aby wycofać transakcję, aplikacja wywołuje metodę Rollback obiektu Session. Po wycofaniu transakcji wszystkie komunikaty wysłane w ramach transakcji są usuwane przez serwer przesyłania komunikatów, a wszystkie komunikaty odebrane w ramach transakcji stają się ponownie dostępne do dostarczenia. W domenie typu punkt z punktem odebrane komunikaty są ponownie umieszczane w kolejkach i stają się ponownie widoczne dla innych aplikacji.

Nowa transakcja jest uruchamiana automatycznie, gdy aplikacja utworzy sesję transakcyjną lub wywoła metodę zatwierdzania lub wycofywania zmian. Oznacza to, że sesja transakcyjna zawsze ma aktywną transakcję.

Gdy aplikacja zamyka sesję transakcyjne, następuje niejawne wycofanie zmian. Gdy aplikacja zamyka połączenie, następuje niejawne wycofanie zmian dla wszystkich sesji transakcyjnych połączenia.

Transakcja jest w całości zawarta w sesji transakcyjnej. Transakcja nie może obejmować sesji. Oznacza to, że nie jest możliwe, aby aplikacja wysyłała i odbierała komunikaty w dwóch lub większej liczbie sesji transakcyjnych, a następnie zatwierdzała lub wycofywała wszystkie te działania jako pojedynczą transakcję.

**Pojęcia pokrewne**

Potwierdzenie komunikatu
Każda sesja, która nie jest transakcją, ma tryb potwierdzenia, który określa sposób potwierdzania komunikatów odbieranych przez aplikację. Dostępne są trzy tryby potwierdzenia, a wybór trybu potwierdzenia ma wpływ na projekt aplikacji.

Dostarczanie komunikatów
Produkt XMS obsługuje trwałe i nietrwałe tryby dostarczania komunikatów oraz asynchroniczne i synchroniczne dostarczanie komunikatów.

## *Potwierdzenie komunikatu*

Każda sesja, która nie jest transakcją, ma tryb potwierdzenia, który określa sposób potwierdzania komunikatów odbieranych przez aplikację. Dostępne są trzy tryby potwierdzenia, a wybór trybu potwierdzenia ma wpływ na projekt aplikacji.

Informacje zawarte w tym temacie mają zastosowanie tylko wtedy, gdy aplikacja nawiązuje połączenie z menedżerem kolejek produktu IBM MQ lub magistralą integracji usług produktu WebSphere Application Server . Informacje te nie dotyczą połączenia z brokerem w czasie rzeczywistym.

Produkt XMS używa tego samego mechanizmu do potwierdzania odbioru komunikatów, który jest używany przez usługę JMS.

Jeśli sesja nie jest transakcją, sposób potwierdzania komunikatów odbieranych przez aplikację jest określany przez tryb potwierdzenia sesji. Trzy tryby potwierdzenia są opisane w następujących akapitach:

**XMSC_AUTO_ACKNOWLEDGE**
Sesja automatycznie potwierdza każdy komunikat odebrany przez aplikację.

Jeśli komunikaty są dostarczane synchronicznie do aplikacji, sesja potwierdza odebranie komunikatu za każdym razem, gdy wywołanie odbioru zakończy się pomyślnie.

Jeśli aplikacja otrzyma komunikat pomyślnie, ale niepowodzenie uniemożliwi potwierdzenie, komunikat ponownie stanie się dostępny do dostarczenia. Dlatego aplikacja musi być w stanie obsłużyć komunikat, który został ponownie dostarczony.

**XMSC_DUPS_OK_ACKNOWLEDGE**
Sesja potwierdza komunikaty odebrane przez aplikację w wybranych momentach.

Użycie tego trybu potwierdzenia zmniejsza ilość pracy, jaką musi wykonać sesja, ale niepowodzenie, które uniemożliwia potwierdzenie komunikatu, może spowodować ponowne udostępnienie więcej niż jednego komunikatu do dostarczenia. Z tego powodu aplikacja musi być w stanie obsługiwać ponownie dostarczane komunikaty.

**POTWIERDZENIE_KLIENTA_XMSC_**
Aplikacja potwierdza odebrane komunikaty, wywołując metodę Acknowledge klasy Message.

Aplikacja może potwierdzić odbiór każdego komunikatu osobno lub może odebrać partię komunikatów i wywołać metodę potwierdzania tylko dla ostatniego komunikatu, który otrzymuje. Po wywołaniu metody Acknowledge wszystkie komunikaty odebrane od czasu ostatniego wywołania metody są potwierdzane.

W połączeniu z dowolnym z tych trybów potwierdzania aplikacja może zatrzymać i zrestartować dostarczanie komunikatów w sesji, wywołując metodę odtwarzania klasy Session. Komunikaty, dla których wcześniej nie potwierdzono odbioru, są ponownie dostarczane. Mogą one jednak nie być dostarczane w tej samej kolejności, w jakiej zostały wcześniej dostarczone. W międzyczasie mogły pojawić się komunikaty o wyższym priorytecie, a niektóre z oryginalnych komunikatów mogły utracić ważność. W domenie typu punkt z punktem niektóre oryginalne komunikaty mogły zostać wykorzystane przez inną aplikację.

Aplikacja może określić, czy komunikat jest ponownie dostarczany, sprawdzając treść pola nagłówka JMSRedelivered komunikatu. W tym celu aplikacja wywołuje metodę Get JMSRedelivered klasy Message.

**Pojęcia pokrewne**

Sesje transakcyjne
Aplikacje XMS mogą uruchamiać transakcje lokalne. *Transakcja lokalna* to transakcja, która obejmuje tylko zmiany zasobów menedżera kolejek lub magistrali integracji usług, z którymi połączona jest aplikacja.

Dostarczanie komunikatów
Produkt XMS obsługuje trwałe i nietrwałe tryby dostarczania komunikatów oraz asynchroniczne i synchroniczne dostarczanie komunikatów.

## *Dostarczanie komunikatów*

Produkt XMS obsługuje trwałe i nietrwałe tryby dostarczania komunikatów oraz asynchroniczne i synchroniczne dostarczanie komunikatów.

## Tryb dostarczania komunikatów

Produkt XMS obsługuje dwa tryby dostarczania komunikatów:

**Trwałe**
Komunikaty trwałe są dostarczane jednorazowo. Serwer przesyłania komunikatów podejmuje specjalne środki ostrożności, takie jak rejestrowanie komunikatów, aby zapewnić, że trwałe komunikaty nie zostaną utracone podczas przesyłania, nawet w przypadku awarii.

**Nietrwałe**
Nietrwałe komunikaty są dostarczane nie więcej niż raz. Komunikaty nietrwałe są mniej niezawodne niż komunikaty trwałe, ponieważ mogą zostać utracone podczas przesyłania w przypadku awarii.

Wybór trybu dostarczania jest kompromisem między niezawodnością a wydajnością. Komunikaty nietrwałe są zwykle transportowane szybciej niż komunikaty trwałe.

## Asynchroniczne dostarczanie komunikatów

Program XMS używa jednego wątku do obsługi wszystkich asynchronicznych dostarczeń komunikatów dla sesji. Oznacza to, że w danym momencie może być uruchomiona tylko jedna funkcja nasłuchiwania komunikatów lub jedna metoda `onMessage()`.

Jeśli więcej niż jeden konsument komunikatów w sesji odbiera komunikaty asynchronicznie, a funkcja nasłuchiwania komunikatów lub metoda `onMessage()` dostarcza komunikat do konsumenta komunikatów, każdy inny konsument komunikatów oczekujący na ten sam komunikat musi kontynuować oczekiwanie. Inne komunikaty, które oczekują na dostarczenie do sesji, muszą również kontynuować oczekiwanie.

Jeśli aplikacja wymaga współbieżnego dostarczania komunikatów, należy utworzyć więcej niż jedną sesję, aby produkt XMS używał więcej niż jednego wątku do obsługi asynchronicznego dostarczania komunikatów. W ten sposób można współbieżnie uruchomić więcej niż jedną funkcję nasłuchiwania komunikatów lub więcej niż jedną metodę `onMessage()`.

Sesja nie jest asynchroniczna przez przypisanie obiektu nasłuchiwania komunikatów do konsumenta. Sesja staje się asynchroniczna tylko wtedy, gdy zostanie wywołana metoda `Connection.Start`. Wszystkie wywołania synchroniczne są dozwolone do momentu wywołania metody `Connection.Start`. Dostarczanie komunikatów do konsumentów jest uruchamiane po wywołaniu `Connection.Start`.

Jeśli wywołania synchroniczne, takie jak utworzenie konsumenta lub producenta, muszą być wykonywane w sesji asynchronicznej, należy wywołać funkcję `Connection.Stop`. Sesję można wznowić, wywołując metodę `Connection.Start` w celu rozpoczęcia dostarczania komunikatów. Jedynym wyjątkiem jest wątek dostarczania komunikatów sesji, który dostarcza komunikaty do funkcji zwrotnej. Ten wątek może wykonać dowolne wywołanie w sesji (z wyjątkiem wywołania Close) w funkcji wywołania zwrotnego komunikatu.

**Uwaga:** W trybie niezarządzanym wywołanie MQDISC w funkcji wywołania zwrotnego nie jest obsługiwane przez klient IBM MQ .NET. Dlatego aplikacja kliencka nie może tworzyć ani zamykać sesji w ramach wywołania zwrotnego MessageListener w trybie odbierania asynchronicznego. Utwórz i usuń sesję poza metodą `MessageListener`.

## Synchroniczne dostarczanie komunikatów

Komunikaty są dostarczane synchronicznie do aplikacji, jeśli aplikacja używa metod odbierania obiektów MessageConsumer.

Za pomocą metod Receive aplikacja może oczekiwać przez określony czas na komunikat lub może oczekiwać w nieskończoność. Alternatywnie, jeśli aplikacja nie chce czekać na komunikat, może użyć metody Receive z opcją No Wait.

**Pojęcia pokrewne**

Sesje transakcyjne

Aplikacje XMS mogą uruchamiać transakcje lokalne. *Transakcja lokalna* to transakcja, która obejmuje tylko zmiany zasobów menedżera kolejek lub magistrali integracji usług, z którymi połączona jest aplikacja.

Potwierdzenie komunikatu

Każda sesja, która nie jest transakcją, ma tryb potwierdzenia, który określa sposób potwierdzania komunikatów odbieranych przez aplikację. Dostępne są trzy tryby potwierdzenia, a wybór trybu potwierdzenia ma wpływ na projekt aplikacji.

## Miejsca docelowe

Aplikacja XMS używa obiektu docelowego do określenia miejsca docelowego wysyłanych komunikatów oraz źródła odbieranych komunikatów.

Aplikacja XMS może utworzyć obiekt docelowy w czasie wykonywania lub uzyskać predefiniowane miejsce docelowe z repozytorium obiektów administrowanych.

Podobnie jak w przypadku fabryki ConnectionFactory, najbardziej elastycznym sposobem określenia miejsca docelowego przez aplikację XMS jest zdefiniowanie go jako obiektu administrowanego. Dzięki temu aplikacje napisane w językach C, C + +, .NET i Javamogą współużytkować definicje miejsca docelowego. Właściwości administrowanych obiektów docelowych można zmieniać bez zmiany kodu.

W przypadku aplikacji .NET miejsce docelowe jest tworzone za pomocą metody CreateTopic lub CreateQueue . Te dwie metody są dostępne zarówno w obiekcie ISession, jak i w obiekcie XMSFactoryFactory w interfejsie API języka .NET . Więcej informacji na ten temat zawierają sekcje “Miejsca docelowe w produkcie .NET” na stronie 674 i ../refdev/sapidest.dita#sapidest.

### *Identyfikatory URI tematu*

Identyfikator URI (uniform resource identifier) tematu określa nazwę tematu; może również określać jedną lub więcej jego właściwości.

Identyfikator URI tematu rozpoczyna się od tematu sekwencji: //, po którym następuje nazwa tematu i (opcjonalnie) lista par nazwa-wartość, które ustawiają pozostałe właściwości tematu. Nazwa tematu nie może być pusta.

Poniżej przedstawiono przykład we fragmencie kodu .NET :

```
topic = session.CreateTopic("topic://Sport/Football/Results?multicast=7");
```

Więcej informacji na temat właściwości tematu, w tym nazwy i poprawnych wartości, których można użyć w identyfikatorze URI, zawiera sekcja Właściwości miejsca docelowego.

Podczas określania identyfikatora URI tematu, który ma być używany w subskrypcji, można używać znaków wieloznacznych. Składnia tych znaków wieloznacznych zależy od typu połączenia i wersji brokera. Dostępna jest następująca opcja:

- WebSphere Application Server magistrala integracji usług

## WebSphere Application Server magistrala integracji usług

Magistrala integracji usług WebSphere Application Server używa następujących znaków wieloznacznych:

* w celu dopasowania dowolnych znaków na jednym poziomie w hierarchii
// zgodność z 0 lub większą liczbą poziomów
//. zgodność z 0 lub większą liczbą poziomów (na końcu wyrażenia tematu)

Tabela 82 na stronie 661 zawiera kilka przykładów użycia tego schematu znaków wieloznacznych.

*Tabela 82. Przykładowe identyfikatory URI używające schematu ze znakami wieloznacznymi dla magistrali integracji usług WebSphere Application Server*

| **Identyfikator URI (Uniform Resource Identifier)** | **Zgodne** | **Przykłady** |
|---|---|---|
| "topic://Sport/ * ball/ Results" | Wszystkie tematy z jedną hierarchiczną nazwą poziomu kończącą się na "piłka" między Sport i Wyniki | "topic://Sport/Football/Results" i "topic:// Sport/Netball/Results" |
| "topic://Sport//Wyniki" | Wszystkie tematy rozpoczynające się od "Sport/" i kończące się na "/Results" | "topic://Sport/Football/Results" i "topic:// Sport/Hockey/National/Div3/Results" |
| "topic://Sport/ Football//." | Wszystkie tematy zaczynające się na "Sport/Football/" | "topic://Sport/Football/Results" i "topic://Sport/Football/TeamNews/ Signings/Managerial" |
| "topic://Sport/ * ball// Results//." | Tematy | "topic://Sport/Football/Results" i "topic://Sport/Netball/National/Div3/ Results/2002/November" |

**Pojęcia pokrewne**

Identyfikatory URI kolejki
Identyfikator URI kolejki określa nazwę kolejki; może również określać jedną lub więcej właściwości kolejki.

Tymczasowe miejsca docelowe
Aplikacje XMS mogą tworzyć tymczasowe miejsca docelowe i korzystać z nich.

## *Identyfikatory URI kolejki*

Identyfikator URI kolejki określa nazwę kolejki; może również określać jedną lub więcej właściwości kolejki.

Identyfikator URI kolejki rozpoczyna się od sekwencji `queue://`, po której następuje nazwa kolejki. Może ona także zawierać listę par nazwa-wartość, które ustawiają pozostałe właściwości kolejki.

W przypadku kolejek systemu IBM MQ (ale nie w przypadku kolejek domyślnego dostawcy przesyłania komunikatów produktu WebSphere Application Server ) menedżer kolejek, w którym znajduje się kolejka, może zostać określony przed kolejką, przy czym nazwa menedżera kolejek może być oddzielona od nazwy kolejki.

Jeśli określono menedżer kolejek, musi to być menedżer, z którym program XMS jest bezpośrednio połączony dla połączenia używającego tej kolejki, lub menedżer kolejek musi być dostępny z tej kolejki. Zdalne menedżery kolejek są obsługiwane tylko w przypadku pobierania komunikatów z kolejek, a nie w przypadku umieszczania komunikatów w kolejkach. Szczegółowe informacje na ten temat zawiera dokumentacja menedżera kolejek systemu IBM MQ .

Jeśli nie określono żadnego menedżera kolejek, dodatkowy/separator jest opcjonalny, a jego obecność lub nieobecność nie ma znaczenia dla definicji kolejki.

Wszystkie poniższe definicje kolejek są równoważne dla kolejki produktu IBM MQ o nazwie QB w menedżerze kolejek o nazwie QM_A, z którym jest bezpośrednio połączony program XMS :

```
queue://QB
queue:///QB
queue://QM_A/QB
```

**Pojęcia pokrewne**

Identyfikatory URI tematu
Identyfikator URI (uniform resource identifier) tematu określa nazwę tematu; może również określać jedną lub więcej jego właściwości.

Tymczasowe miejsca docelowe
Aplikacje XMS mogą tworzyć tymczasowe miejsca docelowe i korzystać z nich.

### *Tymczasowe miejsca docelowe*
Aplikacje XMS mogą tworzyć tymczasowe miejsca docelowe i korzystać z nich.

Aplikacja zwykle używa tymczasowego miejsca docelowego do odbierania odpowiedzi na komunikaty żądań. Aby określić miejsce docelowe, do którego ma zostać wysłana odpowiedź na komunikat żądania, aplikacja wywołuje metodę Set JMSReplyTo obiektu Message reprezentującego komunikat żądania. Miejsce docelowe określone w wywołaniu może być tymczasowym miejscem docelowym.

Chociaż sesja jest używana do utworzenia tymczasowego miejsca docelowego, zasięg tymczasowego miejsca docelowego jest w rzeczywistości połączeniem, które zostało użyte do utworzenia sesji. Każda sesja połączenia może utworzyć producentów komunikatów i konsumentów komunikatów dla tymczasowego miejsca docelowego. Tymczasowe miejsce docelowe pozostaje do momentu jawnego usunięcia lub zakończenia połączenia, w zależności od tego, co nastąpi wcześniej.

Gdy aplikacja tworzy kolejkę tymczasową, jest ona tworzona na serwerze przesyłania komunikatów, z którym aplikacja jest połączona. Jeśli aplikacja jest połączona z menedżerem kolejek, kolejka dynamiczna jest tworzona na podstawie kolejki modelowej, której nazwa jest określona przez właściwość XMSC_WMQ_TEMPORARY_MODEL , a przedrostek używany do tworzenia nazwy kolejki dynamicznej jest określany przez właściwość XMSC_WMQ_TEMP_Q_PREFIX . Jeśli aplikacja jest połączona z magistralą integracji usług, na magistrali tworzona jest kolejka tymczasowa, a przedrostek używany do tworzenia nazwy kolejki tymczasowej jest określany przez właściwość XMSC_WPM_TEMP_Q_PREFIX .

Gdy aplikacja połączona z magistralą integracji usług tworzy temat tymczasowy, przedrostek używany do tworzenia nazwy tematu tymczasowego jest określany przez właściwość XMSC_WPM_TEMP_TOPIC_PREFIX .

**Pojęcia pokrewne**
Identyfikatory URI tematu
Identyfikator URI (uniform resource identifier) tematu określa nazwę tematu; może również określać jedną lub więcej jego właściwości.

Identyfikatory URI kolejki
Identyfikator URI kolejki określa nazwę kolejki; może również określać jedną lub więcej właściwości kolejki.

## Producenci komunikatów

W produkcie XMSproducent komunikatów może zostać utworzony z poprawnym miejscem docelowym lub bez powiązanego miejsca docelowego. Podczas tworzenia producenta komunikatów z miejscem docelowym o wartości NULL podczas wysyłania komunikatu należy określić poprawne miejsce docelowe.

### Producenci komunikatów z powiązanym miejscem docelowym

W tym scenariuszu producent komunikatów jest tworzony przy użyciu poprawnego miejsca docelowego. Podczas operacji wysyłania nie trzeba określać miejsca docelowego.

### Producenci komunikatów bez powiązanego miejsca docelowego

W produkcie XMS .NETmożna utworzyć producenta komunikatów z miejscem docelowym o wartości NULL.

Aby utworzyć producenta komunikatów bez powiązanego miejsca docelowego podczas korzystania z interfejsu API języka .NET , należy przekazać wartość NULL jako parametr do metody `CreateProducer()` obiektu ISession (na przykład `session.CreateProducer(null)`). Podczas wysyłania komunikatu należy jednak określić poprawne miejsce docelowe.

## Konsumenci komunikatów

Konsumenci komunikatów mogą być klasyfikowani jako trwałe i nietrwałe subskrybenty oraz synchroniczni i asynchroniczni konsumenci komunikatów.

### *Trwali subskrybenci*

Trwały subskrybent jest konsumentem komunikatów, który odbiera wszystkie komunikaty publikowane w temacie, w tym komunikaty publikowane, gdy subskrybent jest nieaktywny.

Informacje zawarte w tym temacie mają zastosowanie tylko wtedy, gdy aplikacja nawiązuje połączenie z menedżerem kolejek produktu IBM MQ lub magistralą integracji usług produktu WebSphere Application Server . Informacje te nie dotyczą połączenia z brokerem w czasie rzeczywistym.

Aby utworzyć trwałego subskrybenta dla tematu, aplikacja wywołuje metodę tworzenia trwałego subskrybenta obiektu sesji, określając jako parametry nazwę identyfikującą trwałą subskrypcję i obiekt docelowy reprezentujący temat. Aplikacja może utworzyć trwały subskrybent z selektorem komunikatów lub bez niego oraz określić, czy trwały subskrybent ma odbierać komunikaty publikowane przez własne połączenie.

Sesja używana do tworzenia trwałego subskrybenta musi mieć powiązany identyfikator klienta. Identyfikator klienta jest taki sam, jak identyfikator powiązany z połączeniem używanym do tworzenia sesji. Jest on określany zgodnie z opisem w sekcji “ConnectionFactories i obiekty połączeń” na stronie 655.

Nazwa identyfikująca trwałą subskrypcję musi być unikalna w obrębie identyfikatora klienta, dlatego identyfikator klienta stanowi część pełnego, unikalnego identyfikatora trwałej subskrypcji. Serwer przesyłania komunikatów przechowuje zapis trwałej subskrypcji i zapewnia, że wszystkie komunikaty publikowane w temacie są zachowywane do momentu potwierdzenia przez trwałego subskrybenta lub utraty przez niego ważności.

Serwer przesyłania komunikatów kontynuuje rejestrowanie trwałej subskrypcji nawet po zamknięciu trwałego subskrybenta. Aby ponownie wykorzystać utworzoną wcześniej subskrypcję trwałą, aplikacja musi utworzyć subskrybent trwały, podając tę samą nazwę subskrypcji i używając sesji z tym samym identyfikatorem klienta, co sesje powiązane z subskrypcją trwałą. Tylko jedna sesja w danym momencie może mieć trwałego subskrybenta dla konkretnej trwałej subskrypcji.

Zasięgiem subskrypcji trwałej jest serwer przesyłania komunikatów, który obsługuje zapis subskrypcji. Jeśli dwie aplikacje połączone z różnymi serwerami przesyłania komunikatów tworzą trwałego subskrybenta przy użyciu tej samej nazwy subskrypcji i tego samego identyfikatora klienta, tworzone są dwie całkowicie niezależne trwałe subskrypcje.

Aby usunąć trwałą subskrypcję, aplikacja wywołuje metodę anulowania subskrypcji obiektu sesji, określając jako parametr nazwę identyfikującą trwałą subskrypcję. Identyfikator klienta powiązany z sesją musi być taki sam, jak identyfikator powiązany z subskrypcją trwałą. Serwer przesyłania komunikatów usuwa rekord trwałej subskrypcji, która jest przez niego utrzymywana, i nie wysyła więcej komunikatów do trwałego subskrybenta.

Aby zmienić istniejącą subskrypcję, aplikacja może utworzyć trwały subskrybent przy użyciu tej samej nazwy subskrypcji i tego samego identyfikatora klienta, ale przy użyciu innego tematu i/lub selektora komunikatów. Zmiana subskrypcji trwałej jest równoznaczna z usunięciem subskrypcji i utworzeniem nowej.

W przypadku aplikacji, która nawiązuje połączenie z IBM MQ menedżerem kolejek, program XMS zarządza kolejkami subskrybentów. Dlatego aplikacja nie jest wymagana do określenia kolejki subskrybenta. Produkt XMS zignoruje kolejkę subskrybenta, jeśli została określona.

Należy zauważyć, że nie można zmienić kolejki subskrybenta dla subskrypcji trwałej. Jedynym sposobem na zmianę kolejki subskrybenta jest usunięcie subskrypcji i utworzenie nowej.

W przypadku aplikacji, która łączy się z magistralą integracji usług, każdy trwały subskrybent musi mieć wyznaczony punkt subskrypcji trwałej. Aby określić punkt subskrypcji trwałej dla wszystkich trwałych subskrybentów używających tego samego połączenia, należy ustawić właściwość XMSC_WPM_DUR_SUB_HOME obiektu ConnectionFactory używanego do tworzenia połączenia.

Aby określić punkt subskrypcji trwałej dla pojedynczego tematu, należy ustawić właściwość XMSC_WPM_DUR_SUB_HOME obiektu Destination reprezentującego temat. Aby aplikacja mogła utworzyć trwały subskrybent korzystający z połączenia, należy określić punkt subskrypcji trwałej dla połączenia. Każda wartość określona dla miejsca docelowego nadpisuje wartość określoną dla połączenia.

### *Synchroniczni i asynchroniczni konsumenci komunikatów*

Synchroniczny konsument komunikatów odbiera komunikaty z kolejki synchronicznie, a asynchroniczny konsument komunikatów odbiera komunikaty z kolejki asynchronicznie.

## Synchroniczne konsumenty komunikatów

Synchroniczny konsument komunikatów odbiera jeden komunikat naraz. Jeśli używana jest metoda `Receive(wait interval)` , wywołanie oczekuje na komunikat tylko przez określony czas (w milisekundach) lub do momentu zamknięcia konsumenta komunikatów.

Jeśli używana jest metoda ReceiveNoWait (), synchroniczny konsument komunikatów odbiera komunikaty bez żadnego opóźnienia. Jeśli następny komunikat jest dostępny, jest on odbierany natychmiast, w przeciwnym razie zwracany jest wskaźnik do obiektu Message o wartości NULL.

## Asynchroniczne konsumenty komunikatów

Obiekt nasłuchiwania komunikatów zarejestrowany przez aplikację jest wywoływany za każdym razem, gdy w kolejce dostępny jest nowy komunikat.

### *Komunikaty nieprzetwarzalne w produkcie XMS*

Komunikat nieprzetwarzalny to komunikat, który nie może zostać przetworzony przez odbierającą aplikację MDB. W przypadku napotkania komunikatu nieprzetwarzalnego obiekt XMS MessageConsumer może ponownie umieścić go w kolejce zgodnie z dwiema właściwościami kolejki: BOQUEUE i BOTHRESH.

W pewnych okolicznościach komunikat dostarczony do komponentu MDB może zostać wycofany do kolejki IBM MQ . Taka sytuacja może wystąpić na przykład wtedy, gdy komunikat jest dostarczany w ramach jednostki pracy, która jest następnie wycofywana. Wycofany komunikat jest zazwyczaj ponownie dostarczany, ale niepoprawnie sformatowany komunikat może wielokrotnie powodować niepowodzenie komponentu MDB i dlatego nie można go dostarczyć. Taka wiadomość jest nazywana wiadomością nieprzetwarzaną. Produkt IBM MQ można skonfigurować w taki sposób, aby komunikat nieprzetwarzalny był automatycznie przesyłany do innej kolejki w celu dalszego badania lub odrzucany. Informacje na temat konfigurowania produktu IBM MQ w ten sposób zawiera sekcja Obsługa komunikatów nieprzetwarzalnych w narzędziu ASF.

Czasami w kolejce pojawia się niepoprawnie sformatowany komunikat. W tym kontekście niepoprawnie sformatowany oznacza, że aplikacja odbierająca nie może poprawnie przetworzyć komunikatu. Taki komunikat może spowodować, że aplikacja odbierająca zakończy działanie niepowodzeniem i wycofa ten źle sformatowany komunikat. Komunikat może być następnie wielokrotnie dostarczany do kolejki wejściowej i wielokrotnie wycofywany przez aplikację. Te komunikaty są nazywane komunikatami nieprzetwarzanymi. Obiekt XMS MessageConsumer wykrywa komunikaty nieprzetwarzalne i przekierowuje je do alternatywnego miejsca docelowego.

Menedżer kolejek systemu IBM MQ rejestruje liczbę wycofał każdy komunikat. Gdy ta liczba osiągnie konfigurowalną wartość progową, konsument komunikatów ponownie wyświetla komunikat w nazwanej kolejce wycofania. Jeśli ponowne umieszczenie komunikatu w kolejce nie powiedzie się z jakiegokolwiek powodu, komunikat zostanie usunięty z kolejki wejściowej i ponownie umieszczony w kolejce niedostarczonych komunikatów lub odrzucony.

Obiekty XMS ConnectionConsumer obsługują komunikaty nieprzetwarzalne w ten sam sposób i przy użyciu tych samych właściwości kolejki. Jeśli wiele konsumentów połączeń monitoruje tę samą kolejkę, możliwe, że komunikat nieprzetwarzalny może zostać dostarczony do aplikacji więcej razy niż wartość progowa przed wystąpieniem ponownie w kolejce. Jest to spowodowane sposobem, w jaki konsumenci połączeń monitorują kolejki i komunikaty nieprzetwarzalne w kolejce.

Wartość progowa i nazwa kolejki wycofanych komunikatów są atrybutami kolejki produktu IBM MQ . Nazwy atrybutów to BackoutThreshold i BackoutRequeueQName. Kolejka, do której mają zastosowanie, jest następująca:

- W przypadku przesyłania komunikatów w trybie punkt z punktem jest to bazowa kolejka lokalna. Jest to ważne, gdy konsumenci komunikatów i konsumenci połączeń używają aliasów kolejek.
- W przypadku przesyłania komunikatów w trybie publikowania/subskrypcji w trybie normalnym dostawcy usługi przesyłania komunikatów produktu IBM MQ jest to kolejka modelowa, na podstawie której jest tworzona zarządzana kolejka tematu.
- W przypadku przesyłania komunikatów w trybie publikowania i subskrypcji w trybie migracji dostawcy przesyłania komunikatów produktu IBM MQ jest to kolejka CCSUB zdefiniowana w obiekcie fabryki TopicConnectionlub kolejka CCDSUB zdefiniowana w obiekcie tematu.

Aby ustawić atrybuty BackoutThreshold i BackoutRequeueQName, wprowadź następującą komendę MQSC:

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

W przypadku przesyłania komunikatów w trybie publikowania/subskrypcji, jeśli system tworzy kolejkę dynamiczną dla każdej subskrypcji, wartości tych atrybutów są uzyskiwane z kolejki modelowej IBM MQ classes for JMS ( SYSTEM.JMS.MODEL.QUEUE. Aby zmienić te ustawienia, należy użyć komendy:

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

Jeśli wartość progowa wycofania wynosi zero, obsługa komunikatów nieprzetwarzalnych jest wyłączona, a komunikaty nieprzetwarzalne pozostają w kolejce wejściowej. W przeciwnym razie, gdy liczba wycofanych komunikatów osiągnie wartość progową, komunikat zostanie wysłany do kolejki wycofanych komunikatów o podanej nazwie.

Jeśli liczba wycofanych komunikatów osiągnie wartość progową, ale komunikat nie może przejść do kolejki wycofanych komunikatów, komunikat jest wysyłany do kolejki niedostarczonych komunikatów lub, jeśli komunikat jest nietrwały, jest odrzucany.

Ta sytuacja występuje, jeśli kolejka wycofania nie jest zdefiniowana lub jeśli obiekt MessageConsumer nie może wysłać komunikatu do kolejki wycofania.

## Konfigurowanie systemu do obsługi komunikatów nieprzetwarzalnych

Kolejka używana przez produkt XMS .NET podczas uzyskiwania informacji o atrybutach **BOTHRESH** i **BOQNAME** zależy od stylu wykonywanego przesyłania komunikatów:

- W przypadku przesyłania komunikatów w trybie punkt z punktem jest to bazowa kolejka lokalna. Jest to ważne, gdy aplikacja XMS .NET konsumuje komunikaty z kolejek aliasowych lub kolejek klastra.
- W przypadku przesyłania komunikatów w trybie publikowania i subskrypcji tworzona jest kolejka zarządzana, w której przechowywane są komunikaty dla aplikacji. XMS .NET wysyła zapytanie do kolejki zarządzanej w celu określenia wartości atrybutów **BOTHRESH** i **BOQNAME** .

  Kolejka zarządzana jest tworzona na podstawie kolejki modelowej powiązanej z obiektem tematu, który został zasubskrybowany przez aplikację, i dziedziczy wartości atrybutów **BOTHRESH** i **BOQNAME** z kolejki modelowej. Używana kolejka modelowa zależy od tego, czy aplikacja odbierająca odebrała trwałą, czy nietrwałą subskrypcję:

  – Kolejka modelowa używana na potrzeby trwałych subskrypcji jest określona przez atrybut **MDURMDL** tematu. Wartością domyślną tego atrybutu jest SYSTEM.DURABLE.MODEL.QUEUE.
  – W przypadku subskrypcji nietrwałych używana kolejka modelowa jest określona przez atrybut **MNDURMDL** . Wartością domyślną atrybutu **MNDURMDL** jest SYSTEM.NDURABLE.MODEL.QUEUE.

Podczas uzyskiwania informacji o atrybutach **BOTHRESH** i **BOQNAME** XMS .NET:

- Otwiera kolejkę lokalną lub kolejkę docelową dla kolejki aliasowej.
- Pyta o atrybuty **BOTHRESH** i **BOQNAME** .
- Zamyka kolejkę lokalną lub kolejkę docelową dla kolejki aliasowej.

Opcje otwierania używane podczas otwierania kolejki lokalnej lub kolejki docelowej dla kolejki aliasowej zależą od używanej wersji programu IBM MQ :

- W przypadku produktu IBM MQ 9.1.0 Fix Pack 4 Long Term Support i wcześniejszych oraz produktu IBM MQ 9.1.4 Continuous Delivery i starszych: jeśli kolejka lokalna lub kolejka docelowa dla kolejki aliasowej jest kolejką klastrową, produkt XMS .NET otwiera kolejkę z opcjami `MQOO_INPUT_AS_Q_DEF`, `MQOO_INQUIRE` i `MQOO_FAIL_IF_QUIESCING` . Oznacza to, że użytkownik uruchamiający aplikację odbierającą musi mieć dostęp do zapytań i uzyskiwania dostępu do lokalnej instancji kolejki klastra.

  Program XMS .NET otwiera wszystkie inne typy kolejek lokalnych z opcjami otwarcia `MQOO_INQUIRE` i `MQOO_FAIL_IF_QUIESCING`. Aby program XMS .NET mógł wysyłać zapytania o wartości atrybutów, użytkownik uruchamiający aplikację odbierającą musi mieć dostęp do zapytań w kolejce lokalnej.
- W przypadku korzystania z produktu XMS .NET z produktu IBM MQ 9.1.5 i produktu IBM MQ 9.1.0 Fix Pack 5użytkownik uruchamiający aplikację odbierającą musi mieć dostęp do zapytań w kolejce lokalnej, niezależnie od typu kolejki.

Aby przenieść komunikaty nieprzetwarzalne do kolejki wycofanych komunikatów lub kolejki niedostarczonych komunikatów menedżera kolejek, należy nadać użytkownikowi uruchamiający aplikację uprawnienia `put` i `passall` .

*Obsługa komunikatów nieprzetwarzalnych w ASF*
W przypadku korzystania z narzędzi ASF (Application Server Facilities) konsument ConnectionConsumer, a nie MessageConsumer, przetwarza komunikaty nieprzetwarzalne. Konsument ConnectionConsumer ponownie umieszcza komunikaty w kolejce zgodnie z właściwościami nazwy QName BackoutThreshold i BackoutRequeuekolejki.

Jeśli aplikacja używa klasy ConnectionConsumers, okoliczności, w których komunikat jest wycofywany, zależą od sesji udostępnianej przez serwer aplikacji:

- Jeśli sesja nie jest transakcją, z wartością AUTO_ACKNOWLEDGE lub DUPS_OK_ACKNOWLEDGE, komunikat jest wycofywany tylko po wystąpieniu błędu systemowego lub w przypadku nieoczekiwanego zakończenia aplikacji.
- Jeśli sesja nie jest transakcją z CLIENT_ACKNOWLEDGE, niepotwierdzone komunikaty mogą zostać wycofane przez serwer aplikacji wywołujący metodę `Session.recover()`.

  Zazwyczaj implementacja klienta MessageListener lub serwer aplikacji wywołuje funkcję `Message.acknowledge()`. Program `Message.acknowledge()` potwierdza wszystkie komunikaty dostarczone do tej pory w sesji.
- Gdy sesja jest transakcją, niepotwierdzone komunikaty mogą zostać wycofane przez serwer aplikacji o nazwie `Session.rollback()`.

## Przeglądarki kolejek

Aplikacja używa przeglądarki kolejek do przeglądania komunikatów w kolejce bez ich usuwania.

Aby utworzyć przeglądarkę kolejek, aplikacja wywołuje metodę tworzenia przeglądarki kolejek obiektu ISession, określając jako parametr obiekt docelowy, który identyfikuje kolejkę do przeglądania. Aplikacja może utworzyć przeglądarkę kolejek z selektorem komunikatów lub bez niego.

Po utworzeniu przeglądarki kolejek aplikacja może wywołać metodę GetEnumerator obiektu IQueueBrowser , aby uzyskać listę komunikatów w kolejce. Ta metoda zwraca element wyliczeniowy, który hermetyzuje listę obiektów Message. Kolejność obiektów Message na liście jest taka sama, jak kolejność, w jakiej komunikaty będą pobierane z kolejki. Aplikacja może następnie użyć wyliczeniowego do przeglądania każdego komunikatu po kolei.

Element wyliczenia jest aktualizowany dynamicznie, gdy komunikaty są umieszczane w kolejce i usuwane z kolejki. Za każdym razem, gdy aplikacja wywołuje funkcję IEnumerator.MoveNext() w celu przeglądania następnego komunikatu w kolejce, komunikat odzwierciedla bieżącą zawartość kolejki.

Aplikacja może wywołać metodę GetEnumerator więcej niż raz dla danej przeglądarki kolejek. Każde wywołanie zwraca nowy element wyliczenia. Dlatego aplikacja może używać więcej niż jednego enumeratora do przeglądania komunikatów w kolejce i utrzymywania wielu pozycji w kolejce.

Aplikacja może użyć przeglądarki kolejek, aby wyszukać odpowiedni komunikat do usunięcia z kolejki, a następnie użyć konsumenta komunikatów z selektorem komunikatów, aby usunąć komunikat. Selektor komunikatów może wybrać komunikat zgodnie z wartością pola nagłówka JMSMessageID . Więcej informacji na temat tego i innych pól nagłówka komunikatu JMS zawiera sekcja "Pola nagłówka w komunikacie XMS" na stronie 689.

## Żądający

Aplikacja używa requestera do wysyłania komunikatu żądania, a następnie do oczekiwania i odbierania odpowiedzi.

Wiele aplikacji przesyłania komunikatów jest opartych na algorytmach, które wysyłają komunikat żądania, a następnie oczekują na odpowiedź. Produkt XMS udostępnia klasę o nazwie Requestor, która ułatwia projektowanie tego stylu aplikacji.

W celu utworzenia requestera aplikacja wywołuje konstruktor Create Requestor klasy Requestor, określając jako parametry obiekt Session i obiekt Destination, który identyfikuje miejsce, do którego mają być wysyłane komunikaty żądań. Sesja nie może być transakcją ani mieć trybu potwierdzenia XMSC_CLIENT_ACKNOWLEDGE. Konstruktor automatycznie tworzy tymczasową kolejkę lub temat, do którego mają być wysyłane komunikaty odpowiedzi.

Po utworzeniu requestera aplikacja może wywołać metodę Request obiektu Requestor, aby wysłać komunikat żądania, a następnie oczekiwać na odpowiedź z aplikacji, która odbiera komunikat żądania, i odebrać tę odpowiedź. Wywołanie oczekuje na odebranie odpowiedzi lub na zakończenie sesji, w zależności od tego, co nastąpi wcześniej. Dla każdego komunikatu żądania wymagana jest tylko jedna odpowiedź.

Po zamknięciu requestera aplikacja usuwa tymczasową kolejkę lub temat. Powiązana sesja nie zostanie jednak zamknięta.

## Usuwanie obiektu

Gdy aplikacja usuwa utworzony obiekt XMS , XMS zwalnia zasoby wewnętrzne, które zostały przydzielone do tego obiektu.

Gdy aplikacja tworzy obiekt XMS , program XMS przydziela do obiektu pamięć i inne zasoby wewnętrzne. Program XMS zachowuje te zasoby wewnętrzne do momentu jawnego usunięcia obiektu przez wywołanie metody close lub delete obiektu, w którym to momencie program XMS zwalnia zasoby wewnętrzne. Jeśli aplikacja próbuje usunąć obiekt, który został już usunięty, wywołanie jest ignorowane.

Gdy aplikacja usuwa obiekt połączenia lub sesji, program XMS automatycznie usuwa niektóre powiązane obiekty i zwalnia ich zasoby wewnętrzne. Są to obiekty, które zostały utworzone przez obiekt połączenia lub sesji i nie mają funkcji niezależnej od obiektu. Te obiekty są wyświetlane w sekcji Tabela 83 na stronie 667.

**Uwaga:** Jeśli aplikacja zamknie połączenie z sesjami zależnymi, wszystkie obiekty zależne od tych sesji również zostaną usunięte. Tylko obiekt połączenia lub sesji może mieć obiekty zależne.

*Tabela 83. Obiekty, które są automatycznie usuwane*

| **Usunięty obiekt** | **Metoda** | **Obiekty zależne, które są automatycznie usuwane** |
|---|---|---|
| Połączenie | Zamknij połączenie | Obiekty danych i sesji ConnectionMeta |
| Sesja | Zamknij sesję | MessageConsumer, MessageProducer, QueueBrowseri obiekty Requestor |

## Zarządzane transakcje XA systemu IBM MQ za pośrednictwem serwera XMS

Zarządzane transakcje XA systemu IBM MQ mogą być używane za pośrednictwem serwera XMS.

Aby używać transakcji XA za pośrednictwem produktu XMS, należy utworzyć sesję transakyjną. Jeśli transakcja XA jest używana, sterowanie transakcją odbywa się za pośrednictwem transakcji globalnych DTC (Distributed Transaction Coordinator), a nie za pośrednictwem sesji XMS . Jeśli używane są transakcje XA, w sesji XMS nie można wydać komendy `Session.commit` ani `Session.rollback` . Zamiast tego należy użyć metod `Transscope.Commit` lub `Transscope.Rollback` DTC w celu zatwierdzenia lub wycofania transakcji. Jeśli sesja jest używana dla transakcji XA, producent lub konsument utworzony za pomocą tej sesji musi być częścią transakcji XA. Nie można ich używać dla żadnych operacji poza zasięgiem transakcji XA. Nie można ich używać dla operacji takich jak `Producer.send` lub `Consumer.receive` poza transakcją XA.

Obiekt wyjątku `IllegalStateException` jest zgłaszany, jeśli:

- Sesja transakcyjne XA jest używana w systemie `Session.commit` lub `Session.rollback`.
- Obiekty producenta lub konsumenta, które są używane w sesji transakcyjnej XA, są używane po stronie zasięgu transakcji XA.

Transakcje XA nie są obsługiwane przez konsumentów asynchronicznych.

**Uwaga:**

1. Przed zatwierdzeniem transakcji XA może zostać wykonane zamknięcie obiektu `Producer`, `Consumer`, `Session`lub `Connection` . W takich przypadkach komunikaty w transakcji są wycofywane. Podobnie, jeśli połączenie zostanie zerwane przed zatwierdzeniem transakcji XA, wszystkie komunikaty w transakcji zostaną wycofane. W przypadku obiektu `Producer` wycofanie zmian oznacza, że komunikaty nie są umieszczane w kolejce. W przypadku obiektu `Consumer` wycofanie zmian oznacza, że komunikaty pozostają w kolejce.
2. Jeśli obiekt `Producer` umieszcza komunikat z wartością `TimeToLive` w pliku `TransactionScope` , a po upływie tego czasu generowany jest komunikat `commit` , komunikat może utracić ważność przed wysłaniem komunikatu `commit` . W takim przypadku komunikat nie jest udostępniany obiektom `Consumer` .
3. Obiekty `Session` nie są obsługiwane między wątkami. Użycie transakcji z obiektami `Session` , które są współużytkowane przez wątki, nie jest obsługiwane.

## Typy podstawowe XMS

XMS udostępnia odpowiedniki ośmiu typów podstawowych Java (byte, short, int, long, float, double, char i boolean). Umożliwia to wymianę komunikatów między systemami XMS i JMS bez utraty lub uszkodzenia danych.

Tabela 84 na stronie 668 zawiera listę równoważnych Java typów danych, wielkości oraz minimalnej i maksymalnej wartości każdego typu podstawowego XMS .

*Tabela 84. Typy danych XMS i ich odpowiedniki Java*

| Typ danych XMS | Zgodny typ danychJava | Wielkość | Wartość minimalna | Maksymalna wartość |
|---|---|---|---|---|
| System.Boolean | boolean (boolowskie) | 32 bity | Fałsz | Prawda |
| System.SBYTE | B | 8 bitów | $-2^7$ (-128) | $2^7$-1 (127) |
| System.BYTE | B | 8 bitów | $-2^7$ (-128) | $2^7$-1 (127) |
| System.CHAR | B | 8 bitów | $-2^7$ (-128) | $2^7$-1 (127) |
| System.Int16 | short | 16 bitów | $-2^{15}$ (-32768) | $2^{15}$-1 (32767) |

| *Tabela 84. Typy danych XMS i ich odpowiedniki Java (kontynuacja)* | | | | |
|---|---|---|---|---|
| **Typ danych XMS** | **Zgodny typ danychJava** | **Wielkość** | **Wartość minimalna** | **Maksymalna wartość** |
| System.Int32 | int | 32 bity | $-2^{31}$ (-2147483648) | $2^{31}$-1 (2147483647) |
| System.Int64 | long | 64 bity | $-2^{63}$ (-9223372036854775808) | $2^{63}$-1 (9223372036854775807) |
| System.Single | liczba zmiennopozycyjna | 32 bity | -3.402823E-38 (do 7-cyfrowej precyzji) | 3.402823E+38 (do 7-cyfrowej precyzji) |
| System.Double | double (podwójna) | 64 bity | -1.79769313486231E-308 (do 15 -cyfrowej precyzji) | 1.79769313486231E+308 (do 15 -cyfrowej precyzji) |

## Niejawna konwersja wartości właściwości z jednego typu danych do innego

Gdy aplikacja pobiera wartość właściwości, wartość może zostać przekształcona przez program XMS w inny typ danych. Wiele reguł określa, które konwersje są obsługiwane i w jaki sposób program XMS wykonuje konwersje.

Właściwość obiektu ma nazwę i wartość; z wartością powiązany jest typ danych, z którym wartość właściwości jest również określana jako *typ właściwości*.

Aplikacja używa metod klasy PropertyContext do pobierania i ustawiania właściwości obiektów. Aby uzyskać wartość właściwości, aplikacja wywołuje metodę odpowiednią dla typu właściwości. Aby na przykład uzyskać wartość właściwości będącej liczbą całkowitą, aplikacja zwykle wywołuje metodę właściwości GetInt.

Jeśli jednak aplikacja pobiera wartość właściwości, wartość może zostać przekształcona przez program XMS w inny typ danych. Aby na przykład uzyskać wartość właściwości będącej liczbą całkowitą, aplikacja może wywołać metodę właściwości GetString, która zwraca wartość właściwości w postaci łańcucha. Konwersje obsługiwane przez XMS są przedstawione w sekcji Tabela 85 na stronie 669.

| *Tabela 85. Obsługiwane konwersje z typu właściwości do innych typów danych* | |
|---|---|
| **Typ właściwości** | **Obsługiwane docelowe typy danych** |
| 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 |
| Tablica System.SByte | System.String |
| System.Int16 | System.String, System.Int32, System.Int64 |

Obsługiwane konwersje są obsługiwane przez następujące reguły ogólne:

- Wartości liczbowe właściwości mogą być przekształcane z jednego typu danych na inny, pod warunkiem, że podczas konwersji nie zostaną utracone żadne dane. Na przykład wartość właściwości

o typie danych System.Int32 można przekształcić w wartość o typie danych System.Int64, ale nie można jej przekształcić w wartość o typie danych System.Int16.

- Wartość właściwości dowolnego typu danych można przekształcić w łańcuch.
- Wartość właściwości łańcuchowej może zostać przekształcona w dowolny inny typ danych, pod warunkiem, że łańcuch jest poprawnie sformatowany na potrzeby konwersji. Jeśli aplikacja podejmie próbę przekształcenia wartości właściwości łańcuchowej, która nie jest poprawnie sformatowana, produkt XMS może zwrócić błędy.
- Jeśli aplikacja podejmie próbę konwersji, która nie jest obsługiwana, program XMS może zwrócić błąd.

Podczas przekształcania wartości właściwości z jednego typu danych na inny obowiązują następujące reguły:

- Podczas przekształcania wartości właściwości boolowskiej w łańcuch, wartość true jest przekształcana w łańcuch "true", a wartość false jest przekształcana w łańcuch "false".
- Podczas przekształcania wartości właściwości boolowskiej w liczbowy typ danych, w tym pliku System.SByte, wartość true jest przekształcana w wartość 1, a wartość false w wartość 0.
- Podczas przekształcania wartości właściwości łańcuchowej w wartość boolowską łańcuch "true" (bez rozróżniania wielkości liter) lub "1" jest przekształcany w wartość true, a łańcuch "false" (bez rozróżniania wielkości liter) lub "0" jest przekształcany w wartość false. Nie można przekształcić wszystkich innych łańcuchów.
- Podczas przekształcania wartości właściwości łańcuchowej w wartość o typie danych System.Int32, System.Int64, System.SBytelub System.Int16łańcuch musi mieć następujący format:

  [*odstępy*] [*znak*]*cyfry*

  Komponenty łańcucha są zdefiniowane w następujący sposób:

  ***wartości puste***
  Opcjonalne początkowe znaki odstępu.

  ***znak***
  Opcjonalny znak plus (+) lub minus (-).

  ***cyfry***
  Ciągła sekwencja cyfr (0-9). Musi występować co najmniej jedna cyfra.

  Po sekwencji cyfr łańcuch może zawierać inne znaki, które nie są cyframi, ale konwersja kończy się natychmiast po osiągnięciu pierwszego z tych znaków. Przyjmuje się, że łańcuch reprezentuje dziesiętną liczbę całkowitą.

  XMS może zwrócić błąd, jeśli łańcuch nie jest poprawnie sformatowany.
- Podczas przekształcania wartości właściwości łańcuchowej w wartość o typie danych System.Double lub System.Floatłańcuch musi mieć następujący format:

  [*odstępy*] [*znak*] [*cyfry*] [*punkt*[*d_cyfry*]] [*e_znak*[*e_znak*]*e_cyfry*]

  Komponenty łańcucha są zdefiniowane w następujący sposób:

  ***wartości puste***
  (Opcjonalnie) Czoące się znaki odstępu.

  ***znak***
  (Opcjonalnie) Znak plus (+) lub znak minus (-).

  ***cyfry***
  Ciągła sekwencja cyfr (0-9). W polu *cyfry* lub *d_cyfry*musi występować co najmniej jedna cyfra.

  ***punkt***
  (Opcjonalnie) Separator dziesiętny (.).

  ***d_cyfry***
  Ciągła sekwencja cyfr (0-9). W polu *cyfry* lub *d_cyfry*musi występować co najmniej jedna cyfra.

  ***_znak***
  Znak wykładnika, który jest *E* lub *e*.

***e_sign (znak)***
(Opcjonalnie) Znak plus (+) lub znak minus (-) dla wykładnika.

***e_cyfry***
Ciągła sekwencja cyfr (0-9) dla wykładnika. Jeśli łańcuch zawiera znak wykładnika, musi być obecny co najmniej jeden znak cyfry.

Po sekwencji cyfr lub opcjonalnych znaków reprezentujących wykładnik łańcuch może zawierać inne znaki, które nie są znakami cyfrowymi, ale konwersja kończy się natychmiast po osiągnięciu pierwszego z tych znaków. Przyjmuje się, że łańcuch reprezentuje dziesiętną liczbę zmiennopozycyjną z wykładnikiem, który jest potęgą liczby 10.

XMS może zwrócić błąd, jeśli łańcuch nie jest poprawnie sformatowany.

- Podczas przekształcania liczbowej wartości właściwości w łańcuch, w tym wartości właściwości o typie danych System.SByte, wartość jest przekształcana w reprezentację łańcuchową wartości w postaci liczby dziesiętnej, a nie w łańcuch zawierający znak ASCII dla tej wartości. Na przykład liczba całkowita 65 jest przekształcana w łańcuch "65", a nie w łańcuch "A".
- Podczas przekształcania wartości właściwości tablicy bajtów w łańcuch, każdy bajt jest przekształcany w 2 znaki szesnastkowe, które reprezentują ten bajt. Na przykład tablica bajtów {0xF1, 0x12, 0x00, 0xFF} jest przekształcana w łańcuch "F11200FF".

Konwersje z typu właściwości do innych typów danych są obsługiwane przez metody klas Property i PropertyContext .

## Iteratory

Iterator hermetyzuje listę obiektów i kursor utrzymujący bieżącą pozycję na liście. Pojęcie iteratora dostępne w produkcie IBM MQ Message Service Client (XMS) for C/C++jest implementowane przy użyciu interfejsu IEnumerator w produkcie IBM MQ Message Service Client (XMS) for .NET.

Podczas tworzenia iteratora pozycja kursora znajduje się przed pierwszym obiektem. Aplikacja używa iteratora do pobierania każdego obiektu po kolei.

Klasa iteratora IBM MQ Message Service Client (XMS) for C/C++ jest równoważna klasie Enumerator w produkcie Java. IBM MQ Message Service Client (XMS) for .NET jest podobny do Java i używa interfejsu IEnumerator.

Aplikacja może używać IEnumerator do wykonywania następujących zadań:

- Pobieranie właściwości komunikatu
- Pobieranie par nazwa-wartość w treści komunikatu odwzorowania
- Przeglądanie komunikatów w kolejce
- Aby uzyskać nazwy właściwości komunikatu zdefiniowanych przez JMS , które są obsługiwane przez połączenie

## Identyfikatory kodowanego zestawu znaków

W języku XMS .NETwszystkie łańcuchy są przekazywane przy użyciu rodzimego łańcucha .NET . Ponieważ kodowanie jest stałe, do jego interpretacji nie są wymagane żadne dodatkowe informacje. Dlatego właściwość XMSC_CLIENT_CCSID nie jest wymagana dla aplikacji XMS .NET .

## Kody błędów i wyjątków XMS

XMS korzysta z zakresu kodów błędów w celu wskazania awarii. Te kody błędów nie są jawnie wymienione w tej dokumentacji, ponieważ mogą się różnić w zależności od wersji. Tylko kody wyjątków systemu XMS (w formacie XMS_X_ ...) są udokumentowane, ponieważ pozostają takie same w różnych wersjach systemu XMS.

### Automatyczne ponowne połączenie klienta IBM MQ za pośrednictwem XMS

Skonfiguruj klienta XMS tak, aby automatycznie ponownie nawiązywał połączenie po awarii sieci, menedżera kolejek lub serwera podczas używania klienta IBM WebSphere MQ 7.1 lub nowszego jako dostawcy komunikatów.

Właściwości `WMQ_CONNECTION_NAME_LIST` i `WMQ_CLIENT_RECONNECT_OPTIONS` klasy `MQConnectionFactory` umożliwiają skonfigurowanie połączenia klienta na potrzeby automatycznego ponownego nawiązywania połączenia. Automatyczne ponowne nawiązywanie połączenia przez klienta powoduje ponowne nawiązanie połączenia z klientem po awarii połączenia lub jako opcja po zatrzymaniu menedżera kolejek. Projekt niektórych aplikacji klienckich powoduje, że nie nadają się one do automatycznego ponownego połączenia.

Automatycznie wznawialne połączenia klienckie stają się ponownie nawiązywane po nawiązaniu połączenia.

**Uwaga:** Właściwości `Opcje ponownego połączenia klienta`, `Limit czasu ponownego połączenia klienta`i `Lista nazw połączeń` można również ustawić za pomocą tabeli definicji kanału klienta (CCDT) lub przez włączenie ponownego połączenia klienta za pośrednictwem pliku `mqclient.ini` .

**Uwaga:** Jeśli właściwości ponownego połączenia są ustawione w obiekcie `ConnectionFactory` oraz w tabeli CCDT, reguła pierwszeństwa jest następująca. Jeśli wartość domyślna właściwości listy nazw połączeń jest ustawiona w obiekcie `ConnectionFactory` , pierwszeństwo ma tabela CCDT. Jeśli lista nazw połączeń nie jest ustawiona na wartość domyślną, pierwszeństwo mają wartości właściwości ustawione w obiekcie `ConnectionFactory` . Wartością domyślną listy nazw połączeń jest `localhost(1414)`.

## Pisanie aplikacji XMS .NET

Tematy w tej sekcji zawierają informacje pomocne podczas pisania aplikacji XMS .NET .

### O tym zadaniu

Ta sekcja zawiera informacje dotyczące pisania aplikacji XMS .NET . Ogólne informacje na temat pisania aplikacji XMS zawiera sekcja “Pisanie aplikacji XMS” na stronie 653.

Sekcja obejmuje następujące tematy:



### Typy danych dla .NET

Serwer XMS .NET obsługuje metody 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, i System.Object. Typy danych w produkcie XMS .NET różnią się od typów danych w języku XMS C/C + +. W tym temacie opisano odpowiednie typy danych.

W poniższej tabeli przedstawiono odpowiednie typy danych XMS .NET i XMS C/C + +, a także ich krótki opis.

| *Tabela 86. Typy danych dla XMS .NET i XMS C/C++* | | |
|---|---|---|
| **Typ XMS .NET** | **XMS Typ C/C++** | **Opis** |
| System.SByte | xmsSBYTE<br>xmsINT8 | Wartość 8-bitowa ze znakiem |
| System.Byte | xmsBYTE<br>xmsUINT8 | Wartość 8-bitowa bez znaku |
| System.Int16 | xmsINT16<br>xmsSHORT | Wartość 16-bitowa ze znakiem |
| System.UInt16 | xmsUINT16<br>xmsUSHORT | Wartość 16-bitowa bez znaku |
| System.Int32 | xmsINT32<br>xmsINT | Wartość 32-bitowa ze znakiem |
| System.UInt32 | xmsUINT32<br>xmsUINT | Wartość 32-bitowa bez znaku |
| System.Int64 | xmsLONG<br>xmsINT64 | Wartość 64-bitowa ze znakiem |
| System.UInt64 | xmsULONG<br>xmsUINT64 | Wartość 64-bitowa bez znaku |
| System.Char | xmsCHAR16 | 16-bitowy znak bez znaku (Unicode for .NET) |
| System.Single | xmsFLOAT | 32-bitowa liczba zmiennopozycyjna IEEE |
| System.Double | xmsDOUBLE | 64-bitowa liczba zmiennopozycyjna IEEE |
| System.Boolean | xmsBOOL | Wartość prawda/fałsz |
| Nie dotyczy | xmsCHAR | 8-bitowa wartość ze znakiem lub bez znaku (wartość ze znakiem lub bez znaku zależy od platformy) |
| System.Decimal | Nie dotyczy | 96-bitowa liczba całkowita ze znakiem $10^0$ do $10^{28}$ |
| System.Object | Nie dotyczy | Podstawa wszystkich typów |
| System.String | Nie dotyczy | Typ łańcuchowy |

## Operacje zarządzane i niezarządzane w produkcie .NET

Kod zarządzany jest wykonywany wyłącznie w środowisku wykonawczym języka wspólnego .NET i jest całkowicie zależny od usług udostępnianych przez to środowisko wykonawcze. Aplikacja jest klasowana jako niezarządzana, jeśli dowolna część aplikacji uruchamia lub wywołuje usługi poza wspólnym środowiskiem wykonawczym języka .NET .

Niektóre zaawansowane funkcje nie mogą być obecnie obsługiwane w zarządzanym środowisku .NET .

Jeśli aplikacja wymaga pewnych funkcji, które nie są obecnie obsługiwane w środowisku w pełni zarządzanym, można zmienić aplikację w taki sposób, aby używała środowiska niezarządzanego bez konieczności wprowadzania istotnych zmian w aplikacji. Należy jednak zauważyć, że po dokonaniu tego wyboru stos XMS korzysta z niezarządzanego kodu.

### Połączenia z menedżerem kolejek produktu IBM MQ

Połączenia zarządzane z produktem WMQ_CM_CLIENT nie obsługują komunikacji innej niż TCP ani kompresji kanału. Jednak te połączenia mogą być obsługiwane przez połączenie niezarządzane (WMQ_CM_CLIENT_UNMANAGED). Więcej informacji na ten temat zawiera "Tworzenie aplikacji .NET" na stronie 573.

Jeśli fabryka połączeń jest tworzona z obiektu administrowanego w środowisku niezarządzanym, należy ręcznie zmienić wartość trybu połączenia na XMSC_WMQ_CM_CLIENT_UNMANAGED.

### Połączenia z mechanizmem przesyłania komunikatów magistrali integracji usług produktu WebSphere Application Server

Połączenia z mechanizmem przesyłania komunikatów magistrali integracji usług WebSphere Application Server , które wymagają użycia protokołu SSL (w tym HTTPS), nie są obecnie obsługiwane jako kod zarządzany.

## Miejsca docelowe w produkcie .NET

W produkcie .NETmiejsca docelowe są tworzone zgodnie z typem protokołu i mogą być używane tylko w przypadku typu protokołu, dla którego zostały utworzone.

Dostępne są dwie funkcje służące do tworzenia miejsc docelowych, jedna dla tematów i jedna dla kolejek:

- `IDestination CreateTopic(String topic);`
- `IDestination CreateQueue(String queue);`

Te funkcje są dostępne dla następujących dwóch obiektów w interfejsie API:

- ISesja
- XMSFactoryFactory

W obu przypadkach metody te mogą akceptować łańcuch stylu identyfikatora URI, który może zawierać parametry, w następującym formacie:

```
"topic://some/topic/name?priority=5"
```

Alternatywnie te metody mogą akceptować tylko nazwę miejsca docelowego, czyli nazwę bez przedrostka topic: // lub queue: // i bez parametrów.

Oznacza to, że następujący łańcuch stylu identyfikatora URI:

```
CreateTopic("topic://some/topic/name");
```

otrzymany wynik jest taki sam, jak następująca nazwa miejsca docelowego:

```
CreateTopic("some/topic/name");
```

Podobnie jak w przypadku WebSphere Application Server magistrali integracji usług JMS, tematy można również określać w postaci skróconej, która obejmuje zarówno *topicname* , jak i *topicspace* , ale nie może zawierać parametrów:

```
CreateTopic("topicspace:topicname");
```

## Właściwości w pliku .NET

Aplikacja .NET używa metod interfejsu PropertyContext do pobierania i ustawiania właściwości obiektów.

Interfejs PropertyContext obudowuje metody, które służą do pobierania i ustawiania właściwości. Metody te są dziedziczone, bezpośrednio lub pośrednio, przez następujące klasy:

- BytesMessage
- Połączenie
- ConnectionFactory
- ConnectionMetaDane
- Cel
- MapMessage
- Komunikat
- MessageConsumer
- MessageProducer
- ObjectMessage
- QueueBrowser
- Sesja
- StreamMessage
- TextMessage

Jeśli aplikacja ustawi wartość właściwości, nowa wartość zastąpi poprzednią wartość właściwości.

Więcej informacji na temat właściwości XMS zawiera sekcja Właściwości obiektów XMS.

Aby ułatwić używanie, nazwy i wartości właściwości XMS w pliku XMS są predefiniowane jako stałe publiczne w strukturze o nazwie XMSC. Nazwy tych stałych mają postać XMSC.*stała*, na przykład XMSC.USERID (stała nazwy właściwości) i XMSC.DELIVERY_AS_APP (stała wartości).

Dodatkowo można uzyskać dostęp do stałych IBM MQ za pomocą programu IBM.XMS.MQC MQC. Jeśli IBM.XMS jest już zaimportowana, można uzyskać dostęp do wartości tych właściwości w postaci MQC.*stała*. Na przykład MQC.MQRO_COA_WITH_FULL_DATA.

Ponadto, jeśli używana jest aplikacja hybrydowa, która używa zarówno klas XMS .NET , jak i IBM MQ dla .NET i importuje obie klasy IBM.XMS i IBM.WMQ należy w pełni kwalifikować przestrzeń nazw struktury MQC, aby zapewnić unikalność każdego wystąpienia.

Niektóre zaawansowane funkcje nie są obecnie obsługiwane w zarządzanym środowisku .NET . Więcej informacji na ten temat zawiera sekcja “Operacje zarządzane i niezarządzane w produkcie .NET” na stronie 673 .

## Obsługa nieistniejących właściwości w produkcie .NET

Obsługa nieistniejących właściwości w XMS .NET jest zasadniczo spójna ze specyfikacją JMS, a także z implementacjami języka C i C++ języka XMS.

W usłudze JMS uzyskanie dostępu do nieistniejącej właściwości może spowodować wystąpienie wyjątku systemowego Java , gdy metoda próbuje przekształcić nieistniejącą (null) wartość w wymagany typ. Jeśli właściwość nie istnieje, występują następujące wyjątki:

- Metoda getStringProperty i metoda getObjectzwracają wartość NULL
- Metoda getBooleanzwraca wartość false, ponieważ metoda Boolean.valueOf(null) zwraca wartość false
- Metoda getIntProperty etc.throw java.lang.NumberFormatException , ponieważ metoda Integer.valueOf(null) zgłasza wyjątek

Jeśli właściwość nie istnieje w produkcie XMS .NET , występują następujące wyjątki:

- GetStringWłaściwość i właściwość GetObject(i właściwość GetBytes) zwracają wartość NULL (która jest taka sama jak Java)
- Metoda GetBooleanProperty zgłasza wyjątek System.NullReferenceException
- Metoda GetIntProperty itp. zgłasza wyjątek System.NullReferenceException

Ta implementacja różni się od implementacji Java, ale jest ogólnie zgodna ze specyfikacją JMS oraz interfejsami C i C + + języka XMS . Podobnie jak w przypadku implementacji Java , serwer XMS .NET propaguje wszystkie wyjątki z wywołania System.Convert do programu wywołującego. Inaczej niż w przypadku metody Java , metoda XMS jawnie zgłasza wyjątki NullReference, a nie tylko rodzime zachowanie środowiska .NET polegające na przekazywaniu wartości NULL do procedur konwersji systemu. Jeśli aplikacja ustawi właściwość na typ String (np. "abc" ) i wywoła właściwość GetInt, wyjątek System.FormatException zgłoszony przez metodę Convert.ToInt32("abc") jest propagowany do programu wywołującego, co jest spójne z wartością Java. Wyjątek MessageFormatjest zgłaszany tylko wtedy, gdy typy używane dla metod setProperty i getProperty są niezgodne. To zachowanie jest również spójne z wartością Java.

## Obsługa błędów w produkcie .NET

Wszystkie wyjątki serwera XMS .NET są wyprowadzane z wyjątku System.Exception. Wywołania metod XMS mogą zgłaszać konkretne wyjątki XMS , takie jak wyjątek MessageFormat, ogólne wyjątki XMSException lub wyjątki systemowe, takie jak wyjątek NullReference.

Napisz aplikacje w celu wychwycenia dowolnego z tych błędów, zarówno w konkretnych blokach catch, jak i w ogólnych blokach catch `System.Exception` , zgodnie z wymaganiami aplikacji.

## Korzystanie z obiektów nasłuchiwania wyjątków i komunikatów w produkcie .NET

Aplikacja .NET używa obiektu nasłuchiwania komunikatów do asynchronicznego odbierania komunikatów i używa obiektu nasłuchiwania wyjątków do asynchronicznego powiadamiania o problemie z połączeniem.

### O tym zadaniu

Funkcjonalność zarówno obiektów nasłuchiwania komunikatów, jak i wyjątków jest taka sama w przypadku języka .NET i języka C + +. Istnieją jednak niewielkie różnice w implementacji.

### Procedura

- Aby skonfigurować nasłuchiwanie komunikatów w celu asynchronicznego odbierania komunikatów, wykonaj następujące kroki:
  a) Zdefiniuj metodę, która jest zgodna z sygnaturą delegata obiektu nasłuchiwania komunikatów.
  Definiowana metoda może być metodą statyczną lub metodą instancji i może być zdefiniowana w dowolnej dostępnej klasie. Podpis delegata jest następujący:

```
public delegate void MessageListener(IMessage msg);
```

  i tak można zdefiniować metodę jako:

```
void SomeMethodName(IMessage msg);
```

  b) Utwórz instancję tej metody jako delegata, korzystając z przykładu podobnego do następującego:

```
MessageListener OnMsgMethod = new MessageListener(SomeMethodName)
```

  c) Zarejestruj delegata z co najmniej jednym konsumentem, ustawiając właściwość MessageListener konsumenta:

```
consumer.MessageListener = OnMsgMethod;
```

Delegata można usunąć, ustawiając parametr MessageListener z powrotem na wartość NULL:

```
consumer.MessageListener = null;
```

- Aby skonfigurować obiekt nasłuchiwania wyjątków, wykonaj następujące kroki.

  Obiekt nasłuchiwania wyjątków działa podobnie jak obiekt nasłuchiwania komunikatów, ale ma inną definicję delegowania i jest przypisany do połączenia, a nie do konsumenta komunikatów. Jest taka sama jak w przypadku języka C + +.

  a) Zdefiniuj metodę.

  Podpis delegata jest następujący:

```
public delegate void ExceptionListener(Exception ex);
```

  i tak zdefiniowana metoda może być:

```
void SomeMethodName(Exception ex);
```

  b) Utwórz instancję tej metody jako delegata, korzystając z przykładu podobnego do następującego:

```
ExceptionListener OnExMethod = new ExceptionListener(SomeMethodName)
```

  c) Zarejestruj delegata z połączeniem, ustawiając jego właściwość ExceptionListener :

```
connection.ExceptionListener = OnExMethod ;
```

  Delegata można usunąć, resetując obiekt ExceptionListener do następującej wartości:

```
null:  connection.ExceptionListener = null;
```

# Praca z administrowanymi obiektami XMS .NET

Tematy w tej sekcji zawierają informacje o administrowanych obiektach. Aplikacje XMS mogą pobierać definicje obiektów z centralnego repozytorium obiektów administrowanych i używać ich do tworzenia fabryk połączeń i miejsc docelowych.

## O tym zadaniu

Ta sekcja zawiera informacje pomocne podczas tworzenia obiektów administrowanych i zarządzania nimi, opisujące typy repozytoriów obiektów administrowanych obsługiwanych przez program XMS . W tej sekcji wyjaśniono również, w jaki sposób aplikacja XMS nawiązuje połączenie z repozytorium obiektów administrowanych w celu pobrania wymaganych obiektów administrowanych.

Sekcja obejmuje następujące tematy:

- “Format identyfikatora URI dla kontekstów początkowych XMS” na stronie 684
- “Usługa WWW wyszukiwania JNDI dla XMS .NET” na stronie 685
- “XMS .NET pobieranie obiektów administrowanych” na stronie 685

## XMS .NET obsługiwane typy repozytoriów obiektów administrowanych

Administrowane obiekty systemu plików i LDAP mogą być używane do nawiązywania połączeń z systemami IBM MQ i WebSphere Application Server, podczas gdy nazewnictwo COS może być używane tylko do nawiązywania połączeń z serwerem WebSphere Application Server.

Katalogi obiektów systemu plików mają postać obiektów Java Naming Directory Interface (JNDI) przekształconych do postaci szeregowej. Katalogi obiektów LDAP to katalogi, które zawierają obiekty JNDI . Katalogami obiektów systemu plików i LDAP można administrować za pomocą programu IBM MQ Explorer, który jest dostarczany z produktem IBM MQ lub nowszym. Zarówno system plików, jak i katalogi obiektów LDAP mogą być używane do administrowania połączeniami klienckimi przez scentralizowanie fabryk połączeń i miejsc docelowych produktu IBM MQ . Administrator sieci może wdrożyć wiele aplikacji, które odwołują się do tego samego repozytorium centralnego i które są automatycznie aktualizowane w celu odzwierciedlenia zmian ustawień połączenia wprowadzonych w repozytorium centralnym.

Katalog nazw COS zawiera fabryki połączeń WebSphere Application Server service integration bus i miejsca docelowe, którymi można administrować za pomocą Konsoli administracyjnej produktu WebSphere Application Server . Aby aplikacja XMS mogła pobierać obiekty z katalogu nazw COS, należy wdrożyć usługę WWW wyszukiwania JNDI . Ta usługa Web Service nie jest dostępna we wszystkich WebSphere Application Server service integration technologies. Szczegółowe informacje można znaleźć w dokumentacji produktu.

**Uwaga:** Zrestartuj połączenia aplikacji, aby zmiany w katalogu obiektów zostały uwzględnione.

## Odwzorowanie właściwości XMS .NET dla obiektów administrowanych

Aby aplikacje XMS .NET mogły korzystać z definicji fabryki połączeń i obiektów docelowych produktu IBM MQ JMS i WebSphere Application Server , właściwości pobierane z tych definicji muszą być odwzorowane na odpowiednie właściwości XMS , które można ustawić w fabrykach połączeń i miejscach docelowych produktu XMS .

Aby utworzyć na przykład fabrykę połączeń XMS z właściwościami pobieranymi z fabryki połączeń JMS produktu IBM MQ , właściwości te muszą zostać odwzorowane między sobą.

Wszystkie odwzorowania właściwości są wykonywane automatycznie.

Tabela 87 na stronie 678 przedstawia odwzorowania między niektórymi z najczęściej używanych właściwości fabryk połączeń i miejsc docelowych. Właściwości, które są wyświetlane w tej tabeli, są tylko małym zestawem przykładów, a nie wszystkie właściwości, które są wyświetlane, są odpowiednie dla wszystkich typów połączeń i serwerów.

*Tabela 87. Przykłady odwzorowania nazw dla fabryki połączeń i właściwości miejsca docelowego*

| **Nazwa właściwości IBM MQ JMS** | **XMS Nazwa właściwości** | **WebSphere Application Server service integration bus Nazwa właściwości** |
|---|---|---|
| TRWAŁOŚĆ (NA) | TRYB_XMSC_DELIVERY_MODE | |
| TERMIN WAŻNOŚCI (EXP) | XMSC_TIME_TO_LIVE | |
| PRIORYTET (PRI) | XMSC_PRIORITY | |
| | XMSC_WPM_HOST_NAME | serverName |
| | XMSC_WPM_BUS_NAME | busName |
| | XMSC_WPM_TOPIC_SPACE | topicName |

**Uwaga:** Właściwości przedstawione na rysunku ( Tabela 88 na stronie 679 ) mają zastosowanie zarówno do usługi JMS, jak i do usługi XMS .NET.

*Tabela 88. Właściwości produktu XMS .NET*

| **Właściwość** | **Typ obiektu** | | | | |
|---|---|---|---|---|---|
| | **CF** | **QCF,** | **TCF,** | **Kolejka** | **Temat** |
| APPLICATIONNAME | Y | Y | Y | N/D | N/D |
| ASYNCEXCEPTION (ASYNCEXCEPTION) | Y | Y | Y | N/D | N/D |
| CCDTURL | Y | Y | Y | N/D | N/D |
| CHANNEL | Y | Y | Y | N/D | N/D |
| CONNECTIONNAMELIST (lista nazw połączeń) | Y | Y | Y | N/D | N/D |
| CLIENTRECONNECTOPTIONS | Y | Y | Y | N/D | N/D |
| CLIENTRECONNECTTIMEOUT | Y | Y | Y | N/D | N/D |
| CLIENTID | N/D | Y | N/D | N/D | N/D |
| COMPHDR "1" na stronie 680 | Y | N/D | Y | N/D | N/D |
| COMPMSG "1" na stronie 680 | Y | Y | Y | N/D | N/D |
| CONNOPT "1" na stronie 680 | Y | Y | Y | N/D | N/D |
| CONNTAG (znacznik połączenia) "1" na stronie 680 | Y | Y | Y | N/D | N/D |
| Opis"1" na stronie 680 | N/D | Y | N/D | Y | Y |
| TERMINWAŻNOŚCI "1" na stronie 680 | N/D | N/D | N/D | Y | Y |
| FAILIFQUIESCE | Y | Y | Y | Y | Y |
| nazwa_hosta | N/D | Y | N/D | N/D | N/D |
| LOCALADDRESS | N/D | Y | N/D | N/D | N/D |
| PERSISTENCE | N/D | N/D | N/D | Y | Y |
| PORT | N/D | Y | N/D | N/D | N/D |
| Priorytet"1" na stronie 680 | N/D | N/D | N/D | Y | Y |

| *Tabela 88. Właściwości produktu XMS .NET (kontynuacja)* | | | | | |
|---|---|---|---|---|---|
| **Właściwość** | **Typ obiektu** | | | | |
| | **CF** | **QCF,** | **TCF,** | **Kolejka** | **Temat** |
| WERSJA_DOSTAWCY "1" na stronie 680 | N/D | Y | N/D | N/D | N/D |
| QMANAGER | Y | Y | Y | Y | N/D |
| kolejka"1" na stronie 680 | N/D | N/D | N/D | Y | N/D |
| SHARECONVALLOWED | Y | Y | Y | N/D | N/D |
| Temat"1" na stronie 680 | N/D | N/D | N/D | N/D | Y |
| TRANSPORT "1" na stronie 680 | N/D | Y | N/D | N/D | N/D |

**Uwaga:**

1. Te właściwości nie mają właściwości na poziomie aplikacji, ale można je opcjonalnie ustawić przy użyciu właściwości administrowanych.

## OutboundSNI właściwość

V 9.3.0

Z poziomu produktu IBM MQ 9.3.0można ustawić właściwość XMSC_WMQ_OUTBOUND_SNI, która ustawia właściwość **OutboundSNI** w aplikacji.

Właściwość XMSC_WMQ_OUTBOUND_SNI_PROPERTY przyjmuje następujące wartości:

- XMSC_WMQ_OUTBOUND_SNI_CHANNEL, która jest odwzorowana na "CHANNEL"
- XMSC_WMQ_OUTBOUND_SNI_HOSTNAME, która jest odwzorowywana na wartość HOSTNAME
- XMSC_WMQ_OUTBOUND_SNI_GWIAZDKA, która jest odwzorowana na "*"

Ponadto można ustawić właściwość **OutboundSNI** przy użyciu zmiennej środowiskowej MQOUTBOUND_SNI, która przyjmuje następujące wartości:

- CHANNEL
- HOSTNAME
- *

**Uwaga:** Jeśli nie ustawiono konkretnej wartości, wartością domyślną właściwości jest XMSC_WMQ_OUTBOUND_SNI_CHANNEL.

Kolejność wykonywania operacji ustawiania właściwości **OutboundSNI** w węźle zarządzanym jest następująca:

1. Właściwość poziomu aplikacji
2. Zmienna środowiskowa

Dla właściwości **OutboundSNI** w węźle niezarządzanym obsługiwana jest tylko wartość `mqclient.ini` .

## XMS .NET wymagane właściwości dla administrowanych obiektów ConnectionFactory

Gdy aplikacja tworzy fabrykę połączeń, należy zdefiniować wiele właściwości, aby utworzyć połączenie z serwerem przesyłania komunikatów.

Właściwości wymienione w poniższych tabelach są minimalnymi wymaganymi przez aplikację, aby mogła utworzyć połączenie z serwerem przesyłania komunikatów. Aby dostosować sposób tworzenia połączenia, aplikacja może w razie potrzeby ustawić dowolne dodatkowe właściwości obiektu ConnectionFactory . Więcej informacji na ten temat zawiera sekcja Właściwości fabryki połączeń ConnectionFactory. Dołączona jest pełna lista dostępnych właściwości.

### Połączenie z menedżerem kolejek IBM MQ

*Tabela 89. Ustawienia właściwości administrowanych obiektów ConnectionFactory dla połączeń z menedżerem kolejek produktu IBM MQ .*

| **Wymagana właściwość XMS** | **Wymagana równoważna właściwość IBM MQ JMS** |
|---|---|
| XMSC_CONNECTION_TYPE | XMS określa tę wartość na podstawie nazwy klasy fabryki połączeń i właściwości TRANSPORT (TRAN). |
| XMSC_WMQ_HOST_NAME | NAZWA HOSTA (HOST) |
| XMSC_WMQ_PORT | PORT |
| XMSC_WMQ_QUEUE_MANAGER | Nazwa menedżera kolejek |

### Połączenie z brokerem w czasie rzeczywistym

*Tabela 90. Ustawienia właściwości administrowanych obiektów ConnectionFactory na potrzeby połączeń w czasie rzeczywistym z brokerem.*

| **wymaganeXMS** | **Wymagana równoważna właściwość IBM MQ JMS** |
|---|---|
| XMSC_CONNECTION_TYPE | XMS określa tę wartość na podstawie nazwy klasy fabryki połączeń i właściwości TRANSPORT (TRAN). |
| XMSC_RTT_HOST_NAME | NAZWA HOSTA (HOST) |
| XMSC_RTT_PORT | PORT |

### Połączenie z WebSphere Application Server service integration bus

*Tabela 91. Ustawienia właściwości dla administrowanych obiektów ConnectionFactory na potrzeby połączeń z serwerem WebSphere Application Server service integration bus*

| **XMS właściwość** | **Opis** |
|---|---|
| XMSC_CONNECTION_TYPE | Typ serwera przesyłania komunikatów, z którym łączy się aplikacja.. Jest ona określana na podstawie nazwy klasy fabryki połączeń. |
| XMSC_WPM_BUS_NAME | W przypadku fabryki połączeń: nazwa magistrali integracji usług, z którą aplikacja nawiązuje połączenie. W przypadku miejsca docelowego: nazwa magistrali integracji usług, w której znajduje się miejsce docelowe. |

## XMS .NET wymagane właściwości dla administrowanych obiektów docelowych

Aplikacja, która tworzy miejsce docelowe, musi ustawić kilka właściwości, które aplikacja ma w administrowanym obiekcie miejsca docelowego.

*Tabela 92. Ustawienia właściwości administrowanych obiektów docelowych*

| Typ połączenia | Właściwość | Opis |
|---|---|---|
| IBM MQ menedżer kolejek | KOLEJKA (QU)<br>TEMAT (TOP) | Kolejka, z którą ma zostać nawiązane połączenie<br>Temat używany przez aplikację jako miejsce docelowe |
| Połączenie z brokerem w czasie rzeczywistym | TEMAT (TOP) | Temat używany przez aplikację jako miejsce docelowe |
| WebSphere Application Server service integration bus | topicName<br>queueName | Jeśli aplikacja łączy się z tematem<br>Jeśli aplikacja łączy się z kolejką |

## XMS .NET tworzenie obiektów administrowanych

Definicje ConnectionFactory i obiektu docelowego, które są wymagane przez aplikacje XMS do nawiązania połączenia z serwerem przesyłania komunikatów, muszą zostać utworzone przy użyciu odpowiednich narzędzi administracyjnych.

### Zanim rozpoczniesz

Więcej informacji na temat różnych typów repozytoriów obiektów administrowanych obsługiwanych przez program XMS zawiera sekcja “XMS .NET obsługiwane typy repozytoriów obiektów administrowanych” na stronie 678.

### O tym zadaniu

Aby utworzyć obiekty administrowane dla produktu IBM MQ , należy użyć narzędzia administracyjnego JMS (JMSAdmin) produktu IBM MQ Explorer lub IBM MQ .

Aby utworzyć obiekty administrowane dla produktu IBM MQ lub IBM Integration Bus, należy użyć narzędzia administracyjnego JMS (JMSAdmin) produktu IBM MQ .

Aby utworzyć obiekty administrowane dla WebSphere Application Server service integration bus, należy użyć Konsoli administracyjnej produktu WebSphere Application Server .

W narzędziach administracyjnych właściwość ta jest nazywana w skrócie **APPLICATIONNAME** lub **APPNAME** .

**Uwaga:** Nie można użyć JMSAdmin do ustawienia TRANSPORT (UNMANAGED). Dlatego, aby uzyskać niezarządzany klient XMS przy użyciu nazwy aplikacji wybranej przez administratora, należy wprowadzić następującą komendę:

```
cf.SetIntProperty(XMSC.WMQ_CONNECTION_MODE, XMSC.WMQ_CM_CLIENT_UNMANAGED);
```

Poniższe kroki zawierają podsumowanie czynności wykonywanych w celu utworzenia obiektów administrowanych.

### Procedura

1. Utwórz fabrykę połączeń i zdefiniuj właściwości niezbędne do utworzenia połączenia z aplikacji do wybranego serwera.

Minimalne właściwości wymagane przez program XMS do nawiązania połączenia są zdefiniowane w pliku "XMS .NET wymagane właściwości dla administrowanych obiektów ConnectionFactory" na stronie 681.

2. Utwórz wymagane miejsce docelowe na serwerze przesyłania komunikatów, z którym łączy się aplikacja:
   - W przypadku połączenia z menedżerem kolejek produktu IBM MQ należy utworzyć kolejkę lub temat.
   - W przypadku połączenia z brokerem w czasie rzeczywistym należy utworzyć temat.
   - W przypadku połączenia z WebSphere Application Server service integration busnależy utworzyć kolejkę lub temat.

   Minimalne właściwości wymagane przez program XMS do nawiązania połączenia są zdefiniowane w pliku "XMS .NET wymagane właściwości dla administrowanych obiektów docelowych" na stronie 682.

## XMS .NET tworzenie obiektów InitialContext

Aplikacja musi utworzyć kontekst początkowy, który będzie używany do nawiązywania połączenia z repozytorium obiektów administrowanych w celu pobrania wymaganych obiektów administrowanych.

### O tym zadaniu

Obiekt InitialContext obudowuje połączenie z repozytorium. Interfejs API XMS udostępnia metody do wykonywania następujących zadań:

- Tworzenie obiektu InitialContext
- Wyszukaj obiekt administrowany w repozytorium obiektów administrowanych.

### Procedura

- Więcej informacji na temat tworzenia obiektu InitialContext zawiera sekcja InitialContext dla .NET oraz sekcja Właściwości obiektu InitialContext.

## Właściwości XMS .NET InitialContext

Parametry konstruktora InitialContext obejmują położenie repozytorium obiektów administrowanych, podane jako identyfikator URI (Uniform Resource Indicator). Aby aplikacja mogła nawiązać połączenie z repozytorium, konieczne może być podanie większej ilości informacji niż informacje zawarte w identyfikatorze URI.

W interfejsie JNDI oraz w .NET implementacji XMSdodatkowe informacje są udostępniane konstruktorowi w postaci tabeli mieszającej środowiska.

Położenie administrowanego repozytorium obiektów jest zdefiniowane we właściwości XMSC_IC_URL . Ta właściwość jest zwykle przekazywana w wywołaniu Create, ale może zostać zmodyfikowana w celu nawiązania połączenia z innym katalogiem nazw przed wyszukiwaniem. W przypadku kontekstów FileSystem lub LDAP ta właściwość definiuje adres katalogu. W przypadku nazewnictwa COS jest to adres usługi Web Service, która używa tych właściwości do nawiązywania połączenia z katalogiem JNDI.

Następujące właściwości są przekazywane bez modyfikacji do usługi Web Service, która będzie ich używać do nawiązywania połączenia z katalogiem JNDI.

- XMSC_IC_PROVIDER_URL (XMSC_PROVIDER_URL)
- XMSC_IC_SECURITY_CREDENTIALS
- XMSC_IC_SECURITY_AUTHENTICATION
- XMSC_IC_SECURITY_PRINCIPAL
- XMSC_IC_SECURITY_PROTOCOL

## Format identyfikatora URI dla kontekstów początkowych XMS

Położenie repozytorium obiektów administrowanych jest udostępniane jako identyfikator URI (Uniform Resource Indicator). Format identyfikatora URI zależy od typu kontekstu.

### Kontekst FileSystem

W przypadku kontekstu FileSystem adres URL określa położenie katalogu opartego na systemie plików. Struktura URL jest zgodna z definicją w dokumencie RFC 1738, *Uniform Resource Locator (URL)*: URL ma przedrostek `file://`, a składnia po tym przedrostku jest poprawną definicją pliku, który można otworzyć w systemie, w którym działa produkt XMS .

Ta składnia może być specyficzna dla platformy i może zawierać separatory '/' lub '\'. Jeśli używany jest znak '\', wówczas każdy separator musi być poprzedzony znakiem zmiany znaczenia za pomocą dodatkowego znaku '\'. Uniemożliwia to środowisku .NET interpretowanie separatora jako znaku zmiany znaczenia dla następujących elementów.

Poniższe przykłady ilustrują tę składnię:

```
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
```

### Kontekst LDAP

W przypadku kontekstu LDAP podstawowa struktura adresu URL jest zgodna z definicją w dokumencie RFC 2255 *The LDAP URL Format*, z przedrostkiem bez rozróżniania wielkości liter `ldap://`

Dokładna składnia została zilustrowana w poniższym przykładzie:

```
LDAP://[Hostname][:Port]["/"[DistinguishedName]]
```

Ta składnia jest zgodna z definicją w RFC, ale nie obsługuje żadnych atrybutów, zasięgu, filtrów ani rozszerzeń.

Przykładami tej składni są:

```
ldap://madison:389/cn=JMSData,dc=IBM,dc=UK
ldap://madison/cn=JMSData,dc=IBM,dc=UK
LDAP:///cn=JMSData,dc=IBM,dc=UK
```

### Kontekst WSS

W przypadku kontekstu WSS URL ma postać punktu końcowego usług Web Service z przedrostkiem `http://`.

Alternatywnie można użyć przedrostka `cosnaming://` lub `wsvc://`,

Te dwa przedrostki są interpretowane jako oznaczające, że używany jest kontekst WSS z adresem URL dostępnym za pośrednictwem protokołu http, co umożliwia łatwe wyprowadzenie początkowego typu kontekstu bezpośrednio z adresu URL.

Przykładami tej składni są:

```
http://madison.ibm.com:9080/xmsjndi/services/JndiLookup
cosnaming://madison/jndilookup
```

## Usługa WWW wyszukiwania JNDI dla XMS .NET

Aby uzyskać dostęp do katalogu nazw COS z produktu XMS, na serwerze WebSphere Application Server service integration bus musi być wdrożona usługa WWW JNDI Lookup. Ta usługa WWW tłumaczy informacje Java z usługi nazw COS na formularz, który może zostać odczytany przez aplikacje XMS .

Usługa Web Service jest udostępniana w pliku archiwum korporacyjnego SIBXJndiLookupEAR.earznajdującym się w katalogu instalacyjnym. W przypadku bieżącej wersji produktu IBM MQ Message Service Client (XMS) for .NETplik SIBXJndiLookupEAR.ear znajduje się w katalogu *install_dir*\java\lib . Można go zainstalować na serwerze WebSphere Application Server service integration bus za pomocą Konsoli administracyjnej lub narzędzia skryptowego wsaadmin. Więcej informacji na temat wdrażania aplikacji usług Web Service można znaleźć w dokumentacji produktu.

Aby zdefiniować usługę WWW w aplikacjach XMS , wystarczy ustawić właściwość XMSC_IC_URL obiektu InitialContext na URLpunktu końcowego usługi WWW. Jeśli na przykład usługa Web Service jest wdrożona na hoście serwera o nazwie MyServer, przykładowy punkt końcowy usługi Web Service URL:

```
wsvc://MyHost:9080/SIBXJndiLookup/services/JndiLookup
```

Ustawienie właściwości XMSC_IC_URL umożliwia wywoływanie usług Web Service przez wywołania wyszukania InitialContext w zdefiniowanym punkcie końcowym, który z kolei wyszukuje wymagany administrowany obiekt w usłudze nazw COS.

Aplikacje .NET mogą korzystać z usługi Web Service. Wdrożenie po stronie serwera jest takie samo w przypadku języków XMS C, /C++ i XMS .NET. XMS Produkt .NET wywołuje usługę WWW bezpośrednio za pośrednictwem serwera Microsoft .NET Framework.

## XMS .NET pobieranie obiektów administrowanych

Program XMS pobiera administrowany obiekt z repozytorium przy użyciu adresu podanego podczas tworzenia obiektu InitialContext lub we właściwościach InitialContext .

Obiekty do pobrania mogą mieć następujące typy nazw:

- Prosta nazwa opisująca obiekt docelowy, na przykład miejsce docelowe kolejki o nazwie SalesOrders
- Nazwa złożona, która może składać się z elementów SubContextsoddzielonych znakiem '/' i musi kończyć się nazwą obiektu. Przykładowa nazwa złożona to "Warehouse/PickLists/DispatchQueue2", gdzie Warehouse i Picklists są SubContexts w katalogu nazw, a DispatchQueue2 jest nazwą obiektu docelowego.

# Zapobieganie używaniu przez aplikacje nowszej wersji produktu XMS

Domyślnie po zainstalowaniu nowszej wersji produktu XMS aplikacje używające poprzedniej wersji automatycznie przełączają się na nowszą wersję bez konieczności recompile.Howevermożna uniemożliwić aplikacjom korzystanie z nowszej wersji, ustawiając atrybut w pliku konfiguracyjnym aplikacji.

### O tym zadaniu

Funkcja współistnienia wielu wersji zapewnia, że instalacja nowszej wersji produktu XMS nie spowoduje nadpisania poprzedniej wersji produktu XMS . Zamiast tego wiele instancji podobnych zespołów XMS .NET współistnieje w globalnej pamięci podręcznej zespołu (GAC), ale mają różne numery wersji. Wewnętrznie GAC używa pliku strategii do kierowania wywołań aplikacji do najnowszej wersji produktu XMS. Aplikacje działają bez konieczności rekompilacji i mogą korzystać z nowych funkcji dostępnych w nowszej wersji produktu XMS .NET .

### Procedura

- Jeśli aplikacja jest wymagana do korzystania ze starszej wersji produktu XMS .NET , należy ustawić atrybut `publisherpolicy` na wartość no w pliku konfiguracyjnym aplikacji.

**Uwaga:** Plik konfiguracyjny aplikacji to plik o nazwie składającej się z nazwy programu wykonywalnego, do którego odnosi się plik, z przyrostkiem .config. Na przykład plik konfiguracyjny aplikacji dla pliku text.exe będzie miał nazwę text.exe.config.

Jednak w dowolnym momencie wszystkie aplikacje systemu używają tej samej wersji produktu XMS .NET.

# Zabezpieczanie komunikacji dla aplikacji XMS

Ta sekcja zawiera informacje na temat konfigurowania bezpiecznej komunikacji w celu umożliwienia aplikacjom XMS nawiązywania połączeń SSL (Secure Sockets Layer) z mechanizmem przesyłania komunikatów WebSphere Application Server service integration bus lub menedżerem kolejek systemu IBM MQ .

## O tym zadaniu

Sekcja zawiera następujące tematy:

- “Bezpieczne połączenia z menedżerem kolejek produktu IBM MQ” na stronie 686
- “Odwzorowania nazw CipherSuite i CipherSpec dla połączeń produktu XMS z menedżerem kolejek produktu IBM MQ” na stronie 687
- “Bezpieczne połączenia z mechanizmem przesyłania komunikatów WebSphere Application Server service integration bus” na stronie 687
- “Odwzorowania nazw CipherSuite i CipherSpec dla połączeń z serwerem WebSphere Application Server service integration bus” na stronie 688

## Bezpieczne połączenia z menedżerem kolejek produktu IBM MQ

Aby umożliwić aplikacji produktu XMS .NET nawiązywanie bezpiecznych połączeń z menedżerem kolejek produktu IBM MQ , odpowiednie właściwości muszą być zdefiniowane w obiekcie ConnectionFactory .

Protokół używany w negocjacjach szyfrowania może być protokołem SSL (Secure Sockets Layer) lub TLS (Transport Layer Security), w zależności od tego, który CipherSuite został określony w obiekcie ConnectionFactory .

W poniższej tabeli przedstawiono właściwości fabryki połączeń ConnectionFactory w przypadku połączeń SSL z menedżerem kolejek IBM MQ z krótkim opisem:

*Tabela 93. Właściwości ConnectionFactory dla połączeń z menedżerem kolejek produktu IBM MQ za pośrednictwem protokołu SSL*

| Nazwa właściwości | Opis |
|---|---|
| XMSC_WMQ_SSL_CERT_STORES | Lokalizacje serwerów, na których znajdują się listy odwołań certyfikatów (Certificate Revocation List – CRL) używane podczas nawiązywania połączenia SSL z menedżerem kolejek. |
| XMSC_WMQ_SSL_CIPHER_SPEC | Nazwa specyfikacji szyfrowania używanej w przypadku bezpiecznego połączenia z menedżerem kolejek. |
| XMSC_WMQ_SSL_CIPHER_SUITE | Nazwa zestawu algorytmów szyfrowania używanego w przypadku połączenia TLS z menedżerem kolejek. Na podstawie podanego zestawu algorytmów szyfrowania jest określany protokół używany do negocjowania bezpiecznego połączenia. |
| XMSC_WMQ_SSL_CRYPTO_HW | Szczegóły konfiguracji sprzętu szyfrującego połączonego z systemem klienta. |

| *Tabela 93. Właściwości ConnectionFactory dla połączeń z menedżerem kolejek produktu IBM MQ za pośrednictwem protokołu SSL (kontynuacja)* | |
|---|---|
| **Nazwa właściwości** | **Opis** |
| XMSC_WMQ_SSL_FIPS_REQUIRED | Wartość tej właściwości określa, czy aplikacja może lub nie może używać zestawów algorytmów szyfrowania niezgodnych ze standardem FIPS. Jeśli ta właściwość zostanie ustawiona na wartość true, w przypadku połączeń klient-serwer będzie można używać tylko algorytmów zgodnych ze standardem FIPS. |
| XMSC_WMQ_SSL_KEY_REPOSITORY | Położenie pliku bazy danych kluczy, w którym przechowywane są klucze i certyfikaty. |
| XMSC_WMQ_SSL_KEY_RESETCOUNT | Wartość KeyResetCount reprezentuje całkowitą liczbę nieszyfrowanych bajtów wysłanych i odebranych w ramach konwersacji SSL przed ponownym wynegocjowaniem klucza tajnego. |
| XMSC_WMQ_SSL_PEER_NAME | Nazwa węzła sieci używana na potrzeby połączenia SSL z menedżerem kolejek. |

### *Odwzorowania nazw CipherSuite i CipherSpec dla połączeń produktu XMS z menedżerem kolejek produktu IBM MQ*

Wartość InitialContext jest tłumaczona na wartość właściwości SSLCIPHERSUITE fabryki połączeń JMSAdmin i prawie równoważnej właściwości XMSC_WMQ_SSL_CIPHER_SPEC bazy danych XMS . Podobne tłumaczenie jest konieczne, jeśli określono wartość właściwości XMSC_WMQ_SSL_CIPHER_SUITE, ale pominięto wartość właściwości XMSC_WMQ_SSL_CIPHER_SPEC.

Tabela 94 na stronie 687 zawiera listę dostępnych CipherSpecs i ich odpowiedników w pakiecie JSSE CipherSuite .

| *Tabela 94. Dostępne CipherSpecs i ich odpowiedniki w pakiecie JSSE CipherSuite* | |
|---|---|
| **CipherSpec** | **Odpowiednik 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 |

**Uwaga:** Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA jest nieaktualna. Jednak nadal może być używany do przesyłania do 32 GB danych, zanim połączenie zostanie przerwane i zostanie zgłoszony błąd AMQ9288. Aby uniknąć tego błędu, należy unikać używania potrójnego algorytmu szyfrowania DES lub włączyć resetowanie klucza tajnego podczas korzystania z tej CipherSpec.

## Bezpieczne połączenia z mechanizmem przesyłania komunikatów WebSphere Application Server service integration bus

Aby umożliwić aplikacji XMS .NET nawiązywanie bezpiecznych połączeń z mechanizmem przesyłania komunikatów WebSphere Application Server service integration bus , odpowiednie właściwości muszą być zdefiniowane w obiekcie ConnectionFactory .

XMS udostępnia obsługę protokołów SSL i HTTPS dla połączeń z WebSphere Application Server service integration bus. SSL i HTTPS zapewniają bezpieczne połączenia na potrzeby uwierzytelniania i poufności.

Podobnie jak zabezpieczenia produktu WebSphere , zabezpieczenia produktu XMS są konfigurowane zgodnie ze standardami bezpieczeństwa JSSE i konwencjami nazewnictwa, które obejmują użycie opcji

CipherSuites w celu określenia algorytmów używanych podczas negocjowania bezpiecznego połączenia. Protokół używany w negocjacjach szyfrowania może być protokołem SSL lub TLS, w zależności od tego, który CipherSuite został określony w obiekcie ConnectionFactory .

Tabela 95 na stronie 688 zawiera listę właściwości, które muszą być zdefiniowane w obiekcie ConnectionFactory .

*Tabela 95. Właściwości ConnectionFactory na potrzeby bezpiecznych połączeń z mechanizmem przesyłania komunikatów produktu WebSphere Application Server service integration bus*

| **Nazwa właściwości** | **Opis** |
|---|---|
| XMSC_WPM_SSL_CIPHER_SUITE | Nazwa zestawu algorytmów szyfrowania CipherSuite , który ma być używany w połączeniu TLS z mechanizmem przesyłania komunikatów produktu WebSphere Application Server service integration bus . Na podstawie podanego zestawu algorytmów szyfrowania jest określany protokół używany do negocjowania bezpiecznego połączenia. |
| XMSC_WPM_SSL_KEYRING_LABEL | Certyfikat używany podczas uwierzytelniania na serwerze. |

Poniżej przedstawiono przykład właściwości ConnectionFactory w przypadku bezpiecznych połączeń z mechanizmem przesyłania komunikatów WebSphere Application Server service integration bus :

```
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);
```

Gdzie nazwa_łańcucha powinna być ustawiona na BootstrapTunneledSecureMessaging lub BootstrapSecureMessaging, a numer_portu jest numerem portu, na którym serwer startowy nasłuchuje żądań przychodzących.

Poniżej przedstawiono przykład właściwości ConnectionFactory w przypadku bezpiecznych połączeń z mechanizmem przesyłania komunikatów WebSphere Application Server service integration bus z wstawionymi przykładowymi wartościami:

```
  /* 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");
```

### *Odwzorowania nazw CipherSuite i CipherSpec dla połączeń z serwerem WebSphere Application Server service integration bus*

Ponieważ produkt IBM Global Security Kit (GSKit) używa CipherSpecs , a nie CipherSuites, nazwy CipherSuite w stylu JSSE określone we właściwości XMSC_WPM_SSL_CIPHER_SUITE muszą być odwzorowane na nazwy CipherSpec w stylu GSKit-style.

Tabela 96 na stronie 688 zawiera listę równoważnych CipherSpec dla każdego rozpoznanego pakietu CipherSuite.

*Tabela 96. Dostępne CipherSuites i odpowiadające im CipherSpecs*

| **CipherSuite** | **Odpowiednik 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 |

| *Tabela 96. Dostępne CipherSuites i odpowiadające im CipherSpecs (kontynuacja)* | |
|---|---|
| **CipherSuite** | **Odpowiednik CipherSpec** |
| 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 |

**Uwaga:** Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA jest nieaktualna. Jednak nadal może być używany do przesyłania do 32 GB danych, zanim połączenie zostanie przerwane i zostanie zgłoszony błąd AMQ9288. Aby uniknąć tego błędu, należy unikać używania potrójnego algorytmu szyfrowania DES lub włączyć resetowanie klucza tajnego podczas korzystania z tej CipherSpec.

# Komunikaty produktu XMS

W tej sekcji opisano strukturę i treść komunikatów XMS oraz wyjaśniono, w jaki sposób aplikacje przetwarzają komunikaty XMS .

Sekcja obejmuje następujące tematy:

- “Części komunikatu XMS” na stronie 689
- “Pola nagłówka w komunikacie XMS” na stronie 689
- “Właściwości komunikatu XMS” na stronie 690
- “Treść komunikatu XMS” na stronie 693
- “Selektory komunikatów” na stronie 696
- “Odwzorowywanie komunikatów XMS na komunikaty IBM MQ” na stronie 697

## Części komunikatu XMS

Komunikat XMS składa się z nagłówka, zestawu właściwości i treści.

**Nagłówek**
Nagłówek komunikatu zawiera pola, a wszystkie komunikaty zawierają ten sam zestaw pól nagłówka. Produkt XMS i aplikacje używają wartości pól nagłówka do identyfikowania i kierowania komunikatów. Więcej informacji na temat pól nagłówka zawiera sekcja “Pola nagłówka w komunikacie XMS” na stronie 689.

**Zestaw właściwości**
Właściwości komunikatu określają dodatkowe informacje o komunikacie. Mimo że wszystkie komunikaty mają ten sam zestaw pól nagłówka, każdy komunikat może mieć inny zestaw właściwości. Więcej informacji na ten temat zawiera sekcja “Właściwości komunikatu XMS” na stronie 690.

**Treść**
Treść komunikatu zawiera dane aplikacji. Więcej informacji na ten temat zawiera sekcja “Treść komunikatu XMS” na stronie 693.

Aplikacja może wybrać komunikaty, które chce odbierać. Przy użyciu selektorów komunikatów, które określają kryteria wyboru. Kryteria mogą być oparte na wartościach określonych pól nagłówka i wartościach dowolnych właściwości komunikatu. Więcej informacji na temat selektorów komunikatów zawiera sekcja “Selektory komunikatów” na stronie 696.

## Pola nagłówka w komunikacie XMS

Aby umożliwić aplikacji XMS wymianę komunikatów z aplikacją WebSphere JMS , nagłówek komunikatu XMS zawiera pola nagłówka komunikatu JMS .

Nazwy tych pól nagłówka rozpoczynają się od przedrostka JMS. Opis pól nagłówka komunikatu JMS zawiera sekcja *Java Message Service Specyfikacja*.

XMS implementuje pola nagłówka komunikatu JMS jako atrybuty obiektu Message. Każde pole nagłówka ma własne metody ustawiania i pobierania wartości. Opis tych metod zawiera sekcja IMessage. Pole nagłówka jest zawsze dostępne do odczytu i zapisu.

Tabela 97 na stronie 690 zawiera listę pól nagłówka komunikatu JMS i wskazuje, w jaki sposób wartość każdego pola jest ustawiana dla przesyłanego komunikatu. Niektóre pola są ustawiane automatycznie przez produkt XMS po wysłaniu komunikatu przez aplikację lub, w przypadku pola JMSRedelivered, po odebraniu komunikatu przez aplikację.

*Tabela 97. Pola nagłówka komunikatu JMS.* ]

| **Nazwa pola nagłówka komunikatu JMS** | **Sposób ustawiania wartości dla przesyłanego komunikatu (w formacie *metoda* [*klasa*])** |
|---|---|
| JMSCorrelationID | Ustaw JMSCorrelationID [ Komunikat] |
| JMSDeliveryMode | Send [MessageProducer] |
| JMSDestination | Send [MessageProducer] |
| JMSExpiration | Send [MessageProducer] |
| JMSMessageID | Send [MessageProducer] |
| JMSPriority | Send [MessageProducer] |
| JMSRedelivered | Receive [MessageConsumer] |
| JMSReplyTo | Set JMSReplyTo [ komunikat] |
| JMSTimestamp | Send [MessageProducer] |
| JMSType | Ustaw JMSType [ Message] |

## Właściwości komunikatu XMS

Produkt XMS obsługuje trzy rodzaje właściwości komunikatu: JMS zdefiniowane właściwości, IBM zdefiniowane właściwości i właściwości zdefiniowane przez aplikację.

Aplikacja XMS może wymieniać komunikaty z aplikacją WebSphere JMS , ponieważ produkt XMS obsługuje następujące predefiniowane właściwości obiektu Message:

- Te same właściwości zdefiniowane w pliku JMS, które są obsługiwane przez produkt WebSphere JMS . Nazwy tych właściwości rozpoczynają się przedrostkiem JMSX.
- Te same właściwości zdefiniowane w pliku IBM, które są obsługiwane przez produkt WebSphere JMS . Nazwy tych właściwości zaczynają się od przedrostka JMS_IBM_.

Każda predefiniowana właściwość ma dwie nazwy:

- Nazwa JMS dla właściwości definiowanej przez JMSlub nazwa WebSphere JMS dla właściwości definiowanej przez IBM.

  Jest to nazwa, pod którą właściwość jest znana w produkcie JMS lub WebSphere JMS, a także nazwa, która jest przesyłana z komunikatem, który zawiera tę właściwość. Aplikacja XMS używa tej nazwy do identyfikowania właściwości w wyrażeniu selektora komunikatów.
- Nazwa XMS identyfikująca właściwość we wszystkich sytuacjach z wyjątkiem wyrażenia selektora komunikatów. Każda nazwa XMS jest zdefiniowana jako stała nazwana w klasie `IBM.XMS.XMSC` . Wartością stałej nazwanej jest odpowiadająca jej nazwa JMS lub WebSphere JMS .

Oprócz predefiniowanych właściwości aplikacja XMS może tworzyć i używać własnego zestawu właściwości komunikatu. Te właściwości są nazywane *właściwościami zdefiniowanymi przez aplikację* .

Po utworzeniu komunikatu przez aplikację jego właściwości są dostępne do odczytu i zapisu. Po wysłaniu komunikatu przez aplikację właściwości pozostają dostępne do odczytu i zapisu. Gdy aplikacja odbiera komunikat, właściwości komunikatu są tylko do odczytu. Jeśli aplikacja wywołuje metodę `Clear`

`Properties` klasy Message, gdy właściwości komunikatu są tylko do odczytu, właściwości te stają się dostępne do odczytu i zapisu. Metoda usuwa również właściwości.

Odebrany komunikat, przekazywany po wyczyszczeniu właściwości komunikatu, będzie działać w sposób spójny z zachowaniem przekazywania standardowego produktu WMQ XMS for .NET BytesMessage z wyczyszczonymi właściwościami komunikatu.

Nie jest to jednak zalecane, ponieważ następujące właściwości zostaną utracone:

- Wartość właściwości JMS_IBM_Encoding oznaczająca, że dane komunikatu nie mogą być dekodowane w sposób sensowny.
- Wartość właściwości JMS_IBM_Format oznaczająca, że łańcuch nagłówka między nagłówkiem komunikatu (MQMD lub nowym MQRFH2) a istniejącymi nagłówkami zostanie uszkodzony.

Aby określić wartości wszystkich właściwości komunikatu, aplikacja może wywołać metodę Get Properties klasy Message. Metoda tworzy iterator, który hermetyzuje listę obiektów Property, gdzie każdy obiekt Property reprezentuje właściwość komunikatu. Aplikacja może następnie użyć metod klasy Iterator, aby pobrać kolejno każdy obiekt Property, a także może użyć metod klasy Property, aby pobrać nazwę, typ danych i wartość każdej właściwości.

### *JMS-zdefiniowane właściwości komunikatu*

Niektóre właściwości komunikatu zdefiniowane w usłudze JMS są obsługiwane zarówno przez produkt XMS , jak i przez produkt WebSphere JMS.

Tabela 98 na stronie 691 zawiera listę zdefiniowanych w usłudze JMS właściwości komunikatu, które są obsługiwane zarówno przez produkt XMS , jak i przez produkt WebSphere JMS. Opis właściwości zdefiniowanych przez JMSznajduje się w sekcji *Java Message Service Specyfikacja*. Właściwości zdefiniowane w pliku JMSnie są poprawne dla połączenia z brokerem w czasie rzeczywistym.

Tabela określa typ danych każdej właściwości i wskazuje, w jaki sposób wartość właściwości jest ustawiana dla przesyłanego komunikatu. Niektóre właściwości są ustawiane automatycznie przez produkt XMS po wysłaniu komunikatu przez aplikację lub, w przypadku wartości JMSXDeliveryCount, po odebraniu komunikatu przez aplikację.

*Tabela 98. JMS-zdefiniowane właściwości komunikatu*

| **Nazwa XMS zdefiniowanej właściwości JMS** | **Nazwa JMS** | **Typ danych** | **Sposób ustawiania wartości dla przesyłanego komunikatu (w formacie *metoda* [*klasa*])** |
|---|---|---|---|
| JMSX_APPID | JMSXAppID | System.String | Send [MessageProducer] |
| JMSX_DELIVERY_COUNT (licznik ogranicznik_JMS) | JMSXDeliveryCount | System.Int32 | Receive [MessageConsumer] |
| ID_grupy_JMS | JMSXGroupID | System.String | Ustaw właściwość łańcuchową [PropertyContext] |
| JMSX_GROUPSEQ, | JMSXGroupSeq | System.Int32 | Ustaw właściwość całkowitą [PropertyContext] |
| ID_UŻYTKOWNIKA_JMS | JMSXUserID | System.String | Send [MessageProducer] |

### *IBM-zdefiniowane właściwości komunikatu*

Kilka właściwości komunikatu zdefiniowanych przez IBMjest obsługiwanych przez produkty XMS i WebSphere JMS.

Tabela 99 na stronie 692 zawiera listę IBM zdefiniowanych właściwości komunikatu, które są obsługiwane zarówno przez XMS , jak i WebSphere JMS. Więcej informacji na temat właściwości zdefiniowanych przez IBMzawiera dokumentacja produktu IBM MQ lub WebSphere Application Server .

Tabela określa typ danych każdej właściwości i wskazuje, w jaki sposób wartość właściwości jest ustawiana dla przesyłanego komunikatu. Niektóre właściwości są ustawiane automatycznie przez program XMS , gdy aplikacja wysyła komunikat.

*Tabela 99. IBM-zdefiniowane właściwości komunikatu*

| **XMS nazwa zdefiniowanej właściwości IBM** | **Nazwa WebSphereJMS** | **Typ danych** | **Sposób ustawiania wartości dla przesyłanego komunikatu (w formacie *metoda* [*klasa*])** |
|---|---|---|---|
| JMS_IBM_CHARACTER_SET (zestaw znaków IMS) | Zestaw znaków JMS_IBM_Character_Set | System.Int32 | Ustaw właściwość całkowitą [PropertyContext] |
| JMS_IBM_Encoding | JMS_IBM_Encoding | System.Int32 | Ustaw właściwość całkowitą [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_EXCEPTIONPROBLEM DESTINATION | Miejsce docelowe pakietu JMS_IBM_ExceptionProblem | System.String | Receive [MessageConsumer] |
| JMS_IBM_FEEDBACK, | JMS_IBM_Feedback, | System.Int32 | Ustaw właściwość całkowitą [PropertyContext] |
| JMS_IBM_FORMAT, | JMS_IBM_Format | System.String | Ustaw właściwość łańcuchową [PropertyContext] |
| JMS_IBM_LAST_MSG_IN_GROUP | JMS_IBM_Last_Msg_In_Group | System.Boolean | Ustaw właściwość całkowitą [PropertyContext] |
| JMS_IBM_MSGTYPE | JMS_IBM_MsgType | System.Int32 | Ustaw właściwość całkowitą [PropertyContext] |
| JMS_IBM_PUTAPPLTYPE | Typ JMS_IBM_PutAppl | System.Int32 | Send [MessageProducer] |
| JMS_IBM_PUTDATE, | JMS_IBM_PutDate | System.String | Send [MessageProducer] |
| CZAS JMS_IBM_PUTTIME | JMS_IBM_PutTime | System.String | Send [MessageProducer] |
| JMS_IBM_REPORT_COA | JMS_IBM_Report_COA | System.Int32 | Ustaw właściwość całkowitą [PropertyContext] |

| *Tabela 99. IBM-zdefiniowane właściwości komunikatu (kontynuacja)* | | | |
|---|---|---|---|
| **XMS nazwa zdefiniowanej właściwości IBM** | **Nazwa WebSphereJMS** | **Typ danych** | **Sposób ustawiania wartości dla przesyłanego komunikatu (w formacie *metoda* [*klasa*])** |
| JMS_IBM_REPORT_COD | JMS_IBM_Report_COD | System.Int32 | Ustaw właściwość całkowitą [PropertyContext] |
| JMS_IBM_REPORT_DISCARD_MSG | JMS_IBM_Report_Discard_Msg, | System.Int32 | Ustaw właściwość całkowitą [PropertyContext] |
| JMS_IBM_REPORT_EXCEPTION, | Wyjątek JMS_IBM_Report_Exception | System.Int32 | Ustaw właściwość całkowitą [PropertyContext] |
| JMS_IBM_REPORT_EXPIRATION | Utrata ważności raportu JMS_IBM_Report_Expiration | System.Int32 | Ustaw właściwość całkowitą [PropertyContext] |
| JMS_IBM_REPORT_NAN | JMS_IBM_Report_NAN | System.Int32 | Ustaw właściwość całkowitą [PropertyContext] |
| JMS_IBM_REPORT_PAN | Raport JMS_IBM_Report_PAN | System.Int32 | Ustaw właściwość całkowitą [PropertyContext] |
| JMS_IBM_REPORT_PASS_CORREL_ Identyfikator | JMS_IBM_Report_Pass_Correl_ID (Identyfikator korelacji) | System.Int32 | Ustaw właściwość całkowitą [PropertyContext] |
| JMS_IBM_REPORT_PASS_MSG_ID | JMS_IBM_Report_Pass_Msg_ID | System.Int32 | Ustaw właściwość całkowitą [PropertyContext] |
| JMS_IBM_SYSTEM_MESSAGEID | JMS_IBM_System_MessageID | System.String | Send [MessageProducer] |

### *Właściwości komunikatu definiowane przez aplikację*

Aplikacja XMS może tworzyć i używać własnego zestawu właściwości komunikatu. Gdy aplikacja wysyła komunikat, właściwości te są również przesyłane razem z komunikatem. Aplikacja odbierająca, za pomocą selektorów komunikatów, może wybrać komunikaty, które ma otrzymywać, na podstawie wartości tych właściwości.

Aby umożliwić aplikacji produktu WebSphere JMS wybieranie i przetwarzanie komunikatów wysyłanych przez aplikację XMS , nazwa właściwości zdefiniowanej przez aplikację musi być zgodna z regułami tworzenia identyfikatorów w wyrażeniach selektora komunikatów. Więcej informacji na ten temat zawiera sekcja “Selektory komunikatów w produkcie JMS” na stronie 150. Wartość właściwości zdefiniowanej przez aplikację musi mieć jeden z następujących typów danych: System.Boolean, System.SByte, System.Int16, System.Int32, System.Int64, System.Float, System.Doublelub System.String.

## Treść komunikatu XMS

Treść komunikatu zawiera dane aplikacji. Jednak komunikat nie może mieć treści i może składać się tylko z pól i właściwości nagłówka.

Produkt XMS obsługuje pięć typów treści komunikatu:

**Bajty**
Treść zawiera strumień bajtów. Komunikat o treści tego typu jest nazywany *komunikatem bajtowym*. Interfejs IBytesMessage zawiera metody przetwarzania treści komunikatu bajtowego.

**Odwzoruj**
Treść zawiera zestaw par nazwa-wartość, z którymi każda wartość ma powiązany typ danych. Komunikat o treści tego typu jest nazywany *komunikatem odwzorowania*. Interfejs IMapMessage zawiera metody do przetwarzania treści komunikatu odwzorowania.

**Obiekt**
Treść zawiera przekształcony do postaci szeregowej obiekt Java lub .NET . Komunikat o treści tego typu jest nazywany *komunikatem obiektu*. Interfejs IObjectMessage zawiera metody służące do przetwarzania treści komunikatu obiektu.

**Strumień**
Treść zawiera strumień wartości, z którym każda wartość ma powiązany typ danych. Komunikat o treści tego typu jest nazywany *komunikatem strumienia*. Interfejs IStreamMessage zawiera metody służące do przetwarzania treści komunikatu strumienia.

**Tekst**
Treść zawiera łańcuch. Komunikat o treści tego typu jest nazywany *komunikatem tekstowym*. Interfejs ITextMessage zawiera metody przetwarzania treści komunikatu tekstowego.

Interfejs IMessage jest elementem nadrzędnym wszystkich obiektów komunikatów i może być używany w funkcjach przesyłania komunikatów do reprezentowania dowolnego typu komunikatów produktu XMS .

Informacje na temat wielkości oraz maksymalnej i minimalnej wartości każdego z tych typów danych zawiera sekcja Tabela 84 na stronie 668.

## Komunikaty bajtowe

Treść komunikatu bajtowego zawiera strumień bajtów. Treść zawiera tylko rzeczywiste dane i to aplikacje wysyłające i odbierające są odpowiedzialne za interpretowanie tych danych.

Komunikaty bajtowe są przydatne, gdy aplikacja XMS musi wymieniać komunikaty z aplikacjami, które nie korzystają z aplikacyjnego interfejsu programistycznego produktu XMS lub JMS .

Po utworzeniu przez aplikację komunikatu bajtowego treść komunikatu jest tylko do zapisu. Aplikacja składa dane aplikacji w treści, wywołując odpowiednie metody zapisu interfejsu IBytesMessage dla .NET. Za każdym razem, gdy aplikacja zapisuje wartość w strumieniu komunikatów bajtowych, wartość ta jest składana bezpośrednio po poprzedniej wartości zapisanej przez aplikację. XMS utrzymuje kursor wewnętrzny w celu zapamiętania pozycji ostatnio złożonego bajtu.

Gdy aplikacja wysyła komunikat, jego treść staje się tylko do odczytu. W tym trybie aplikacja może wysyłać komunikat wielokrotnie.

Gdy aplikacja odbiera komunikat bajtowy, treść komunikatu jest tylko do odczytu. Aplikacja może użyć odpowiednich metod odczytu interfejsu IBytesMessage w celu odczytania treści strumienia komunikatów bajtów. Aplikacja odczytuje kolejno bajty, a XMS utrzymuje kursor wewnętrzny, aby zapamiętać pozycję ostatniego odczytanego bajtu.

Jeśli aplikacja wywołuje metodę Reset interfejsu IBytesMessage , gdy treść komunikatu bajtowego jest dostępna do zapisu, treść staje się dostępna tylko do odczytu. Metoda zmienia również położenie kursora na początku strumienia komunikatów bajtów.

Jeśli aplikacja wywołuje metodę `Clear Body` interfejsu IMessage interfejsu .NET , gdy treść komunikatu bajtowego jest tylko do odczytu, treść staje się dostępna do zapisu. Metoda również usuwa zawartość ciała.

## Komunikaty odwzorowania

Treść komunikatu odwzorowania zawiera zestaw par nazwa-wartość, z którymi każda wartość ma powiązany typ danych.

W każdej parze nazwa-wartość nazwa jest łańcuchem, który identyfikuje wartość, a wartość jest elementem danych aplikacji, który ma jeden z typów danych XMS wymienionych w tabeli Tabela 100 na stronie 696. Kolejność par nazwa-wartość nie jest zdefiniowana. Klasa MapMessage zawiera metody służące do ustawiania i pobierania par nazwa-wartość.

Aplikacja może uzyskać dostęp do pary nazwa-wartość losowo, podając jej nazwę.

Aplikacja .NET może użyć właściwości MapNames , aby uzyskać wyliczenie nazw w treści komunikatu odwzorowania.

Gdy aplikacja pobiera wartość z pary nazwa-wartość, wartość może zostać przekształcona przez XMS w inny typ danych. Na przykład, aby uzyskać liczbę całkowitą z treści komunikatu odwzorowania, aplikacja może wywołać metodę GetString klasy MapMessage , która zwraca liczbę całkowitą jako łańcuch. Obsługiwane konwersje są takie same, jak te, które są obsługiwane, gdy program XMS przekształca wartość właściwości z jednego typu danych na inny. Więcej informacji na temat obsługiwanych konwersji zawiera sekcja “Niejawna konwersja wartości właściwości z jednego typu danych do innego” na stronie 669.

Po utworzeniu przez aplikację komunikatu odwzorowania treść komunikatu jest dostępna do odczytu i zapisu. Po wysłaniu komunikatu przez aplikację treść pozostaje dostępna do odczytu i zapisu. Gdy aplikacja odbiera komunikat odwzorowania, treść komunikatu jest tylko do odczytu. Jeśli aplikacja wywołuje metodę Clear Body klasy Message, gdy treść komunikatu odwzorowania jest tylko do odczytu, treść staje się czytelna i dostępna do zapisu. Metoda również usuwa zawartość ciała.

## Komunikaty obiektów

Treść komunikatu obiektu zawiera przekształcony do postaci szeregowej obiektJava lub .NET .

Aplikacja XMS może odebrać komunikat obiektu, zmienić jej pola nagłówka i właściwości, a następnie wysłać go do innego miejsca docelowego. Aplikacja może również skopiować treść komunikatu obiektu i użyć jej do utworzenia innego komunikatu obiektu. Program XMS traktuje treść komunikatu obiektu jako tablicę bajtów.

Po utworzeniu komunikatu obiektu przez aplikację jego treść jest dostępna do odczytu i zapisu. Po wysłaniu komunikatu przez aplikację treść pozostaje dostępna do odczytu i zapisu. Gdy aplikacja odbiera komunikat obiektu, treść komunikatu jest tylko do odczytu. Jeśli aplikacja wywołuje metodę `Clear Body` interfejsu IMessage interfejsu .NET , gdy treść komunikatu obiektu jest tylko do odczytu, treść staje się dostępna do odczytu i zapisu. Metoda również usuwa zawartość ciała.

## Komunikaty strumienia

Treść komunikatu strumienia zawiera strumień wartości, z którym każda wartość ma powiązany typ danych.

Typ danych wartości to jeden z typów danych XMS wymienionych w tabeli Tabela 100 na stronie 696.

Po utworzeniu przez aplikację komunikatu strumienia treść komunikatu jest dostępna do zapisu. Aplikacja składa dane aplikacji w treści, wywołując odpowiednie metody zapisu interfejsu IStreamMessage dla .NET. Za każdym razem, gdy aplikacja zapisuje wartość w strumieniu komunikatów, wartość i jej typ danych są składane bezpośrednio po poprzedniej wartości zapisanej przez aplikację. XMS utrzymuje kursor wewnętrzny w celu zapamiętania pozycji ostatnio złożonej wartości.

Gdy aplikacja wysyła komunikat, jego treść staje się tylko do odczytu. W tym trybie aplikacja może wysyłać komunikat wiele razy.

Gdy aplikacja odbiera komunikat strumienia, treść komunikatu jest tylko do odczytu. Aplikacja może użyć odpowiednich metod odczytu interfejsu IStreamMessage dla .NET w celu odczytania treści strumienia komunikatów. Aplikacja odczytuje wartości w kolejności, a XMS utrzymuje kursor wewnętrzny w celu zapamiętania pozycji ostatnio odczytanej wartości.

Gdy aplikacja odczytuje wartość ze strumienia komunikatów, wartość może zostać przekształcona przez program XMS w inny typ danych. Na przykład, aby odczytać liczbę całkowitą ze strumienia komunikatów, aplikacja może wywołać metodę ReadString , która zwraca liczbę całkowitą jako łańcuch.

Obsługiwane konwersje są takie same, jak te, które są obsługiwane, gdy program XMS przekształca wartość właściwości z jednego typu danych na inny. Więcej informacji na temat obsługiwanych konwersji zawiera sekcja “Niejawna konwersja wartości właściwości z jednego typu danych do innego” na stronie 669.

Jeśli podczas próby odczytania przez aplikację wartości ze strumienia komunikatów wystąpi błąd, kursor nie jest zaawansowany. Aplikacja może naprawić błąd, próbując odczytać wartość jako inny typ danych.

Jeśli aplikacja wywołuje metodę Reset interfejsu IStreamMessage dla XMS , gdy treść komunikatu strumienia jest tylko do zapisu, treść staje się tylko do odczytu. Metoda zmienia również pozycję kursora na początku strumienia komunikatów.

Jeśli aplikacja wywołuje metodę `Clear Body` interfejsu IMessage interfejsu XMS , gdy treść komunikatu strumienia jest tylko do odczytu, treść staje się tylko do zapisu. Metoda również usuwa zawartość ciała.

### Komunikaty tekstowe

Treść komunikatu tekstowego zawiera łańcuch.

Po utworzeniu komunikatu tekstowego przez aplikację jego treść jest dostępna do odczytu i zapisu. Po wysłaniu komunikatu przez aplikację treść pozostaje dostępna do odczytu i zapisu. Gdy aplikacja otrzymuje komunikat tekstowy, treść komunikatu jest tylko do odczytu. Jeśli aplikacja wywołuje metodę Clear Body interfejsu IMessage dla elementu .NET , gdy treść komunikatu tekstowego jest tylko do odczytu, treść staje się dostępna do odczytu i zapisu. Metoda również usuwa zawartość ciała.

### *Typy danych dla elementów danych aplikacji*

Aby zapewnić, że aplikacja XMS może wymieniać komunikaty z aplikacją IBM MQ classes for JMS , obie aplikacje muszą mieć możliwość interpretowania danych aplikacji w treści komunikatu w taki sam sposób.

Z tego powodu każdy element danych aplikacji zapisany w treści komunikatu przez aplikację XMS musi mieć jeden z typów danych wymienionych w sekcji Tabela 100 na stronie 696. Dla każdego typu danych tabela przedstawia zgodny typ danych Java . Produkt XMS udostępnia metody służące do zapisywania elementów danych aplikacji tylko z tymi typami danych.

*Tabela 100. Typy danych XMS kompatybilne z typami danych Java*

| XMS Typ danych | Reprezentuje | Zgodny typ danych Java |
|---|---|---|
| System.Boolean | Wartość boolowska true lub false | boolean (boolowskie) |
| System.Char16 | Znak dwubajtowy | char |
| System.SByte | 8-bitowa liczba całkowita ze znakiem | B |
| System.Int16 | 16-bitowa liczba całkowita ze znakiem | short |
| System.Int32 | 32-bitowa liczba całkowita ze znakiem | int |
| System.Int64 | 64-bitowa liczba całkowita ze znakiem | long |
| System.Float | Liczba zmiennopozycyjna ze znakiem | liczba zmiennopozycyjna |
| System.Double | Liczba zmiennopozycyjna podwójnej precyzji ze znakiem | double (podwójna) |
| System.String | Łańcuch znaków | Łańcuch |

Informacje na temat wielkości, wartości maksymalnej i wartości minimalnej dla każdego z tych typów danych zawiera sekcja “Typy podstawowe XMS” na stronie 668.

## Selektory komunikatów

Aplikacja XMS używa selektorów komunikatów do wybierania komunikatów, które mają być odbierane.

Gdy aplikacja tworzy konsument komunikatów, może powiązać wyrażenie selektora komunikatów z konsumentem. Wyrażenie selektora komunikatów określa kryteria wyboru.

Podczas nawiązywania połączenia przez aplikację z menedżerem kolejek produktu IBM WebSphere MQ 7.0 wybór komunikatów jest dokonywany po stronie menedżera kolejek. Program XMS nie dokonuje wyboru i po prostu dostarcza komunikat odebrany od menedżera kolejek, co zapewnia lepszą wydajność.

Aplikacja może utworzyć więcej niż jednego konsumenta komunikatów, z których każdy ma własne wyrażenie selektora komunikatów. Jeśli komunikat przychodzący spełnia kryteria wyboru więcej niż jednego konsumenta komunikatów, produkt XMS dostarcza komunikat do każdego z tych konsumentów.

Wyrażenie selektora komunikatów może odwoływać się do następujących właściwości komunikatu:

- Właściwości zdefiniowane przez JMS
- Właściwości zdefiniowane przez IBM
- Właściwości definiowane przez aplikację

Może również odwoływać się do następujących pól nagłówka komunikatu:

- JMSCorrelationID
- JMSDeliveryMode
- JMSMessageID
- JMSPriority
- JMSTimestamp
- JMSType

Wyrażenie selektora komunikatów nie może jednak odwoływać się do danych w treści komunikatu.

Poniżej przedstawiono przykład wyrażenia selektora komunikatów:

```
JMSPriority > 3 AND manufacturer = 'Jaguar' AND model in ('xj6','xj12')
```

Produkt XMS dostarcza komunikat do konsumenta za pomocą tego wyrażenia selektora komunikatów tylko wtedy, gdy komunikat ma priorytet większy niż 3. Właściwość zdefiniowana przez aplikację, producent, ma wartość `Jaguar;` , a inna właściwość zdefiniowana przez aplikację, model ma wartość `xj6` lub `xj12.` .

Reguły składni tworzenia wyrażenia selektora komunikatów w XMS są takie same, jak w IBM MQ classes for JMS. Więcej informacji na temat tworzenia wyrażenia selektora komunikatów zawiera dokumentacja produktu IBM MQ . Należy zauważyć, że w wyrażeniu selektora komunikatów nazwy właściwości zdefiniowanych przez JMSmuszą być nazwami JMS, a nazwy właściwości zdefiniowanych przez IBMmuszą być nazwami IBM MQ classes for JMS . Nazw XMS nie można używać w wyrażeniu selektora komunikatów.

## Odwzorowywanie komunikatów XMS na komunikaty IBM MQ

Pola i właściwości nagłówka JMS komunikatu XMS są odwzorowywane na pola w strukturach nagłówka komunikatu IBM MQ .

Gdy aplikacja XMS jest połączona z menedżerem kolejek produktu IBM MQ , komunikaty wysyłane do menedżera kolejek są odwzorowywane na komunikaty produktu IBM MQ w taki sam sposób, jak komunikaty produktu IBM MQ classes for JMS są odwzorowywane na komunikaty produktu IBM MQ w podobnych okolicznościach.

Jeśli właściwość XMSC_WMQ_TARGET_CLIENT obiektu docelowego jest ustawiona na wartość XMSC_WMQ_TARGET_DEST_JMS, pola nagłówka JMS i właściwości komunikatu wysyłanego do miejsca docelowego są odwzorowywane na pola w strukturach nagłówka MQMD i MQRFH2 komunikatu IBM MQ . Ustawienie właściwości XMSC_WMQ_TARGET_CLIENT w ten sposób zakłada, że aplikacja odbierająca komunikat może obsłużyć nagłówek MQRFH2 . Dlatego aplikacja odbierająca może być inną aplikacją XMS , aplikacją IBM MQ classes for JMS lub rodzimą aplikacją IBM MQ , która została zaprojektowana do obsługi nagłówka MQRFH2 .

Jeśli właściwość XMSC_WMQ_TARGET_CLIENT obiektu Destination jest ustawiona na wartość XMSC_WTARGET_DEST_MQ, pola nagłówka JMS i właściwości komunikatu wysyłanego do miejsca docelowego są odwzorowywane na pola w strukturze nagłówka MQMD komunikatu IBM MQ . Komunikat nie zawiera nagłówka MQRFH2 , a wszystkie pola i właściwości nagłówka JMS, których nie można odwzorować na pola w strukturze nagłówka MQMD, są ignorowane. Aplikacja, która odbiera komunikat, może być rodzimą aplikacją IBM MQ , która nie jest zaprojektowana do obsługi nagłówka MQRFH2 .

Komunikaty produktu IBM MQ odebrane z menedżera kolejek są odwzorowywane na komunikaty produktu XMS w taki sam sposób, jak komunikaty produktu IBM MQ są odwzorowywane na komunikaty produktu IBM MQ classes for JMS w podobnych okolicznościach.

Jeśli przychodzący komunikat IBM MQ ma nagłówek MQRFH2 , wynikowy komunikat XMS ma treść, której typ jest określany na podstawie wartości właściwości **Msd** znajdującej się w folderze `mcd` nagłówka MQRFH2 . Jeśli właściwość **Msd** nie występuje w nagłówku MQRFH2 lub jeśli komunikat IBM MQ nie ma nagłówka MQRFH2 , wynikowy komunikat XMS ma treść, której typ jest określony przez wartość pola *`Format`* w nagłówku MQMD. Jeśli pole *`Format`* jest ustawione na wartość MQFMT_STRING, komunikat XMS jest komunikatem tekstowym. W przeciwnym razie komunikat XMS jest komunikatem bajtowym. Jeśli komunikat IBM MQ nie ma nagłówka MQRFH2 , ustawiane są tylko te pola i właściwości nagłówka JMS, które mogą być wyprowadzane z pól w nagłówku MQMD.

Więcej informacji na temat odwzorowywania komunikatów IBM MQ classes for JMS na komunikaty IBM MQ zawiera sekcja “Odwzorowywanie komunikatów JMS na komunikaty IBM MQ” na stronie 153.

### *Odczytywanie i zapisywanie deskryptora komunikatu z aplikacji IBM MQ Message Service Client (XMS) for .NET*

Można uzyskać dostęp do wszystkich pól deskryptora komunikatu (MQMD) komunikatu IBM MQ z wyjątkiem pól StrucId i Version; pole BackoutCount może być odczytywane, ale nie zapisywane.

Atrybuty komunikatu udostępniane przez IBM MQ Message Service Client (XMS) for .NET ułatwiają aplikacjom XMS ustawianie pól MQMD oraz prowadzenie aplikacji IBM WebSphere MQ .

Podczas korzystania z przesyłania komunikatów w trybie publikowania/subskrypcji obowiązują pewne ograniczenia. Na przykład pola MQMD, takie jak MsgID i CorrelId, jeśli są ustawione, są ignorowane.

Funkcja jest również niedostępna, jeśli właściwość **PROVIDERVERSION** jest ustawiona na wartość 6.

### *Uzyskiwanie dostępu do danych komunikatu IBM MQ z aplikacji IBM MQ Message Service Client (XMS) for .NET*

Dostęp do pełnych danych komunikatu produktu IBM MQ , w tym do nagłówka MQRFH2 (jeśli istnieje) i innych nagłówków IBM MQ (jeśli istnieje) w aplikacji IBM MQ Message Service Client (XMS) for .NET , można uzyskać jako do treści komunikatu JMSBytesMessage.

Funkcja opisana w tym temacie jest dostępna tylko w przypadku nawiązywania połączenia z menedżerem kolejek w wersji IBM WebSphere MQ 7.0 lub nowszej, gdy dostawca przesyłania komunikatów produktu IBM MQ jest w trybie normalnym.

Właściwości obiektu docelowego określają, w jaki sposób aplikacja XMS uzyskuje dostęp do całego komunikatu produktu IBM MQ (w tym nagłówka MQRFH2 , jeśli jest obecny) jako do treści komunikatu JMSBytesMessage.

## ALW Tworzenie aplikacji klienckich AMQP

Obsługa interfejsu API AMQP przez produkt IBM MQ umożliwia administratorowi produktu IBM MQ utworzenie kanału AMQP. Po uruchomieniu kanał ten definiuje numer portu, który akceptuje połączenia z aplikacji klienckich AMQP.

Kanał AMQP można zainstalować w systemach AIX, Linux, and Windows . Nie jest on dostępny w systemach IBM i i z/OS.

Aplikacja kliencka AMQP 1.0 może nawiązać połączenie z menedżerem kolejek przy użyciu kanału AMQP.

## Tworzenie aplikacji przy użyciu biblioteki produktu Apache Qpid JMS

### Wprowadzenie

Biblioteka produktu Apache Qpid JMS używa protokołu AMQP 1.0 do udostępnienia implementacji specyfikacji JMS 2.

Produkt Apache Qpid JMS używa niektórych aspektów protokołu AMQP 1.0 w inny sposób niż interfejsy API przesyłania komunikatów produktu MQ Light . Produkt IBM MQ 9.2 dodaje obsługę kanałów produktu IBM MQ AMQP, aby aplikacje produktu Apache Qpid JMS mogły nawiązywać połączenie z produktem IBM MQ i wykonywać przesyłanie komunikatów w trybie publikowania/subskrypcji, w tym używać subskrypcji współużytkowanych.

V 9.3.0 Produkt IBM MQ 9.3 dodaje obsługę kanałów produktu IBM MQ AMQP, aby aplikacje produktu Apache Qpid JMS mogły nawiązywać połączenie z produktem IBM MQ i wykonywać przesyłanie komunikatów między punktami. Więcej informacji zawiera sekcja “Obsługa typu punkt z punktem w kanałach AMQP” na stronie 704.

V 9.3.0 Produkt IBM MQ 9.3.0 dodaje obsługę przeglądania kolejek dla kanałów produktu IBM MQ AMQP, aby aplikacje JMS produktu Apache Qpid mogły nawiązywać połączenie z produktem IBM MQ i przeglądać komunikaty z kolejki. Więcej informacji zawiera sekcja “Obsługa typu punkt z punktem w kanałach AMQP” na stronie 704.

V 9.3.0 Program IBM MQ 9.3.0 dodaje dwa dodatkowe atrybuty kanałów AMQP: TMPMODEL i TMPQPRFX. Te atrybuty są przeznaczone dla kolejki modelowej i przedrostka kolejki tymczasowej, który ma być używany podczas tworzenia kolejki tymczasowej.

#### Komunikacja wewnętrzna z innymi aplikacjami IBM MQ

Istnieje możliwość wysyłania komunikatów między aplikacjami Apache Qpid JMS i innymi aplikacjami IBM MQ . Na przykład aplikacja Qpid Apache może publikować komunikaty w temacie, a aplikacje MQ Light mogą je odbierać, tworząc subskrypcję.

Aplikacja Apache Qpid JMS może również publikować komunikaty zużywane przez tradycyjne aplikacje IBM MQ , na przykład za pomocą wywołania API MQSUB w celu zasubskrybowania tego samego tematu.

Podobnie aplikacje Apache Qpid JMS mogą subskrybować tematy IBM MQ , w których tradycyjne aplikacje IBM MQ publikują komunikaty.

Istnieje również możliwość, że aplikacja Apache Qpid JMS współużytkuje subskrypcję z aplikacją MQ Light , o ile oba klienty określają tę samą nazwę zasobu współużytkowanego i ten sam wzorzec tematu.

Należy zauważyć, że w tym celu aplikacja Apache Qpid JMS nie może łączyć się z identyfikatorem klienta. Dzięki temu nazwa subskrypcji produktu IBM MQ używana przez obie aplikacje będzie taka sama.

**Ostrzeżenie:** Aplikacja Apache Qpid JMS nie może współużytkować subskrypcji z aplikacją IBM MQ JMS .

#### Ograniczenia produktuApache Qpid JMS

Obsługiwane są następujące możliwości JMS :

- Potwierdzanie, automatyczne potwierdzanie i dups w trybie ok acknowledge (DUPS_OK_ACKNOWLEDGE)
  - Nawiązywanie połączenia z referencjami lub bez nich
  - Tworzenie konsumenta w miejscu docelowym tematu
  - Tworzenie trwałego konsumenta w miejscu docelowym tematu
  - Tworzenie współużytkowanego konsumenta w miejscu docelowym tematu
  - Tworzenie współużytkowanego trwałego konsumenta w miejscu docelowym tematu
  - Tryby potwierdzania i automatycznego potwierdzania klienta
  - Potwierdzenie komunikatu i potwierdzenie sesji

  - Anulowanie subskrypcji trwałej subskrypcji
  - V 9.3.0 Tworzenie kolejki tymczasowej
  - V 9.3.0 Tworzenie konsumenta w miejscu docelowym kolejki lub kolejki tymczasowej
  - V 9.3.0 JMS MessageListeners (Procesy nasłuchiwania komunikatów JMS)
  - V 9.3.0 Konsument JMS do odbierania treści. Metoda JMS 2.0 wywołała metodę `Consumer.receiveBody()`
  - V 9.3.0 Obsługiwane są następujące typy komunikatów JMS:
    - BytesMessage
    - MapMessage
    - ObjectMessage
    - StreamMessage
    - TextMessage
  - V 9.3.0 Przeglądanie komunikatów z kolejki

Następujące możliwości produktu JMS nie są obsługiwane przez klienty AMQP:

- Użycie sesji transakcyjnych i transakcyjnych JMSContexts
  - Użycie selektorów komunikatów
  - Użycie atrybutu **nolocal**
  - Użycie sesji transakcyjnych
  - Wykorzystanie opóźnienia dostawy
  - W katalogu IBM MQ 9.3.0: przeglądanie komunikatów z kolejki.
  - Tworzenie wielu trwałych subskrypcji lub konsumentów o tym samym identyfikatorze klienta i temacie
  - V 9.3.0 Tematy tymczasowe JMS
  - Filtry AMQP nie są obsługiwane.

V 9.3.3 W produkcie IBM MQ 9.3.3 następująca uwaga nie ma już zastosowania do użytkowników produktu Continuous Delivery .

**Ostrzeżenie:** V 9.3.0 Potwierdzenie przez klienta: jeśli mają być używane nieustalone operacje przesyłania komunikatów AMQP, to znaczy, jeśli wymagane jest potwierdzenie komunikatów przez klienta, wówczas klienty muszą rozstrzygać w odpowiednim czasie, wysyłając potwierdzenia komunikatów w odpowiednim czasie podczas korzystania z trybu potwierdzania klienta, lub należy rozważyć ustawienie wyższej wartości właściwości MARKINT (**MsgMarkBrowseInterval**) menedżera kolejek.

Wartością domyślną parametru **MsgMarkBrowseInterval** jest pięć sekund. Jeśli aplikacja nie zostanie ustawiona w tej wartości domyślnej, mogą zostać wyświetlone zduplikowane komunikaty. Aby uniknąć duplikowania komunikatów, należy odpowiednio zwiększyć wartość parametru **MsgMarkBrowseInterval** , najlepiej ustawić go na NOLIMIT, aby reprezentował nieograniczony przedział czasu. Jeśli aplikacja ulegnie awarii lub rozłączy się przed rozstrzygnięciem komunikatu, komunikaty są udostępniane innej aplikacji.

Więcej informacji na ten temat zawiera sekcja **MsgMarkBrowseInterval** . Ponieważ jest to właściwość menedżera kolejek, ustawiona wartość zostanie zastosowana do wszystkich aplikacji połączonych z tym menedżerem kolejek.

W przypadku protokołu AMQP parametr **MsgMarkBrowseInterval** jest poprawny tylko dla kolejek, a nie dla subskrypcji.

### Pobieranie przykładowych klientów AMQP

Produkt IBM MQ nie jest dostarczany z klientami AMQP, ale można pobrać klienty MQ Light lub klienty typu Open Source AMQP oparte na bibliotekach Qpid serwera Apache . Więcej informacji na ten temat zawierają IBM MQ Light i Apache Qpid.

Można również pobrać inne klienty typu Open Source AMQP oparte na bibliotekach Apache Qpid. Więcej informacji na ten temat zawiera https://qpid.apache.org/index.html.

**Ostrzeżenie:** Dział wsparcia IBM nie może zapewnić obsługi konfiguracji lub defektów dla tych pakietów klienta, a wszelkie pytania dotyczące użycia lub raporty dotyczące defektów kodu powinny być kierowane do odpowiednich projektów.

### Wdrażanie klientów AMQP na serwerze IBM MQ

Gdy aplikacja jest gotowa do wdrożenia, wymaga wszystkich możliwości monitorowania, niezawodności i bezpieczeństwa innych aplikacji korporacyjnych. Może również wymieniać dane z innymi aplikacjami korporacyjnymi.

Po wdrożeniu klienta AMQP można wymieniać komunikaty z aplikacjami IBM MQ . Jeśli na przykład do wysłania komunikatu łańcuchowego JavaScript używany jest klient AMQP, aplikacja IBM MQ odbierze komunikat produktu MQ , w którym pole formatu deskryptora MQMD ma wartość MQSTR.

### Zarządzanie kanałem AMQP

Kanał AMQP może być zarządzany w taki sam sposób, jak inne kanały produktu MQ . Do definiowania, uruchamiania i zatrzymywania kanałów oraz zarządzania nimi można używać komend MQSC, komunikatów komend PCF lub komend IBM MQ Explorer . W sekcji Tworzenie i używanie kanałów AMQPudostępniono przykładowe komendy służące do definiowania i uruchamiania połączeń klientów z menedżerem kolejek.

Po uruchomieniu kanału AMQP można go przetestować, łącząc się z klientem AMQP 1.0 . Na przykład MQ Light, Apache Qpid Proton lub Apache Qpid JMS.

**Zadania pokrewne**

Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

## ALW MQ Light, Apache Qpid JMSi AMQP (Advanced Message Queuing Protocol)

Klient MQ Light , klienty Qpid Apache , takie jak Apache Proton i interfejsy API Apache Qpid JMS są oparte na protokole łącznika OASIS Standard AMQP 1.0 . Protokół AMQP określa sposób wysyłania komunikatów między nadawcami i odbiorcami. Aplikacja działa jako nadawca, gdy wysyła komunikat do brokera komunikatów, takiego jak IBM MQ. Produkt IBM MQ działa jako nadawca, gdy wysyła komunikat do aplikacji AMQP.

Niektóre z zalet protokołu AMQP są następujące:

- Otwarty standaryzowany protokół
- Kompatybilność z innymi klientami Open Source AMQP 1.0
- Wiele dostępnych implementacji klienta Open Source

Chociaż każdy klient AMQP 1.0 może nawiązać połączenie z kanałem AMQP, niektóre funkcje AMQP nie są obsługiwane, na przykład transakcje lub wiele sesji.

Więcej informacji na ten temat zawiera serwis WWW AMQP.org i dokument OASIS Standard AMQP 1.0 w formacie PDF.

Interfejsy API MQ Light i Apache Qpid JMS mają następujące funkcje przesyłania komunikatów:

- Dostarczenie jednorazowe

- Co najmniej jednokrotne dostarczenie komunikatu
- Adresowanie docelowe łańcucha tematu
- Trwałość komunikatu i miejsca przeznaczenia
- Współużytkowane miejsca docelowe umożliwiające wielu subskrybentom współużytkowanie obciążenia
- Przejęcie klienta w celu łatwego rozstrzygania zawieszonych klientów
- Konfigurowalny odczyt z wyprzedzeniem komunikatów
- Konfigurowalne potwierdzenie komunikatów

Pełna dokumentacja funkcji API Apache Qpid JMS znajduje się w serwisie WWW Qpid JMS.

**Zadania pokrewne**
Tworzenie i używanie kanałów AMQP
Zabezpieczanie klientów AMQP

## ALW Obsługa protokołu AMQP 1.0

Kanały AMQP udostępniają poziom obsługi aplikacji zgodnych ze specyfikacją AMQP 1.0-compliant .

Kanały AMQP obsługują podzbiór protokołu AMQP 1.0 . Można połączyć klienty zgodne ze specyfikacją AMQP 1.0 z kanałem AMQP produktu IBM MQ . Aby można było używać wszystkich funkcji przesyłania komunikatów obsługiwanych przez kanały AMQP, należy poprawnie ustawić wartości niektórych pól AMQP 1.0 .

Informacje te dotyczą sposobu formatowania pól AMQP i zawierają listę opcji specyfikacji AMQP 1.0 , które nie są obsługiwane przez kanały AMQP.

Następujące funkcje specyfikacji AMQP 1.0 nie są obsługiwane lub ich użycie jest ograniczone:

### Ramka ATTACH

V 9.3.0

Kanały AMQP oczekują, że możliwości w ramce ATTACH zawierają jeden z następujących elementów:

```
topic
temporary queue
queue
shared
```

Możliwości implikują typ obiektu, a w przypadku wielu możliwości kolejność priorytetów wyboru możliwości to temat, kolejka tymczasowa i kolejka.

Jeśli możliwość nie zawiera oczekiwanej wartości, domyślną możliwością jest temat. Wszelkie inne możliwości są ignorowane.

**Uwaga:** Niektóre klienty AMQP nie ustawiają tych możliwości i uzyskają domyślne zachowanie publikowania/subskrypcji produktu IBM MQ . Na przykład produkt Quarkus Reactive Messaging AMQP 1.0 Connector ustawia tylko możliwości począwszy od wersji 2.8.0CR1 .

V 9.3.0 Kanały AMQP oczekują, że plik `distribution-Mode` w ramce ATTACH zawiera jedną z następujących wartości dla źródła lub celu:

- `Przeniesienie`
- `kopia`

gdzie `move` oznacza destrukcyjne pobranie, a `copy` oznacza przeglądarkę.

**Uwaga:** Jeśli parametr `distribution-Mode` nie jest ustawiony lub jest ustawiony na wartość inną niż `copy`, przyjmowany jest parametr `move` .

## Nazwy odsyłaczy

Kanały AMQP oczekują, że nazwa odsyłacza AMQP będzie zgodna z jednym z pięciu formatów:

- Zwykły temat (do publikowania i subskrybowania)
  - Publikowanie komunikatów: zwykły łańcuch tematu (na przykład nazwa odsyłacza "`/sports/football`") powoduje opublikowanie komunikatu w temacie `/sports/football` .
  - Subskrybowanie tematu w celu odbierania komunikatów: zwykły łańcuch tematu (na przykład nazwa odsyłacza "`/sports/football`" ) powoduje zdefiniowanie subskrypcji w temacie `/sports/football` .
- Prywatny temat szczegółowy (na potrzeby subskrybowania)
  - Szczegółowy łańcuch tematu opisujący subskrypcję prywatną w postaci: "`private:`*`topic string`*" (na przykład: "`private:/sports/football`"). Zachowanie jest takie samo jak zwykły łańcuch tematu. Deklaracja `private` odróżnia subskrypcję specyficzną dla konkretnego klienta AMQP od subskrypcji współużytkowanej przez klienty.
- Współużytkowany temat szczegółowy (na potrzeby subskrybowania)
  - Szczegółowy łańcuch tematu opisujący współużytkowaną subskrypcję w postaci: "`share:`*`share name`*`:`*`topic string`*" (na przykład: "`share:bbc:/sports/football`").
- V 9.3.0 Kolejka (dla przesyłania komunikatów między punktami dla producenta i konsumenta)
  - Producent wysyła komunikaty; łańcuch nazwy kolejki powoduje, że producent wysyła komunikat do kolejki.
  - Konsument odbiera komunikaty; łańcuch nazwy kolejki powoduje, że konsument odbiera komunikaty z kolejki.
- V 9.3.0 Puste (dla przesyłania komunikatów typu punkt z punktem w kolejce tymczasowej)
  - Producent wysyła komunikaty do kolejki tymczasowej. Puste pole powoduje, że producent wysyła komunikat do kolejki tymczasowej.
  - Konsument odbiera komunikaty w kolejce tymczasowej. Puste pole powoduje, że konsument odbiera komunikaty z kolejki tymczasowej.

Więcej informacji na temat sposobu odwzorowywania komunikatów AMQP na komunikaty produktu IBM MQ i z nich zawiera sekcja “Odwzorowanie pól AMQP na pola produktu IBM MQ (komunikaty przychodzące)” na stronie 708.

## Maksymalna długość łańcuchów tematów, nazw współużytkowanych i identyfikatorów klientów

Łańcuch tematu, nazwa zasobu współużytkowanego i identyfikator klienta muszą znajdować się w zakresie 10237 bajtów. Ponadto maksymalna długość identyfikatora klienta wynosi 256 znaków.

Te maksymalne długości oznaczają, że można mieć jedną z następujących możliwości:

- bardzo długi łańcuch tematu, pod warunkiem, że nazwa zasobu współużytkowanego jest krótka
- długa nazwa zasobu współużytkowanego, ale krótki łańcuch tematu

## Identyfikatory kontenerów

Kanały AMQP oczekują, że identyfikator kontenera dla komendy AMQP Open będzie zawierał unikalny identyfikator klienta AMQP. Maksymalna długość identyfikatora klienta AMQP wynosi 256 znaków, a identyfikator może zawierać znaki alfanumeryczne, znak procentu (%), ukośnik (/), kropkę (.) i podkreślenie (_).

## Sesje

Kanały AMQP obsługują tylko jedną sesję AMQP. Klient AMQP, który próbuje utworzyć więcej niż jedną sesję AMQP, odbiera komunikat o błędzie i jest odłączony od kanału.

### Transakcje

Kanały AMQP nie obsługują transakcji AMQP. Ramka przyłączenia AMQP, która próbuje skoordynować nową transakcję lub ramka przesyłania AMQP, która próbuje zadeklarować nową transakcję, została odrzucona z komunikatem o błędzie.

### Stan dostarczenia

Kanały AMQP obsługują tylko stan dostarczania dla ramek dyspozycji o wartości Zaakceptowane, Zwolnione lub Zmodyfikowane. Należy zauważyć, że jeśli używany jest stan Zmodyfikowany, kanały AMQP nie obsługują opcji "undeliverable-here".

**Zadania pokrewne**

Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

## V 9.3.0 ALW Obsługa typu punkt z punktem w kanałach AMQP

Kanał AMQP produktu IBM MQ udostępnia obsługę wysyłania komunikatów do kolejek i odbierania komunikatów z kolejek.

Klienty AMQP, takie jak biblioteka Apache Qpid™ JMS , żądają możliwości `queue` lub `temporary-queue` podczas wysyłania ramki przyłączenia AMQP. Możliwości umożliwiają kanałowi AMQP zidentyfikowanie obiektu jako kolejki, kolejki tymczasowej lub tematu. W przypadku braku możliwości kolejki lub kolejki tymczasowej, a nawet dowolnej możliwości, przyjmuje się, że żądanie dotyczy tematu.

Kanały AMQP produktu IBM MQ udostępniają obsługę typu kolejki dla następujących elementów:

**Odbieranie i wysyłanie kolejki**
Komunikaty mogą być wysyłane do kolejki i pobierane z kolejki. W przypadku odbierania komunikatów obsługiwane są tryby synchroniczny i asynchroniczny.

**V 9.3.0 Komunikat przeglądania kolejki**
Oprócz umieszczania komunikatów w kolejce i pobierania komunikatów z kolejki, komunikaty mogą być również przeglądane z kolejki.

**Obsługa kolejki tymczasowej**
Komunikaty mogą być wysyłane do kolejki tymczasowej i pobierane z kolejki tymczasowej. Należy zauważyć, że usuwanie kolejki tymczasowej jest obsługiwane, jeśli ten sam obiekt kolejki tymczasowej użyty do utworzenia kolejki tymczasowej jest również używany do usunięcia kolejki tymczasowej.

SYSTEM SYSTEM.DEFAULT.MODEL.QUEUE jest używany podczas tworzenia kolejki tymczasowej, a przedrostek kolejki tymczasowej to AMQP.*.

SYSTEM SYSTEM.DEFAULT.MODEL.QUEUE jest domyślnie tymczasową kolejką dynamiczną, ale można użyć właściwości **`Definition type`** w systemie SYSTEM.DEFAULT.MODEL.QUEUE , aby zmienić kolejkę na trwałą kolejkę dynamiczną.

**stała kolejka dynamiczna**
Trwała kolejka dynamiczna jest usuwana, gdy klient AMQP, taki jak biblioteka Apache Qpid JMS , wysyła żądanie z ramką `detach` z atrybutem **`closed`** ustawionym na wartość *true*.

**Ważne:**

**Zachowanie Qpid JMS :**
Należy wywołać komendę API Qpid JMS , na przykład metodę `javax.jms.TemporaryQueue.delete()` , aby zniszczyć kolejkę po użyciu, a proces ten również czyści komunikaty znajdujące się w kolejce.

Jeśli taka komenda nie zostanie wydana, po zamknięciu połączenia kolejka pozostanie bez komunikatów.

-

**tymczasowa kolejka dynamiczna**
Tymczasowa kolejka dynamiczna jest usuwana, gdy klient AMQP zamyka połączenie.

**Ważne:**

**Zachowanie Qpid JMS :**
W przypadku wywołania komendy API Qpid JMS , na przykład metody `javax.jms.TemporaryQueue.delete()` , zamknięcia połączenia JMS lub zerwania połączenia, kolejka jest usuwana i wszystkie komunikaty są tracone.

Zamknięcie samej sesji JMS nie powoduje usunięcia kolejki tymczasowej, nawet jeśli kolejka tymczasowa mogła zostać utworzona przy użyciu metody `javax.jms.Session.createTemporaryQueue()` .

-

**Zadania pokrewne**
Tworzenie i używanie kanałów AMQP
Zabezpieczanie klientów AMQP

## ALW Odwzorowywanie pól komunikatów AMQP i IBM MQ

Komunikaty AMQP składają się z nagłówka, adnotacji dostarczania, adnotacji komunikatu, właściwości, właściwości aplikacji, treści i stopki.

Komunikaty AMQP składają się z następujących części:

**Nagłówek**
Opcjonalny nagłówek zawiera pięć stałych atrybutów komunikatu:

- **trwała** -określa wymagania dotyczące trwałości
- **priority** -względny priorytet komunikatu
- **ttl** -czas życia w milisekundach
- **first-acquirer** -jeśli jest to prawda, komunikat nie został uzyskany przez żaden inny odsyłacz
- **delivery-count** -liczba poprzednich, niepomyślnych prób dostarczenia.

**Dostarczanie-adnotacje**
Opcjonalne. Określa niestandardowe atrybuty nagłówka komunikatu dla różnych docelowych odbiorców. Adnotacje dostarczania przekazują informacje z węzła wysyłającego do węzła odbierającego.

**Adnotacje komunikatu**
Opcjonalne. Określa niestandardowe atrybuty nagłówka komunikatu dla różnych docelowych odbiorców. Sekcja message-Adnotacje jest używana dla właściwości komunikatu, które są przeznaczone dla infrastruktury i powinny być propagowane w każdym kroku dostarczania.

**Właściwości**
Opcjonalne. Ta część jest odpowiednikiem deskryptora komunikatu produktu MQ . Zawiera on następujące stałe pola:

- **message-id** -identyfikator komunikatu aplikacji.
- **user-id** -identyfikator użytkownika tworzącego
- **to** -adres węzła, do którego przeznaczony jest komunikat.
- **subject** -temat wiadomości.
- **reply-to** -węzeł, do którego wysyłane są odpowiedzi.
- **correlation-id** -identyfikator korelacji aplikacji
- **content-type** -typ treści MIME
- **content-encoding** -typ treści MIME. Używany jako modyfikator typu treści.
- **absolute-expired-time** -czas, w którym komunikat jest uznawany za wygasły.
- **creation-time** -godzina utworzenia tego komunikatu.

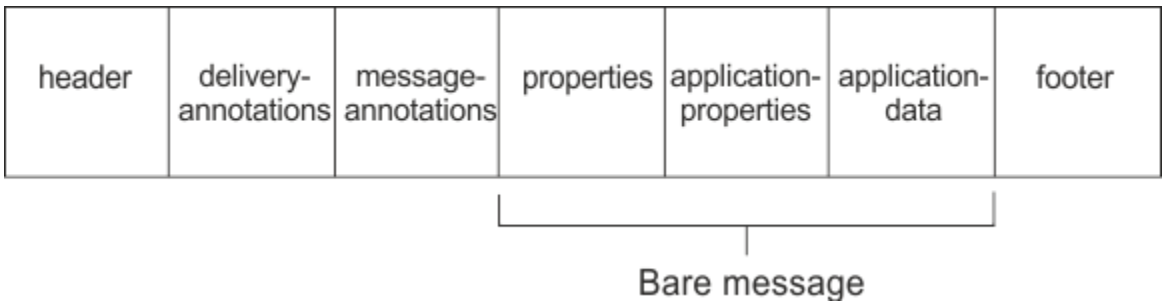
- **group-id** -grupa, do której należy ten komunikat.
- **group-sequence** -numer kolejny tego komunikatu w obrębie jego grupy.
- **reply-to-group-id** -grupa, do której należy komunikat odpowiedzi.

**Aplikacje-właściwości**
Odpowiednik właściwości komunikatu produktu MQ .

**Treść**
Odpowiednik ładunku użytkownika produktu MQ .

**Stopka**
Opcjonalne. Stopka jest używana w celu uzyskania szczegółowych informacji o wiadomości lub dostarczeniu, które mogą być obliczane lub wartościowane tylko po skonstruowaniu lub wyświetleniu całej wiadomości (na przykład skrótu wiadomości, HMAC, sygnatur i szczegółów szyfrowania).

Format komunikatu AMQP został przedstawiony na poniższym rysunku:




Części właściwości, application-properties i application-data są nazywane "samym komunikatem". Jest to komunikat wysłany przez nadawcę i niezmienialny. Odbiorca widzi cały komunikat, w tym nagłówek, stopkę, adnotacje dostarczania i adnotacje komunikatu.

Pełny opis formatu komunikatu AMQP 1.0 zawiera standard OASIS dostępny pod adresem https://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf.

**Zadania pokrewne**
Tworzenie i używanie kanałów AMQP
Zabezpieczanie klientów AMQP

## ALW Odwzorowanie pól IBM MQ na pola AMQP (komunikaty wychodzące)

Po opublikowaniu komunikatu produktu IBM MQ i wysłaniu go przez produkt IBM MQ do konsumenta AMQP, będzie on propagować niektóre atrybuty komunikatu produktu IBM MQ do równoważnych atrybutów komunikatu produktu AMQP.

### header (nagłówek)

Nagłówek jest uwzględniany tylko wtedy, gdy jedno z pięciu pól w nagłówku zawiera wartość inną niż domyślna. Tylko pola z wartością inną niż domyślna są uwzględniane w nagłówku. Pięć pól nagłówka jest początkowo uzyskiwanych z równoważnej właściwości `mq_amqp.Hdr` (jeśli jest ustawiona), a następnie modyfikowanych w sposób przedstawiony w poniższej tabeli:

*Tabela 101. Odwzorowania pól nagłówka*

| Pole | Wartość domyślna | Wartość |
|---|---|---|
| Trwałe | Fałsz | Wartość true, jeśli parametr `MQMD.Persistence` ma wartość `MQPER_PERSISTENT`, w przeciwnym razie wartość false. |

| *Tabela 101. Odwzorowania pól nagłówka (kontynuacja)* | | |
|---|---|---|
| **Pole** | **Wartość domyślna** | **Wartość** |
| priorytet | 4 | Z pliku `mq_amqp.Hdr.Pri`, jeśli jest ustawiony, lub w inny sposób z pliku `MQMD.Priority`, jeśli jest ustawiony. Jeśli żadna z tych wartości nie jest ustawiona, należy ustawić wartość 4. |
| ttl | nie dotyczy | `MQMD.Expiry` w milisekundach. Jeśli `MQMD.Expiry` ma wartość MQEI_UNLIMITED, należy ustawić wartość maksymalną pola ttl AMQP |
| pierwszy nabywca | Fałsz | Z wartości `mq_amqp.Hdr.Fac`(jeśli jest ustawiona) lub z wartości false (w przeciwnym razie). |
| liczba dostarczeń | 0 | Od `mq_amqp.Hdr.Dct`, jeśli jest ustawiona, lub 0 w przeciwnym razie. |

## adnotacja dostarczania

Ustawiane zgodnie z potrzebami przez kanał AMQP.

## adnotacja komunikatu

Nie uwzględniono.

## Właściwości

**Właściwości** nie będą modyfikowane w stosunku do równoważnych właściwości `mq_amqp.Prp` , jeśli zostaną ustawione. Jeśli komunikat nie był pierwotnie komunikatem AMQP (czyli typem PutApplnie jest MQAT_AMQP), to sekcja właściwości jest generowana w sposób opisany w poniższej tabeli:

| *Tabela 102. Odwzorowania pól właściwości* | |
|---|---|
| **Nazwa** | **Wartość** |
| id_komunikatu | Parametr `MQMD.MsgId` jest ustawiony jako binarny. |
| id-użytkownika | Format UTF-8 `MQMD.UserIdentifier` jest ustawiony jako binarny w kolejności bajtów obowiązującej w sieci. |
| do | Kolejka, z której komunikat został odebrany, lub, w przypadku publikacji, łańcuch tematu. |
| obiekt | Nie ustawiono. |
| odpowiedz-do | Wartość `MQMD.ReplyToQ` , jeśli nie jest pusta, w przeciwnym razie nie jest ustawiona. |
| id-korelacji | Parametr `MQMD.CorrelId` jest ustawiany jako binarny, jeśli nie jest pusty, w przeciwnym razie nie jest ustawiany. |
| Content-Type | Nie ustawiono. |
| kodowanie-treści | Nie ustawiono. |
| bezwzględny-czas-ważności | Nie ustawiono. |
| czas utworzenia | Pola `MQMD.PutDate` i `MQMD.PutTime` są używane do generowania znacznika czasu. |

| *Tabela 102. Odwzorowania pól właściwości (kontynuacja)* | |
|---|---|
| **Nazwa** | **Wartość** |
| ID_grupy | Nie ustawiono. |
| sekwencje grup | Nie ustawiono. |
| reply-to-group-id (identyfikator grupy odpowiedzi) | Nie ustawiono. |

### właściwości-aplikacji

Wszystkie właściwości IBM MQ w grupie "usr" są dodawane jako **application-properties**.

### treść

Kanał AMQP wykonuje operację get with convert, aby przekształcić ładunek IBM MQ na UTF-8.

Jeśli ładunek IBM MQ nie zawiera komunikatu AMQP, ładunek IBM MQ jest ustawiany w treści jako pojedyncza sekcja danych łańcuchowych dla formatu MQFMT_STRING (pod warunkiem, że konwersja do formatu UTF-8 powiodła się) lub jako pojedyncza sekcja danych binarnych.

Jeśli dołączono komunikat formatu AMQP, jest on ustawiany jako treść. Wszystkie nagłówki IBM MQ (z wyjątkiem właściwości komunikatów zwracanych w uchwycie komunikatu), które poprzedzają komunikat AMQP, są dołączane jako wartość binarna, jeśli treść jest sekwencją AMQP. W przeciwnym razie nagłówki IBM MQ są usuwane.

### stopka

Stopka nie jest uwzględniana.

**Zadania pokrewne**

Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

**Odsyłacze pokrewne**

MQMD-deskryptor komunikatu

## ALW Odwzorowanie pól AMQP na pola produktu IBM MQ (komunikaty przychodzące)

Gdy kanał AMQP odbiera komunikat i umieszcza go w katalogu IBM MQ, propaguje niektóre atrybuty komunikatu AMQP do równoważnych atrybutów komunikatu IBM MQ .

Podczas odwzorowywania przychodzącego komunikatu AMQP obowiązują następujące ograniczenia:

- Jeśli pole `message-id` lub `correlation-id` w części właściwości ma identyfikator UUID lub identyfikator ulong, komunikat zostanie odrzucony.
- Dowolny `message-annotations` powoduje, że komunikat jest odrzucany.
- Sekcje `delivery-annotations` i `footer` są dozwolone, ale nie są propagowane do komunikatu IBM MQ .

W poniższych podsekcjach przedstawiono wyrażenie IBM MQ komunikatu AMQP.

## deskryptor komunikatu

*Tabela 103. Deskryptor komunikatu AMQP*

| **Pole** | **Wartość** |
|---|---|
| StrucId | MQMD_STRUC_ID (Identyfikator struktury kolejki) |
| Wersja | MQMD_VERSION_1 |
| Raport | MQRO_BRAK |
| MsgType | MQMT_DATAGRAM, |
| Utrata ważności | Wartość pobrana z pola `ttl` w nagłówku komunikatu AMQP |
| Opinie | MQFB_BRAK |
| Kodowanie | MQENC_NORMAL |
| CodedCharSetId | 1208 (UTF-8) |
| Format | Patrz ładunek |
| Priorytet | Wartość pobrana z pola `priority` w nagłówku komunikatu AMQP. Jeśli jest ustawiona, ograniczona do maksymalnie 9. Jeśli nie jest ustawiona, przyjmuje wartość domyślną 4. |
| Trwałość | Jeśli pole `durable` w nagłówku komunikatu AMQP jest ustawione na wartość true, należy ustawić wartość MQPER_PERSISTENT. W przeciwnym razie należy ustawić wartość MQPER_NOT_PERSISTENT. |
| MagId | Menedżer kolejek przydziela unikalną 24-bajtową wartość pola MsgId. |
| Korellda | Wartość pobrana z pola `correlation-id` we właściwościach AMQP, jeśli jest ustawiona. Ustawiona na 24-bajtową wartość binarną. W przeciwnym razie ustaw wartość MQCI_NONE/. |
| BackoutCount | 0 |
| ReplyToQ | V 9.3.0 Wartość pobrana z pola `reply-to` we właściwościach AMQP, jeśli jest ustawiona. W przeciwnym razie należy ustawić wartość "". |
| ReplyToQMgr | "" |
| V 9.3.0 Raport | Wartość pochodząca z dowolnej właściwości raportu JMS IBM ustawionej we właściwościach aplikacji AMQP. |
| UserIdentifier | Ustaw identyfikator uwierzytelnionego użytkownika, który nawiązał połączenie z kanałem AMQP |
| AccountingToken | MQACT_NONE |
| Dane_tożsamości_aplikacji | Łańcuch szesnastkowy. Ustawiona na ostatnie 8 bajtów identyfikatora połączenia produktu MQ kanału AMQP. |
| Typ_aplikacji_wstawiającej | MQAT_AMQP |
| Nazwa_aplikacji_wstawiającej | |
| PutDate | Wartość pobrana z pola `creation-time` we właściwościach AMQP, jeśli jest ustawiona. W przeciwnym razie należy ustawić bieżącą datę. |
| PutTime | Wartość pobrana z pola `creation-time` we właściwościach AMQP, jeśli jest ustawiona. W przeciwnym razie należy ustawić bieżący czas. |

| *Tabela 103. Deskryptor komunikatu AMQP (kontynuacja)* | |
|---|---|
| **Pole** | **Wartość** |
| Dane_pochodzenia_aplikacji | "" |

## Właściwości komunikatu

Istnieją dwie przyczyny ustawiania właściwości komunikatu:

- Umożliwia przepływ części komunikatu AMQP przez menedżer kolejek bez wpływu na ładunek komunikatu.
- Umożliwia wybór `application-properties`.

W poniższej tabeli przedstawiono właściwości ustawione w komunikacie AMQP:

| *Tabela 104. Właściwości komunikatu AMQP* | | | |
|---|---|---|---|
| **Nazwa właściwości** | **Nazwa MQRFH2** | **Typ** | **Opis** |
| Program AMQPListener | mq_amqp.Lis | MQTYPE_STRING (łańcuch MQTYPE) | Łańcuch identyfikujący kanał AMQP. Jest on używany do generowania komunikatu, aby zainteresowane strony mogły określić, która wersja umieściła komunikat (na przykład zespół serwisowy podczas diagnozowania problemów). Poprawność wartości nie jest sprawdzana przez menedżer kolejek i nie może być udokumentowana zewnętrznie. |
| Wersja protokołu AMQP | mq_amqp.Ver | MQTYPE_STRING (łańcuch MQTYPE) | Wersja komunikatu AMQP. Jeśli nie istnieje, przyjmowana jest wartość "1.0". Poprawność wartości nie jest sprawdzana przez menedżer kolejek. |
| Klient AMQPClient | mq_amqp.Cli | MQTYPE_STRING (łańcuch MQTYPE) | Łańcuch identyfikujący interfejs API. Jest on używany do wysyłania komunikatu AMQP do kanału, aby zainteresowane strony mogły określić, która wersja umieściła komunikat (na przykład zespół serwisowy podczas diagnozowania problemów). Poprawność wartości nie jest sprawdzana przez menedżer kolejek i nie może być udokumentowana zewnętrznie. |
| AMQPDurable | mq_amqp.Hdr.Dur | MQTYPE_BOOLEAN (wartość boolowska) | Wartość pola `durable` w nagłówku komunikatu AMQP, jeśli jest ustawiona. |
| Priorytet AMQPPriority | mq_amqp.Hdr.Pri | MQTYPE_INT32 | Wartość pola `priority` w nagłówku komunikatu AMQP, jeśli jest ustawiona. |

*Tabela 104. Właściwości komunikatu AMQP (kontynuacja)*

| Nazwa właściwości | Nazwa MQRFH2 | Typ | Opis |
|---|---|---|---|
| Komenda AMQPTtl | mq_amqp.Hdr.Ttl | MQTYPE_INT64 | Wartość pola `ttl` w nagłówku komunikatu AMQP, jeśli jest ustawiona. |
| AMQPFirstAcquirer | mq_amqp.Hdr.Fac | MQTYPE_BOOLEAN (wartość boolowska) | Wartość pola `first-acquirer` w nagłówku komunikatu AMQP, jeśli jest ustawiona. |
| AMQPDeliveryCount | mq_amqp.Hdr.Dct | MQTYPE_INT64 | Wartość pola `delivery-count` w nagłówku komunikatu AMQP, jeśli jest ustawiona. |
| AMQPMsgId | mq_amqp.Prp.Mid | MQTYPE_STRING (łańcuch MQTYPE) | Wartość pola `message-id` we właściwościach AMQP, jeśli jest ustawiona jako łańcuch. |
| | | MQTYPE_BYTE_STRING ŁAŃCUCH | Wartość pola `message-id` we właściwościach AMQP, jeśli jest ustawiona jako łańcuch bajtów. |
| AMQPUserId | mq_amqp.Prp.Uid | MQTYPE_BYTE_STRING ŁAŃCUCH | Wartość pola `user-id` we właściwościach AMQP, jeśli jest ustawiona. |
| Komenda AMQPTo | mq_amqp.Prp.To | MQTYPE_STRING (łańcuch MQTYPE) | Wartość pola `to` we właściwościach AMQP, jeśli jest ustawiona. |
| Podmiot AMQP | mq_amqp.Prp.Sub | MQTYPE_STRING (łańcuch MQTYPE) | Wartość pola `subject` we właściwościach AMQP, jeśli jest ustawiona. |
| AMQPReplyTo | mq_amqp.Prp.Rto | MQTYPE_STRING (łańcuch MQTYPE) | Wartość pola `reply-to` we właściwościach AMQP, jeśli jest ustawiona. |
| AMQPCorrelationId | mq_amqp.Prp.Cid | MQTYPE_STRING (łańcuch MQTYPE) | Wartość pola `correlation-id` we właściwościach AMQP, jeśli jest ustawiona jako łańcuch. |
| | | MQTYPE_BYTE_STRING ŁAŃCUCH | Wartość pola `correlation-id` we właściwościach AMQP, jeśli jest ustawiona jako łańcuch bajtów. |
| AMQPContentType | mq_amqp.Prp.Cnt | MQTYPE_STRING (łańcuch MQTYPE) | Wartość pola `content-type` we właściwościach AMQP, jeśli jest ustawiona. |
| AMQPContentEncoding | mq_amqp.Prp.Cneing | MQTYPE_STRING (łańcuch MQTYPE) | Wartość pola `content-encoding` we właściwościach AMQP, jeśli jest ustawiona. |
| Czas AMQPAbsoluteExpiry | mq_amqp.Prp.Aet | MQTYPE_STRING (łańcuch MQTYPE) | Wartość pola `absolute-expiry-time` we właściwościach AMQP, jeśli jest ustawiona. |
| AMQPCreationTime | mq_amqp.Prp.Crt | MQTYPE_STRING (łańcuch MQTYPE) | Wartość pola `creation-time` we właściwościach AMQP, jeśli jest ustawiona. |

| *Tabela 104. Właściwości komunikatu AMQP (kontynuacja)* | | | |
|---|---|---|---|
| **Nazwa właściwości** | **Nazwa MQRFH2** | **Typ** | **Opis** |
| AMQPGroupId | mq_amqp.Prp.Gid | MQTYPE_STRING (łańcuch MQTYPE) | Wartość pola `group-id` we właściwościach AMQP, jeśli jest ustawiona. |
| AMQPGroupSequence | mq_amqp.Prp.Gsq | MQTYPE_INT64 | Wartość pola `group-sequence` we właściwościach AMQP, jeśli jest ustawiona. |
| AMQPReplyToGroupId | mq_amqp.Prp.Rtg | MQTYPE_STRING (łańcuch MQTYPE) | Wartość pola `reply-to-group-id` we właściwościach AMQP, jeśli jest ustawiona. |

Każda właściwość aplikacji z komunikatu AMQP jest ustawiona jako właściwość komunikatu IBM MQ . Sekcja `application-properties` musi być tworzona w taki sam sposób, jak bajt po bajcie, dlatego obowiązują następujące ograniczenia:

- Jeśli właściwość aplikacji jest odrzucana przez kod sprawdzania poprawności MQSETMP, komunikat, który ma zostać odrzucony. Na przykład:
  - Długość nazwy właściwości jest ograniczona do wartości MQ_MAX_PROPERTY_NAME_LENGTH.
  - Nazwa właściwości musi być zgodna z regułami zdefiniowanymi w specyfikacji języka Java dla identyfikatorów języka Java .
  - Nazwa właściwości nie może zaczynać się od łańcucha `JMS` ani `usr.JMS` , z wyjątkiem udokumentowanych właściwości JMS, które można ustawić.
  - Nazwa właściwości nie może być słowem kluczowym SQL.
- Właściwość aplikacji zawierająca znak Unicode U+002E (".") powoduje odrzucenie komunikatu. Właściwość musi być możliwa do wyrażenia w grupie usr właściwości używanych przez usługę JMS.
- Obsługiwane są tylko właściwości: null, boolean, byte, short, int, long, float, double, binary i string. Właściwość aplikacji dowolnego innego typu spowoduje odrzucenie komunikatu.

V 9.3.0 Przy użyciu komendy `application-properties`można ustawić następujące właściwości JMS:

- JMS_IBM_REPORT_EXCEPTION (wyjątek 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

Należy zauważyć, że nazwy i wartości właściwości są spójne z odpowiednimi szczegółami produktu “Odwzorowywanie pól specyficznych dla dostawcy JMS” na stronie 166 , a wartości, które nie są poprawne, są ignorowane.

## ładunek

- W przypadku protokołu AMQP body z pojedynczą sekcją danych binarnych dane binarne (z wyjątkiem bitów protokołu AMQP) są umieszczane jako ładunek IBM MQ w formacie MQFMT_NONE.

- W przypadku protokołu AMQP body z pojedynczą sekcją danych łańcuchowych dane łańcuchowe (z wyjątkiem bitów protokołu AMQP) są umieszczane jako ładunek IBM MQ w formacie MQFMT_STRING.
- W przeciwnym razie komenda body protokołu AMQP utworzy ładunek w takim formacie, w jakim jest on dostępny (bez modyfikacji), z formatem MQFMT_AMQP.

**Zadania pokrewne**

Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

## ALW Niezawodność dostarczania komunikatów

W tej sekcji porównano funkcje niezawodności dla interfejsów API MQ Light i Apache Qpid JMS.

**Zadania pokrewne**

Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

## ALW Niezawodność komunikatów MQ Light

Istnieją cztery funkcje interfejsu API MQ Light , które umożliwiają sterowanie niezawodnością dostarczania komunikatów do i z aplikacji AMQP.

Są to:

- "Jakość usługi komunikatu (QOS)" na stronie 713
- "Automatyczne potwierdzenie subskrybenta" na stronie 714
- "Czas życia subskrypcji" na stronie 714
- "Trwałość komunikatu" na stronie 714

### Jakość usługi komunikatu (QOS)

MQ Light API oferuje dwie jakości usług:

- Co najwyżej raz
- Co najmniej raz

Można wybrać, która jakość usługi ma być używana przez publikatorów i subskrybentów.

Jeśli używany jest klient MQ Light , należy ustawić opcję klienta lub subskrypcji **qos** na wartość *QOS_AT_MOST_ONCE* lub *QOS_AT_LEAST_ONCE*.

Jeśli używany jest inny klient AMQP, należy ustawić atrybut **settled** ramki przesyłania (dla publikatorów) lub ramki dyspozycji (dla subskrybentów) na wartość *true* lub *false*, w zależności od jakości usługi, która ma zostać osiągnięta.

Jakość usługi określa, kiedy komunikat jest usuwany ze strony `sending` konwersacji.

**Publikowanie**

Jeśli publikator wybierze co najwyżej **QOS** *0* (jeden raz), nie czeka na potwierdzenie z menedżera kolejek, zanim odrzuci kopię komunikatu.

Jeśli połączenie z menedżerem kolejek nie powiedzie się przed zakończeniem wysyłania, komunikat może nie zostać odebrany przez subskrybentów.

Jeśli publikator wybierze wartość **QOS** *1* (co najmniej raz), przed usunięciem kopii komunikatu oczekuje na potwierdzenie przez menedżer kolejek, że komunikat został zapisany w kolejkach subskrybentów.

Jeśli połączenie z menedżerem kolejek nie powiedzie się podczas wysyłania, publikator ponownie wysyła komunikat po ponownym nawiązaniu połączenia z menedżerem kolejek.

**subskrypcja**

Jeśli subskrybent wybierze wartość **QOS** *0* , menedżer kolejek nie czeka na potwierdzenie od subskrybenta przed usunięciem jego kopii komunikatu.

Jeśli połączenie z subskrybentem nie powiedzie się przed odebraniem komunikatu, komunikat ten może zostać utracony.

Jeśli subskrybent wybierze opcję **QOS** *1* , menedżer kolejek czeka na potwierdzenie od subskrybenta przed usunięciem kopii komunikatu. V 9.3.3 Od IBM MQ 9.3.3potwierdzone komunikaty są usuwane w zadaniach wsadowych w celu zwiększenia wydajności. Więcej informacji na ten temat zawiera sekcja “Usuwanie potwierdzonych komunikatów AMQP z kolejki w partiach” na stronie 716.

Jeśli połączenie z subskrybentem nie powiedzie się przed odebraniem komunikatu, komunikat jest zachowywany przez menedżer kolejek. Menedżer kolejek ponownie wysyła komunikat do subskrybenta po ponownym nawiązaniu połączenia przez menedżer kolejek lub do innego subskrybenta, jeśli subskrypcja jest współużytkowana.

## Automatyczne potwierdzenie subskrybenta

Jeśli subskrybent wybierze opcję **QOS** *1* (co najmniej raz), musi potwierdzić odbiór każdego komunikatu, zanim menedżer kolejek odrzuci jego kopię. Subskrybent może zdecydować, kiedy ma potwierdzać komunikaty.

Jeśli parametr **auto-confirm** ma wartość *true*, klient MQ Light automatycznie potwierdza dostarczenie każdego komunikatu po pomyślnym odebraniu komunikatu przez sieć.

Dzięki temu w przypadku awarii sieci komunikat zostanie ponownie dostarczony do aplikacji. Jednak w dalszym ciągu możliwe jest, że aplikacja utraci komunikat, jeśli wystąpi awaria aplikacji między potwierdzaniem komunikatu przez klienta MQ Light a przetwarzaniem komunikatu przez aplikację.

Jeśli parametr **auto-confirm** ma wartość *false*, klient MQ Light nie potwierdza automatycznie dostarczenia komunikatu, ale pozostawia go w aplikacji, aby zdecydować, kiedy powinien zostać potwierdzony.

Umożliwia to aplikacji wykonanie aktualizacji zasobu zewnętrznego, takiego jak baza danych lub plik, przed potwierdzeniem w menedżerze kolejek, że komunikat został przetworzony i można go usunąć.

## Czas życia subskrypcji

Gdy aplikacja subskrybuje, decyduje, czy subskrypcja i miejsce docelowe, w którym są przechowywane komunikaty dla tej subskrypcji, nadal istnieją po rozłączeniu aplikacji.

MQ Light Opcja subskrypcji **ttl** służy do określania czasu (w milisekundach), przez który subskrypcja nadal będzie istnieć po rozłączeniu się aplikacji. Jeśli aplikacja ponownie nawiąże połączenie przed tym czasem, subskrypcja zostanie wznowiona, a aplikacja będzie mogła nadal odbierać komunikaty z tej subskrypcji.

Jeśli czas życia upłynie bez ponownego nawiązywania połączenia przez aplikację, subskrypcja zostanie usunięta, a wszystkie komunikaty zapisane w miejscu docelowym zostaną utracone, nawet jeśli są komunikatami trwałymi.

Jeśli ważne jest, aby nie tracić komunikatów, należy określić wartość czasu życia dla aplikacji, która jest wystarczająco wysoka, aby zapewnić, że komunikaty nie zostaną utracone podczas wyłączenia.

## Trwałość komunikatu

Trwałość komunikatów jest kontrolowana przez aplikacje publikujące i subskrybujące oraz przez konfigurację obiektów tematu produktu IBM MQ .

Jeśli subskrybent AMQP używa wartości **QOS** *0* (co najwyżej raz) i tworzy nietrwałą subskrypcję, kanał AMQP zawsze umieszcza nietrwałe komunikaty w kolejce subskrybenta, niezależnie od innych opcji opisanych w poniższym tekście.

Należy zauważyć, że jeśli menedżer kolejek jest zatrzymany, zarówno subskrypcja, jak i komunikaty zostaną utracone.

Jeśli publikator AMQP ustawia nagłówek **durable** AMQP na wartość *true*, kanał AMQP umieszcza trwałe komunikaty w kolejkach subskrybentów.

Jeśli menedżer kolejek zostanie zatrzymany z dowolnej przyczyny, po zrestartowaniu menedżera kolejek komunikaty będą nadal dostępne dla subskrybentów.

Jeśli nagłówek **durable** nie jest ustawiony, kanał AMQP wybiera trwałość publikowanych komunikatów na podstawie atrybutu **DEFPSIST** odpowiedniego obiektu tematu IBM MQ .

Domyślnie jest to SYSTEM SYSTEM.BASE.TOPIC, który używa atrybutu **DEFPSIST** o wartości *NO* (nietrwały).

**Ostrzeżenie:** Nowsze wersje klienta MQ Light nie obsługują ustawiania trwałego nagłówka AMQP.

**Zadania pokrewne**

Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

## ALW niezawodność komunikatów Apache Qpid JMS

Istnieją cztery funkcje biblioteki Apache Qpid™ JMS , które umożliwiają sterowanie niezawodnością dostarczania komunikatów do i z aplikacji AMQP.

Są to:

- “Publikowanie” na stronie 715 V 9.3.0 /Producent przesyłania komunikatów w trybie punkt z punktem
  - Utrata ważności komunikatu
  - Trwałość komunikatu
- “subskrypcja” na stronie 716
  - Trwałość subskrypcji
  - Tryb potwierdzenia sesji V 9.3.0 (dotyczy również przesyłania komunikatów od punktu do punktu konsumenta)

## Publikowanie

### Utrata ważności komunikatu

Ustawienie wartości czasu życia producenta JMS ma wpływ na czas utraty ważności komunikatów publikowanych przez tego producenta komunikatów.

Upewnij się, że wartość czasu życia dla producenta JMS jest wystarczająco duża, aby komunikaty zostały zużyte przed utratą ważności.

Pozostawienie nieustawionej wartości czasu życia zapobiega utracie ważności komunikatu z kolejki subskrypcji.

### Trwałość komunikatu

Ustawienie trybu dostarczania producenta komunikatów JMS ustawia trwałość komunikatu IBM MQ opublikowanego w określonym temacie.

Upewnij się, że używany jest produkt **DeliveryMode**.Trwałe (PERSISTENT) dla komunikatów, które muszą zostać zachowane po zakończeniu działania menedżera kolejek lub w przypadku wyłączenia.

## subskrypcja

### Trwałość subskrypcji

Kanały AMQP obsługują tworzenie trwałych subskrypcji przy użyciu trwałych wersji metod konsumenta tworzenia produktu JMS :

- **`createDurableConsumer()`**
- **`createSharedDurableConsumer()`**

### Tryb potwierdzania sesji

Aby zagwarantować, że zużyty komunikat zostanie w pełni przetworzony przed usunięciem go z kolejki subskrypcji IBM MQ , należy utworzyć sesję JMS przy użyciu programu **`Session`**.Tryb CLIENT_ACKNOWLEDGE i użycie metody **`message.acknowledge()`** w celu potwierdzenia tego komunikatu i wszystkich innych komunikatów odebranych wcześniej w tej sesji.

### Pojęcia pokrewne

Tworzenie aplikacji klienckich AMQP
Obsługa interfejsu API AMQP przez produkt IBM MQ umożliwia administratorowi produktu IBM MQ utworzenie kanału AMQP. Po uruchomieniu kanał ten definiuje numer portu, który akceptuje połączenia z aplikacji klienckich AMQP.

## V 9.3.3 Usuwanie potwierdzonych komunikatów AMQP z kolejki w partiach

Jeśli aplikacja AMQP używa dostarczania komunikatów QOS_AT_LEAST_ONCE (1), usługa AMQP oczekuje na potwierdzenie od aplikacji przed usunięciem kopii komunikatu, która jest zachowywana po wysłaniu tego komunikatu do aplikacji. W produkcie IBM MQ 9.3.3potwierdzone komunikaty są usuwane z kolejki w partiach, a nie pojedynczo, co powoduje zwiększenie wydajności.

## O tym zadaniu

W systemach Long Term Support i Continuous Delivery przed IBM MQ 9.3.3każdy komunikat jest usuwany z kolejki osobno.

W produkcie IBM MQ 9.3.3można użyć dwóch właściwości systemowych **`com.ibm.mq.AMQP.BATCHSZ`** i **`com.ibm.mq.AMQP.BATCHINT`** , aby dostroić przetwarzanie potwierdzeń w partiach w celu zwiększenia wydajności:

**com.ibm.mq.AMQP.BATCHSZ**
Ten atrybut definiuje maksymalną liczbę potwierdzeń, które mogą zostać odebrane, zanim usługa AMQP usunie komunikaty. Liczba może należeć do zakresu od 1 do 9999. Jeśli ustawiono niepoprawną liczbę lub jeśli podana liczba jest poza zakresem, zostanie użyta wartość domyślna 50.

Wielkość zadania wsadowego nie ma wpływu na sposób przesyłania komunikatów. Komunikaty są zawsze przesyłane indywidualnie, ale są usuwane w zadaniu wsadowym po odebraniu potwierdzeń przez usługę AMQP. Rzeczywista wielkość zadania wsadowego może być mniejsza niż wartość określona w parametrze **`com.ibm.mq.AMQP.BATCHINT`**. Na przykład zadanie wsadowe zostanie zakończone, jeśli upłynie okres ustawiony przez atrybut **`com.ibm.mq.AMQP.BATCHINT`** .

**com.ibm.mq.AMQP.BATCHINT**
Ten atrybut definiuje czas (w milisekundach), przez który usługa AMQP przechowuje potwierdzone komunikaty w kolejce. Jeśli zadanie wsadowe nie jest pełne, zostanie ono wyczyszczone po upływie tego czasu. Można podać dowolną liczbę milisekund, od 1 do 999 999 999. Wartością domyślną jest 50. Jeśli wartość tego atrybutu nie zostanie podana, zostanie użyta wartość domyślna 50.

**Uwagi:**

1. To, czy usługa AMQP oczekuje na potwierdzenie przed usunięciem komunikatu, zależy od tego, która z dwóch następujących jakości usługi jest używana przez aplikację do dostarczania komunikatów:
   - QOS dla QOS_AT_MOST_ONCE (0)

Jeśli aplikacja AMQP używa tej jakości usługi, nie potwierdza ona komunikatów, dlatego usługa AMQP usuwa komunikaty po ich wysłaniu do aplikacji bez oczekiwania na potwierdzenie.

- QOS_AT_LEAST_ONCE (1)

  Jeśli aplikacja AMQP używa tej jakości usługi, potwierdza komunikaty, więc usługa AMQP zachowuje kopię każdego komunikatu po wysłaniu go do aplikacji, dopóki nie otrzyma potwierdzenia od aplikacji. Jeśli aplikacja rozłączy się lub utraci połączenie przed potwierdzeniem komunikatu, komunikat zostanie udostępniony innym aplikacjom. Usługa AMQP nie usuwa komunikatu z kolejki, dopóki nie zostanie potwierdzona.

2. MQ Appliance Właściwości systemowe **com.ibm.mq.AMQP.BATCHSZ** i **com.ibm.mq.AMQP.BATCHINT** nie mają zastosowania w systemie IBM MQ Appliance. W systemie IBM MQ Applianceużywana jest wartość domyślna 50.

.

## Procedura

Właściwości systemowe **com.ibm.mq.AMQP.BATCHSZ** i **com.ibm.mq.AMQP.BATCHINT** umożliwiają dostrojenie przetwarzania potwierdzeń w zadaniach wsadowych.

Po utworzeniu menedżera kolejek w programie IBM MQ 9.3.3plik `amqp_java.properties` zawiera następujące wartości domyślne właściwości systemu:

```
-Dcom.ibm.mq.AMQP.BATCHSIZE=50
-Dcom.ibm.mq.AMQP.BATCHINT=50
```

W zależności od użytej częstotliwości komunikatów można dostroić przetwarzanie potwierdzeń w partiach w celu zwiększenia wydajności. Migrowany menedżer kolejek nie ma tych właściwości w pliku `amqp_java.properties` . Dlatego w przypadku migrowanego menedżera kolejek lub jeśli właściwości nie są ustawione, używane są wartości domyślne. Można dodać te właściwości, aby dostroić wartości w celu zoptymalizowania wydajności.

Potwierdzone komunikaty są usuwane w partiach, gdy spełniony jest jeden z następujących warunków:

- Liczba potwierdzonych komunikatów osiągnęła wartość **com.ibm.mq.AMQP.BATCHSZ**.
- Wartość **com.ibm.mq.AMQP.BATCHINT** jest przekraczana po uruchomieniu zadania wsadowego.
- Aplikacja rozłącza lub zamyka kolejkę lub temat przed spełnieniem dwóch poprzednich warunków.

# ALW Topologie dla klientów AMQP z produktem IBM MQ

Przykładowe topologie ułatwiające tworzenie klientów AMQP do pracy z produktem IBM MQ.

**Zadania pokrewne**

Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

## ALW Klienty AMQP komunikujące się za pośrednictwem produktu IBM MQ

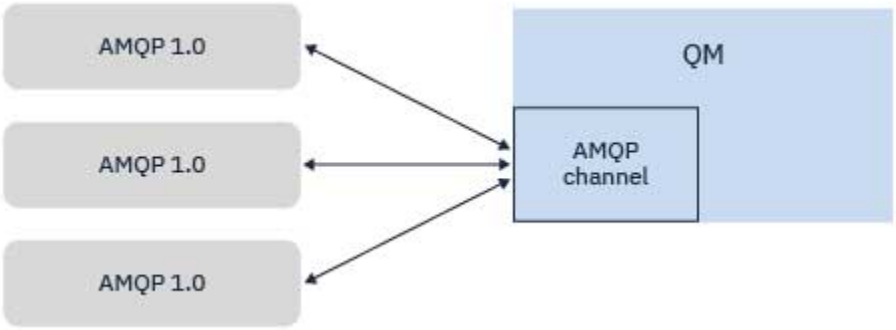
Produkt IBM MQ może być używany jako dostawca przesyłania komunikatów dla dowolnej aplikacji, która jest zgodna ze specyfikacją AMQP 1.0. Chociaż każdy klient AMQP 1.0 może nawiązać połączenie z kanałem AMQP, niektóre funkcje AMQP nie są obsługiwane, na przykład transakcje lub wiele sesji.

Definiując co najmniej jeden kanał AMQP, klienty AMQP 1.0 mogą nawiązywać połączenie z menedżerem kolejek i wysyłać komunikaty do łańcucha tematu. Klienty mogą również subskrybować wzorzec tematu w celu odbierania komunikatów zgodnych ze wzorcem.

W poniższym scenariuszu jedynymi aplikacjami, które wysyłają i odbierają komunikaty, są aplikacje AMQP 1.0 .

Aplikacje mogą zdecydować, czy miejsca docelowe utworzone przez subskrybowanie łańcucha tematu są trwałe, aby komunikaty nie były tracone, jeśli aplikacja tymczasowo utraci połączenie z menedżerem kolejek.

Aplikacje mogą również wybrać czas przechowywania komunikatów przed ich wyczyszczeniem z miejsca docelowego.




**Zadania pokrewne**

Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

## ALW Klienty AMQP wymieniające komunikaty z aplikacjami produktu IBM MQ

Definiując i uruchamiając kanał AMQP, aplikacje AMQP 1.0 mogą publikować komunikaty odbierane przez istniejące aplikacje produktu MQ . Wszystkie komunikaty publikowane za pośrednictwem kanału AMQP są wysyłane do tematów produktu MQ , a nie do kolejek produktu MQ . Aplikacja MQ , która utworzyła subskrypcję za pomocą wywołania API MQSUB, odbiera komunikaty publikowane przez aplikacje AMQP 1.0 , pod warunkiem, że łańcuch tematu lub obiekt tematu używany przez aplikację MQ jest zgodny z łańcuchem tematu opublikowanym przez klient AMQP.

Dane komunikatu AMQP, atrybuty i właściwości są ustawiane w komunikacie MQ odebranym przez aplikację MQ . Więcej informacji na temat odwzorowań komunikatów AMQP na produkt MQ zawiera sekcja “Odwzorowanie pól AMQP na pola produktu IBM MQ (komunikaty przychodzące)” na stronie 708.

Jeśli aplikacja MQ utworzyła trwałą subskrypcję, komunikaty publikowane przez aplikację AMQP są przechowywane w kolejce, która jest jej podparta. Komunikaty są następnie odbierane przez aplikację MQ , gdy aplikacja wznowi swoją subskrypcję. Jeśli aplikacja AMQP określa czas życia komunikatu, a aplikacja MQ nie połączy się ponownie w czasie życia, komunikat utraci ważność w kolejce.

Aplikacje AMQP 1.0 mogą również odbierać komunikaty publikowane przez istniejące aplikacje produktu MQ . Komunikaty publikowane przez aplikacje produktu MQ w temacie lub łańcuchu tematu produktu MQ są odbierane przez aplikację AMQP 1.0 , pod warunkiem że aplikacja zasubskrybowała wzorzec tematu zgodny z opublikowanym łańcuchem tematu.

Jeśli aplikacja AMQP 1.0 określa wartość czasu życia dla subskrypcji, a aplikacja AMQP rozłącza się na czas dłuższy niż czas życia, subskrypcja traci ważność w menedżerze kolejek i wszystkie komunikaty przechowywane w kolejce subskrypcji są tracone.

Pola MQMD, właściwości komunikatu i dane aplikacji są ustawiane w komunikacie AMQP odebranym przez aplikację AMQP. Więcej informacji na temat odwzorowań komunikatów produktu MQ na protokół AMQP zawiera sekcja “Odwzorowanie pól IBM MQ na pola AMQP (komunikaty wychodzące)” na stronie 706.

**Zadania pokrewne**

Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

# ALW Konfigurowanie klientów AMQP do bezpośredniej interakcji z aplikacjami w kolejkach produktu IBM MQ

V 9.3.0 Implementacja produktu IBM MQ AMQP obsługuje publikowanie/subskrybowanie i punkt z punktem. Dla każdego klienta AMQP, który nie obsługuje wskazywania punktu, wykonaj następujące kroki, aby wysłać komunikaty do kolejki lub odebrać komunikaty z kolejki.

## Przegląd

Na przykład załóżmy, że istnieje aplikacja pobierająca komunikaty z kolejki wejściowej IN_QUEUE i umieszczająca te komunikaty w kolejce wyjściowej OUT_QUEUE. Klienty AMQP mogą umieszczać komunikaty w produkcie IN_QUEUEi pobrać komunikaty z produktu OUT_QUEUE .

**Uwaga:** W samej aplikacji nie są wymagane żadne zmiany.


Aby publikator AMQP umieścił komunikat w kolejce, należy utworzyć subskrypcję administracyjną dla łańcucha tematu, w którym jest publikowany klient AMQP, z miejscem docelowym żądanej kolejki. Patrz sekcja “Wysyłanie komunikatów do aplikacji:” na stronie 719.

Aby subskrybent AMQP mógł pobrać komunikaty z kolejki, należy zastąpić kolejkę kolejką aliasową o takiej samej nazwie i miejscem docelowym obiektu tematu reprezentującego łańcuch tematu subskrybowany przez klienta AMQP. Patrz sekcja “Pobieranie komunikatów z aplikacji:” na stronie 719

## Wysyłanie komunikatów do aplikacji:

Aplikacja już odbiera komunikaty z produktu IN_QUEUE i klient AMQP ma mieć możliwość publikowania komunikatów, aby mogły one zostać przetworzone przez aplikację do tej kolejki.

W tym celu należy utworzyć nową subskrypcję administracyjną, w której łańcuch tematu, z którego ta subskrypcja odbiera komunikaty, jest łańcuchem tematu publikowany przez klienta AMQP. Kolejka docelowa tej subskrypcji jest kolejką wejściową dla aplikacji IN_QUEUE.

Wszystkie komunikaty publikowane w zdefiniowanym łańcuchu tematu dla tej subskrypcji administracyjnej są kierowane do zdefiniowanego miejsca docelowego (w tym przypadku jest to IN_QUEUE).

Zakładając, że klient AMQP publikuje w łańcuchu tematu /application/in, można utworzyć subskrypcję administracyjną APP_INprzy użyciu następującej komendy MQSC:

```
DEF SUB(APP_IN) TOPICSTR('/application/in') DEST('IN_QUEUE')
```

Po zdefiniowaniu tego obiektu wszystkie komunikaty publikowane w /application/in są kierowane do miejsca docelowego IN_QUEUE, w którym są pobierane przez aplikację w taki sam sposób, jak inne komunikaty umieszczane w tej kolejce przez inne aplikacje.

## Pobieranie komunikatów z aplikacji:

Aplikacja umieszcza komunikaty w pliku OUT_QUEUE, gdzie mogą być one pobierane i przetwarzane przez inne klienty.

Jednak w takim przypadku komunikaty mają być dostarczane do klienta AMQP, ale klienty AMQP używają tylko publikowania/subskrybowania i nie mogą odbierać komunikatów bezpośrednio z kolejki.

Aby zastąpić klienty, które wcześniej odebrały komunikat, subskrybowanym klientem AMQP, należy utworzyć obiekt tematu dla łańcucha tematu, który został zasubskrybowany przez klienta AMQP, oraz kolejkę aliasową.

**Ostrzeżenie:** Jeśli użytkownik zdefiniuje kolejkę aliasową, a następnie uruchomi aplikację wysyłającą, zanim klienty AMQP będą miały możliwość subskrybowania, komunikaty wysyłane przez aplikację wysyłającą do kolejki (teraz temat) zostaną utracone, ponieważ nie ma żadnych subskrybentów.

Zmiany opisane w tym tekście zastępują klienty poprzednio otrzymujące komunikaty tylko subskrybowanym klientem AMQP. Aby można było korzystać z kombinacji klientów AMQP i innych klientów w celu pobierania komunikatów, należy wprowadzić bardziej szczegółowe zmiany.

Przyjmując, że klient AMQP subskrybuje łańcuch tematu `/application/out`, można zdefiniować obiekt tematu APP_OUT przy użyciu następującej komendy MQSC:

```
DEF TOPIC(APP_OUT) TOPICSTR('/application/out')
```

Wszystkie komunikaty dostarczane do tego obiektu tematu są dostarczane do klienta AMQP subskrybującego ten sam łańcuch tematu.

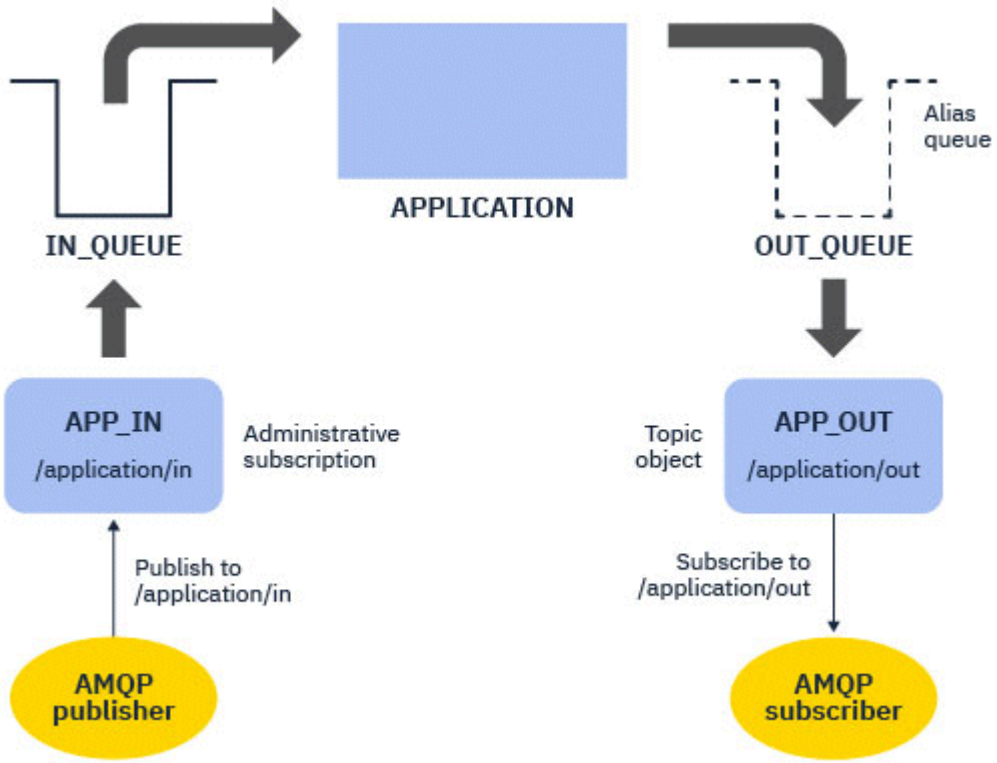
Następnie należy upewnić się, że komunikaty umieszczane w programie OUT_QUEUE przez aplikację są dostarczane do tego nowego obiektu tematu, aby były wysyłane do klienta subskrybującego.

W tym celu należy zastąpić istniejącą kolejkę OUT_QUEUE kolejką aliasową o takiej samej nazwie typem docelowym utworzonego właśnie obiektu tematu, używając następującej komendy MQSC:

```
DEF QALIAS(OUT_QUEUE) TARGTYPE(TOPIC) TARGET(APP_OUT)
```

Teraz komunikaty umieszczane przez aplikację w kolejce OUT_QUEUE nie oczekują na odebrany w kolejce. Zamiast tego są dostarczane do miejsca docelowego tej kolejki aliasowej, czyli do nowego obiektu tematu APP_OUT.

Klient AMQP subskrybujący łańcuch tematu reprezentowany przez ten obiekt tematu `/application/out`otrzymuje z kolejki aliasowej wszystkie komunikaty wysłane do tego obiektu tematu.




**Zadania pokrewne**

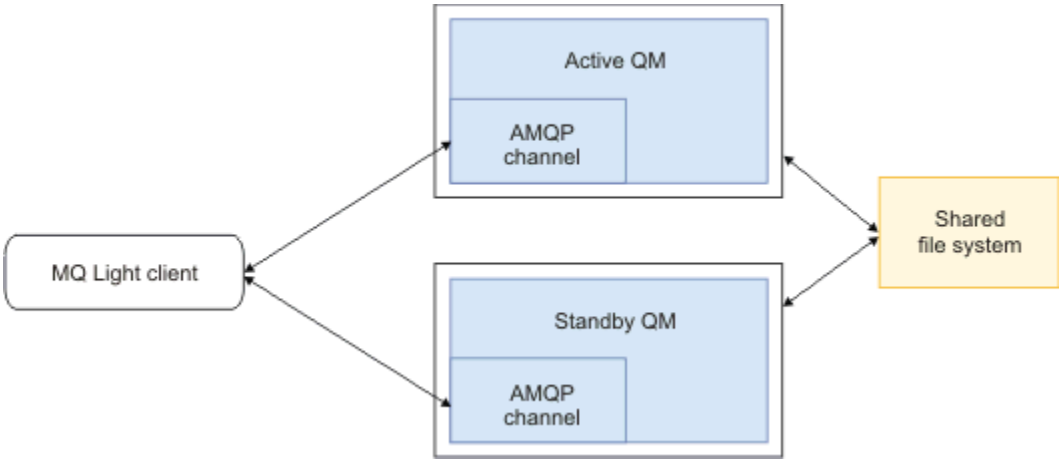
Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

## ALW Konfigurowanie klienta AMQP na potrzeby wysokiej dostępności

Aplikacje AMQP 1.0 można skonfigurować w celu nawiązania połączenia z aktywną instancją menedżera kolejek z wieloma instancjami IBM MQ i przełączenia awaryjnego do instancji rezerwowej menedżera kolejek z wieloma instancjami w parze wysokiej dostępności (HA). W tym celu należy skonfigurować aplikację AMQP z dwoma parami adresów IP i portów.

Interfejs API klienta AMQP można skonfigurować przy użyciu funkcji niestandardowej, która jest wywoływana, jeśli klient utraci połączenie z serwerem. Funkcja może połączyć się z alternatywnym adresem IP, na przykład z rezerwowym menedżerem kolejek IBM MQ lub z oryginalnym adresem IP. W przypadku innych klientów AMQP, jeśli klient obsługuje konfigurację wielu punktów końcowych połączenia, należy skonfigurować aplikację z dwiema parami host-port i użyć funkcji ponownego połączenia udostępnianych przez bibliotekę AMQP w celu przełączenia do rezerwowego menedżera kolejek.




**Zadania pokrewne**

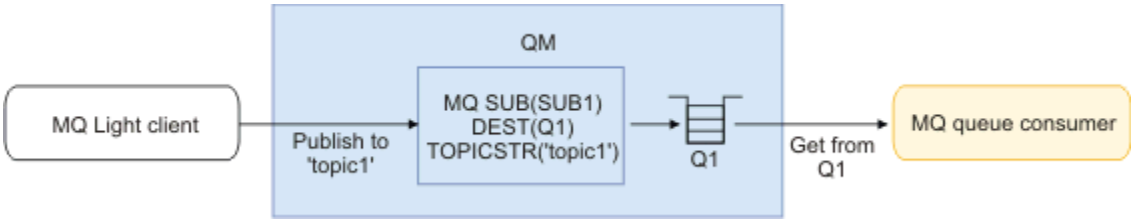
Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

## ALW Konfigurowanie publikowania/subskrybowania dla klientów AMQP

Klienty AMQP mogą publikować w temacie z subskrypcją produktu IBM MQ , która kieruje komunikaty do kolejki produktu IBM MQ odczytywanej przez istniejącą aplikację. Jeśli aplikacja AMQP 1.0 ma wysyłać komunikaty do istniejącej aplikacji produktu IBM MQ , która jest skonfigurowana do odczytywania z kolejki, należy zdefiniować administrowaną subskrypcję programu IBM MQ w menedżerze kolejek.

Skonfiguruj subskrypcję tak, aby używała wzorca tematu zgodnego z łańcuchem tematu używanym przez aplikację AMQP. Ustaw miejsce docelowe subskrypcji na nazwę kolejki, z której aplikacja IBM MQ pobiera lub przegląda komunikaty.




**Zadania pokrewne**

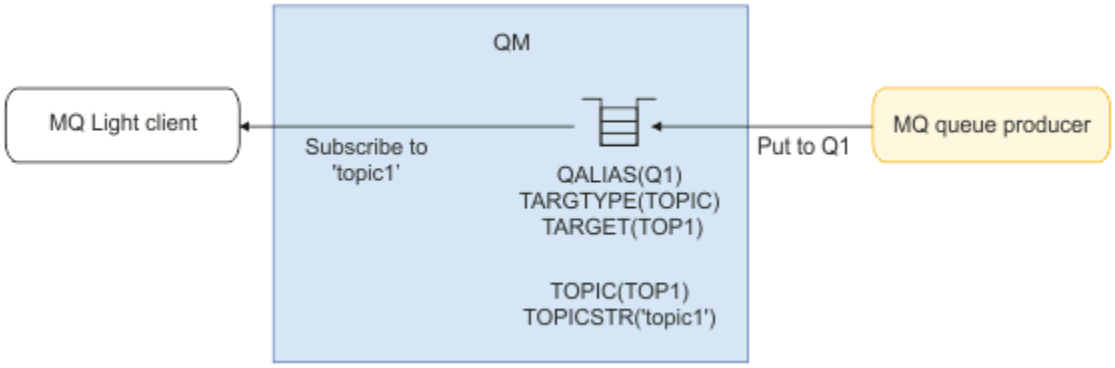
Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

## ALW Klient AMQP używający aliasu kolejki do odbierania komunikatów z aplikacji produktu IBM MQ

Klient AMQP może subskrybować temat i odbierać komunikaty umieszczane w kolejce aliasowej przez aplikację IBM MQ . Jeśli aplikacja AMQP 1.0 ma odbierać komunikaty z istniejącej aplikacji produktu IBM MQ , która jest skonfigurowana do umieszczania komunikatów w kolejce, należy zdefiniować alias kolejki (QALIAS) w menedżerze kolejek.

Alias kolejki musi mieć taką samą nazwę jak kolejka, którą aplikacja IBM MQ otwiera do umieszczenia. Alias kolejki musi określać typ podstawowy TOPIC i obiekt podstawowy obiektu tematu IBM MQ , który zawiera łańcuch tematu zgodny ze wzorcem tematu zasubskrybowanym przez aplikację AMQP.




**Zadania pokrewne**

Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

## ALW Klient AMQP wysyłający żądania do serwera aplikacji i odbierający odpowiedzi z serwera aplikacji

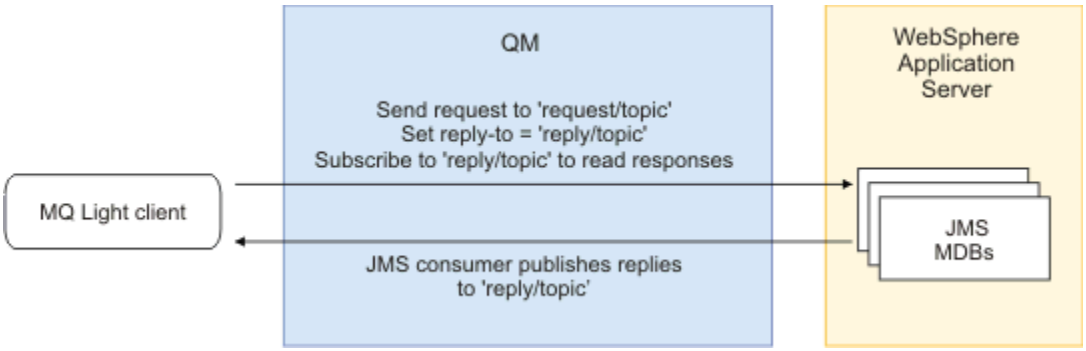
Klient AMQP może wprowadzać żądania do komponentu bean sterowanego komunikatami działającego na serwerze aplikacji i odbierać odpowiedzi z tematu odpowiedzi. Produkt IBM MQ obsługuje aplikacje AMQP 1.0 , które ustawiają temat odpowiedzi w komunikatach publikowanej przez produkt IBM MQ . Gdy komunikat AMQP jest publikowany z ustawionym atrybutem reply-to, wartość pola reply-to jest ustawiana jako właściwość JMS dla odbieranych konsumentów JMS. To ustawienie umożliwia konsumentom JMS odczytywanie tematu odpowiedzi z komunikatu i wysyłanie komunikatu odpowiedzi z powrotem do klienta AMQP.

Właściwość JMS to **JMSReplyTo**. Łańcuch odpowiedzi AMQP musi mieć jeden z następujących typów:

- Łańcuch tematu. Na przykład `'reply/topic'`
- Adres AMQP URL w postaci `amqp://host:port/[topic-string]`. Na przykład: `amqp://localhost:5672/reply/topic`

Jeśli jako pole odpowiedzi zostanie podany adres AMQP URL , wszystkie elementy oprócz łańcucha tematu na końcu adresu URL są usuwane przed ustawieniem właściwości **JMSReplyTo** .

Więcej informacji na temat odwzorowań adresu zwrotnego AMQP na właściwość **JMSReplyTo** zawiera sekcja “Odwzorowanie pól AMQP na pola produktu IBM MQ (komunikaty przychodzące)” na stronie 708




**Zadania pokrewne**

Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

## ALW Współdziałanie między aplikacjami MQ Light i Apache Qpid JMS

Aplikacje MQ Light i Apache Qpid JMS działają w podobny sposób, a subskrybując temat, należy utworzyć subskrypcje programu IBM MQ zgodne z tą samą konwencją nazewnictwa.

## Prywatna, niewspółużytkowana subskrypcja

Nazwa subskrypcji programu IBM MQ utworzonej przez aplikację to `:private:<clientid>:<topicstring>`.

Aplikacja używająca innego identyfikatora klienta nie może uzyskać dostępu do subskrypcji utworzonych przez inne aplikacje, ponieważ nazwa subskrypcji jest generowana automatycznie i zawiera identyfikator klienta AMQP.

Zarówno aplikacje Apache Qpid JMS , jak i MQ Light używają tej konwencji nazewnictwa dla subskrypcji prywatnych.

## Subskrypcje współużytkowane globalnie

Nazwa globalnie współużytkowanej subskrypcji produktu IBM MQ utworzonej przez klient AMQP to `:share:<sharename>:<topicstring>`.

Jeśli kilka aplikacji o różnych identyfikatorach klienta AMQP określa tę samą nazwę zasobu współużytkowanego i ten sam łańcuch tematu, współużytkują one pojedynczą subskrypcję i mogą współpracować w celu przetwarzania komunikatów dla tej subskrypcji. Tego wzorca można użyć, aby skalować liczbę aplikacji procesów roboczych, które wysysają komunikaty z subskrypcji.

Zarówno aplikacje Apache Qpid JMS , jak i MQ Light używają tej konwencji nazewnictwa dla globalnie współużytkowanych subskrypcji. W przypadku serwera Apache Qpid JMSwymagane jest, aby połączenie JMS nie miało określonego identyfikatora klienta.

Biblioteka Apache Qpid JMS automatycznie generuje identyfikator klienta AMQP, ale ten identyfikator klienta nie jest używany na potrzeby nazewnictwa subskrypcji produktu IBM MQ .

**Uwaga:** Globalnie współużytkowane subskrypcje są nadal ograniczone do pojedynczego menedżera kolejek.

## Prywatne subskrypcje współużytkowane

Nazwa prywatnej współużytkowanej subskrypcji produktu IBM MQ utworzonej przez klient AMQP to `:privateshare:<clientid>:<sharename>:<topicstring>`.

Jeśli kilka wątków z pojedynczej aplikacji Apache Qpid JMS używa tej samej nazwy zasobu współużytkowanego i tego samego łańcucha tematu, a identyfikator klienta został skonfigurowany dla połączenia JMS , wątki te współużytkują ten sam obiekt subskrypcji IBM MQ .

Jednak inne połączenia Apache Qpid JMS nie mogą współużytkować subskrypcji, ponieważ muszą używać innego identyfikatora klienta.

Klienty MQ Light nie obsługują prywatnej subskrypcji współużytkowanej i nie mogą korzystać z komunikatów z prywatnej subskrypcji współużytkowanej utworzonej przez aplikację Apache Qpid JMS .

## Subskrypcje programu IBM MQ JMS

Subskrypcje produktu IBM MQ JMS używają innego schematu nazewnictwa niż kanały AMQP. Aplikacje MQ Light lub Apache Qpid JMS nie mogą współużytkować subskrypcji z aplikacjami produktu IBM MQ JMS .

**Pojęcia pokrewne**

Tworzenie aplikacji klienckich AMQP

Obsługa interfejsu API AMQP przez produkt IBM MQ umożliwia administratorowi produktu IBM MQ utworzenie kanału AMQP. Po uruchomieniu kanał ten definiuje numer portu, który akceptuje połączenia z aplikacji klienckich AMQP.

## ALW Właściwości elementu sterującego programu nasłuchującego AMQP produktu IBM MQ

Aby zwiększyć wydajność aplikacji wielowątkowej, można dostroić liczbę wątków roboczych, które powinny być używane przez usługę AMQP, konfigurując właściwość w pliku właściwości AMQP.

Właściwości usługi nasłuchiwania AMQP można skonfigurować w następujących plikach właściwości:

- Windows W systemach Windows : `amqp_win.properties` .
- Linux AIX W systemach AIX and Linux : `amqp_unix.properties` .

Można skonfigurować następujące właściwości:

*Tabela 105. Właściwości usługi nasłuchiwania AMQP*

| Właściwość | Opis |
|---|---|
| **com.ibm.mq.MQXR.Workers** | Liczba wątków roboczych serwera utworzonych przez usługę nasłuchiwania AMQP. Jeśli ta wartość nie zostanie podana, domyślnie będzie równa liczbie procesorów logicznych w systemie. |
| **MQIBindType** | Typ powiązania dla usługi AMQP: FASTPATH, SHAREDlub SAMODZIELNE. Wartością domyślną jest FASTPATH. |

Usługa nasłuchiwania AMQP równoważy obciążenie związane z połączeniami klienckimi w wielu wątkach roboczych. Liczbę wątków roboczych, które powinny być używane przez usługę AMQP, można określić przy użyciu właściwości **com.ibm.mq.MQXR.Workers** .

Administrator menedżera kolejek produktu IBM MQ może dostroić liczbę wątków roboczych w celu zwiększenia wydajności aplikacji wielowątkowej. Zwykle najlepsza wydajność jest osiągana, gdy liczba wątków roboczych jest zgodna z liczbą procesorów logicznych w systemie. Może to jednak nie zawsze mieć miejsce w przypadku pewnych konfiguracji komputera i parametrów obciążenia klienta, dlatego w celu znalezienia optymalnej liczby wątków roboczych może być wymagany element strojenia.

Przed strojeniem należy dokładnie zapoznać się z naturą aplikacji klienckich i ich obciążeniami. Pomiar wydajności aplikacji przy użyciu różnych liczników wątków i testów porównawczych powinien pomóc w określeniu optymalnej liczby wątków roboczych.

**Uwaga:** MQ Appliance Te właściwości nie mają zastosowania w systemie IBM MQ Appliance. W systemie IBM MQ Applianceużywane są wartości domyślne.

# Tworzenie aplikacji REST przy użyciu produktu IBM MQ

Aplikacje REST można tworzyć w celu wysyłania i odbierania komunikatów. Produkt IBM MQ obsługuje różne interfejsy REST API w zależności od platformy i możliwości.

Następujące opcje są obsługiwanymi przez IBM MQ opcjami, które można wybrać w celu wysyłania i odbierania komunikatów z produktu IBM MQ:

- IBM MQ messaging REST API
- IBM z/OS Connect EE
- IBM Integration Bus
- DataPower

### IBM MQ messaging REST API

Za pomocą messaging REST API można wysyłać, odbierać i przeglądać komunikaty IBM MQ w formacie zwykłego tekstu. Opcja messaging REST API jest domyślnie włączona.

Dostępna jest obsługa wielu różnych nagłówków HTTP , których można używać do ustawiania wspólnych właściwości komunikatu.

Produkt messaging REST API jest w pełni zintegrowany z zabezpieczeniami systemu IBM MQ . Aby można było używać messaging REST API, użytkownicy muszą być uwierzytelniani na serwerze mqweb i muszą mieć przypisaną rolę MQWebUser .

Więcej informacji zawiera sekcja “Przesyłanie komunikatów przy użyciu programu REST API” na stronie 727. Patrz także Kurs: pierwsze kroki z przesyłaniem komunikatów IBM MQ REST API w produkcie IBM Developer, który zawiera przykłady użycia interfejsu REST API przesyłania komunikatów (Go and Node.js ).

## IBM z/OS Connect EE

IBM z/OS Connect EE to produkt z/OS , który umożliwia budowanie interfejsów REST API na podstawie istniejących zasobów aplikacyjnych produktu z/OS , takich jak transakcje CICS lub IMS , oraz kolejek i tematów produktu IBM MQ . Istniejący zasób z/OS jest ukryty przed użytkownikiem. Dzięki temu można włączyć usługi REST, nie zmieniając ich ani żadnych istniejących aplikacji, które z nich korzystają.

Produkt IBM z/OS Connect EE udostępnia automatyczną transformację danych w celu translacji między danymi JSON używanymi przez interfejsy API REST a bardziej tradycyjnymi strukturami języka, na przykład języka COBOL, oczekiwanymi przez wiele aplikacji mainframe.

Pakiet API produktu IBM z/OS Connect EE oparty na platformie Eclipse może być używany do budowania obszernego interfejsu API zgodnego ze specyfikacją REST, w którym używane są parametry zapytania i segmenty ścieżki URL , manipulując formatem JSON przepływającym przez środowisko wykonawcze produktu IBM z/OS Connect EE.

Produkt IBM z/OS Connect EE może być używany do ujawniania kolejek i tematów systemu IBM MQ jako interfejsów API zgodnych ze specyfikacją REST za pośrednictwem dostawcy usług IBM MQ . Obsługiwane są dwa różne typy usług:

- Usługi jednokierunkowe: udostępniają interfejs REST API, który umożliwia wykonanie pojedynczej operacji IBM MQ w kolejce lub temacie. W zależności od dokładnej konfiguracji żądanie HTTP może spowodować wysłanie komunikatu do kolejki lub opublikowanie go w temacie. W przeciwnym razie żądanie HTTP może spowodować destrukcyjne odebranie komunikatu z kolejki.
- Usługi dwukierunkowe: udostępniają interfejs REST API na bazie pary kolejek używanych przez aplikację typu żądanie-odpowiedź zaplecza. Programy wywołujące wysyłają żądanie HTTP do usługi dwukierunkowej. Ładunek żądania HTTP jest transformowany z formatu JSON na tradycyjną strukturę języka i umieszczany w kolejce żądań, w której jest przetwarzany przez aplikację zaplecza oraz w odpowiedzi umieszczonej w kolejce odpowiedzi. Ta odpowiedź jest pobierana przez usługę, przekształcana ze struktury tradycyjnego języka w format JSON i wysyłana z powrotem do programu wywołującego jako treść odpowiedzi POST.

Więcej informacji na temat produktu IBM z/OS Connect EE zawiera sekcja z/OS Connect EE.

Więcej informacji na temat dostawcy usług IBM MQ zawiera sekcja Korzystanie z dostawcy usług IBM MQ.

## IBM Integration Bus

IBM Integration Bus to wiodąca technologia integracji firmy IBM, która może być używana do łączenia aplikacji i systemów bez względu na obsługiwane przez nie formaty komunikatów i protokoły.

Produkt IBM Integration Bus zawsze obsługiwał interfejs IBM MQ i udostępnia węzły *HTTPInput* i *HTTPRequest* , których można używać do tworzenia interfejsu zgodnego ze specyfikacją REST na podstawie produktu IBM MQ, a także wiele innych systemów, takich jak bazy danych.

Produkt IBM Integration Bus może być używany do znacznie większej liczby czynności niż udostępnienie prostego interfejsu REST na serwerze IBM MQ. Jego możliwości mogą być używane do zaawansowanego manipulowania ładunkiem, wzbogacania ładunku i wielu innych udoskonaleń w ramach REST API.

Więcej informacji na ten temat zawiera przykład technologii , który udostępnia interfejs JSON over REST na potrzeby aplikacji IBM MQ , która oczekuje ładunku XML.

### DataPower

Brama DataPower jest pojedynczą wielokanałową bramą, która zapewnia bezpieczeństwo, kontrolę, integrację i zoptymalizowany dostęp do szeregu systemów, w tym IBM MQ. Jest dostępny zarówno w formie sprzętowej, jak i wirtualnej.

Jedną z usług udostępnianych przez produkt DataPower jest brama wieloprotokołowa, która może przyjmować dane wejściowe w jednym protokole i generować dane wyjściowe w innym protokole. W szczególności produkt DataPower można skonfigurować w taki sposób, aby akceptował dane HTTP(S) i kierował je do produktu IBM MQ za pośrednictwem połączenia klienckiego, które może być używane do budowania interfejsu REST na bazie produktu IBM MQ. Inne usługi DataPower , takie jak transformacja, mogą być również używane w celu rozszerzenia interfejsu REST.

Więcej informacji na ten temat zawiera sekcja Multi-Protocol Gateway.

## Przesyłanie komunikatów przy użyciu programu REST API

Za pomocą programu messaging REST API można wykonywać proste przesyłanie komunikatów w trybie punkt z punktem i publikowanie. Istnieje możliwość publikowania komunikatów w temacie, wysyłania komunikatów do kolejki, przeglądania komunikatów w kolejce i niszczycielskiego pobierania komunikatów z kolejki. Informacje są wysyłane i odbierane z messaging REST API w formacie zwykłego tekstu.

### Zanim rozpoczniesz

**Uwaga:**

- Opcja messaging REST API jest domyślnie włączona. Można wyłączyć messaging REST API , aby uniemożliwić przesyłanie komunikatów. Więcej informacji na temat włączania i wyłączania messaging REST APIzawiera sekcja Konfigurowanie messaging REST API.
- Produkt messaging REST API jest zintegrowany z zabezpieczeniami systemu IBM MQ . Aby można było używać messaging REST API, użytkownicy muszą być uwierzytelniani na serwerze mqweb i muszą mieć przypisaną rolę MQWebUser . Użytkownik musi również mieć uprawnienia dostępu do określonej kolejki lub tematu. Więcej informacji na temat zabezpieczeń dla produktu REST API zawiera sekcja Zabezpieczenia IBM MQ Console i REST API.
- Jeśli produkt Advanced Message Security (AMS) jest używany z produktem messaging REST API, należy zauważyć, że wszystkie komunikaty są szyfrowane przy użyciu kontekstu serwera mqweb, a nie kontekstu użytkownika, który opublikuje komunikat.
- Podczas odbierania lub przeglądania komunikatu obsługiwane są tylko komunikaty w formacie IBM MQ `MQSTR` lub `JMS TextMessage` . Następnie wszystkie komunikaty są destrukcyjnie odbierane w punkcie synchronizacji, a wszystkie nieobsłużone komunikaty pozostają w kolejce. Kolejkę IBM MQ można skonfigurować w taki sposób, aby przenoś te komunikaty nieprzetwarzalne do alternatywnego miejsca docelowego. Więcej informacji zawiera sekcja “Obsługa komunikatów nieprzetwarzalnych w produkcie IBM MQ classes for JMS” na stronie 240.
- Produkt messaging REST API nie zapewnia jednorazowego dostarczania komunikatów z obsługą transakcyjną. Jeśli zostanie wysłane żądanie HTTP POST i połączenie nie powiedzie się przed odebraniem przez klienta odpowiedzi HTTP , klient nie będzie mógł natychmiast stwierdzić, czy komunikat został wysłany do określonej kolejki, czy opublikowany w określonym temacie. Jeśli zostanie wykonane żądanie HTTP DELETE i połączenie nie powiedzie się przed odebraniem przez klienta odpowiedzi HTTP , komunikat mógł zostać usunięty ze zniszczonej kolejki i utracony, ponieważ nie ma sposobu na wycofanie destrukcyjnego pobrania.
- V 9.3.0 W pliku IBM MQ 9.3.0znaki nowego wiersza w łańcuchach przychodzących nie są już usuwane przez operację HTTP POST. Aplikacje REST, które używają wcześniejszych wersji, nie powinny używać nowych wierszy w komunikatach, które są wysyłane lub publikowane przy użyciu interfejsu API REST, ponieważ zostaną utracone.

## Procedura

- “Pierwsze kroki w produkcie messaging REST API” na stronie 728
- “Korzystanie z messaging REST API” na stronie 731
- REST API obsługa błędów
- wykrywanieREST API
- REST API -obsługa języków narodowych

**Odsyłacze pokrewne**

Informacje dodatkowe dotyczące przesyłania komunikatów REST API

**Informacje pokrewne**

Kurs: pierwsze kroki z przesyłaniem komunikatów produktu IBM MQ REST API

# Pierwsze kroki w produkcie messaging REST API

Szybkie rozpoczęcie pracy z produktem messaging REST API i wypróbowanie kilku przykładowych komend za pomocą komendy cURL.

## Zanim rozpoczniesz

Aby rozpocząć pracę z produktem messaging REST API, przykłady w tej czynności mają następujące wymagania:

- W przykładach użyto komendy cURL do wysyłania żądań REST w celu umieszczenia i pobrania komunikatów z kolejki. Dlatego, aby wykonać to zadanie, należy zainstalować w systemie program cURL .
- W przykładach używany jest menedżer kolejek `QM1`. Utwórz menedżer kolejek o takiej samej nazwie lub zastąp istniejący menedżer kolejek w systemie. Menedżer kolejek musi znajdować się na tym samym komputerze co serwer mqweb.
- Aby wykonać tę czynność, użytkownik musi mieć pewne uprawnienia pozwalające na użycie komendy **dspmqweb**:
  - z/OS W systemie z/OS użytkownik musi mieć uprawnienia do uruchamiania komendy **dspmqweb** i dostęp z uprawnieniami do zapisu do pliku `mqwebuser.xml`.
  - Multi W przypadku wszystkich innych systemów operacyjnych użytkownik musi być użytkownikiem uprzywilejowanym.

IBM i W systemach IBM ikomendy powinny być uruchamiane w powłoce QSHELL.

## Procedura

1. Upewnij się, że serwer mqweb jest skonfigurowany dla produktu messaging REST API:
   - Upewnij się, że serwer mqweb został skonfigurowany do użycia przez produkt administrative REST API, produkt administrative REST API dla systemu MFT, produkt messaging REST APIlub produkt IBM MQ Console.Więcej informacji na temat konfigurowania serwera mqweb z podstawowym rejestrem zawiera sekcja Podstawowa konfiguracja serwera mqweb.
   - Jeśli serwer mqweb jest już skonfigurowany, upewnij się, że w kroku 5 sekcji Konfiguracja podstawowa dla serwera mqwebdodano odpowiednich użytkowników w celu włączenia przesyłania komunikatów.
     - Jeśli parametr **mqRestMessagingAdoptWebUserContext** jest ustawiony na wartość `true` w konfiguracji serwera mqweb, użytkownicy produktu messaging REST API muszą być członkami roli `MQWebUser` . Role `MQWebAdmin` i `MQWebAdminRO` nie mają zastosowania do messaging REST API. Użytkownicy muszą również mieć uprawnienia dostępu do kolejek i tematów, które są używane do przesyłania komunikatów za pośrednictwem OAM lub RACF.

– Jeśli właściwość **mqRestMessagingAdoptWebUserContext** jest ustawiona na wartość `false` w konfiguracji serwera mqweb, identyfikator użytkownika używany do uruchamiania serwera mqweb musi być autoryzowany do uzyskiwania dostępu do kolejek używanych na potrzeby przesyłania komunikatów za pośrednictwem usługi OAM lub RACF.

2. z/OS
W systemie z/OSustaw zmienną środowiskową WLP_USER_DIR tak, aby można było używać komendy **dspmqweb** . Należy ustawić zmienną tak, aby wskazywała konfigurację serwera mqweb, wprowadzając następującą komendę:

```
export WLP_USER_DIR=WLP_user_directory
```

, gdzie *WLP_user_directory* jest nazwą katalogu, który jest przekazywany do `crtmqweb`. Na przykład:

```
export WLP_USER_DIR=/var/mqm/web/installation1
```

Więcej informacji na ten temat zawiera sekcja Tworzenie serwera mqweb.

3. Określ adres URL REST API , wprowadzając następującą komendę:

```
dspmqweb status
```

W przykładach w poniższych krokach przyjęto założenie, że REST API URL jest domyślnym URL `https://localhost:9443/ibmmq/rest/v2/`. Jeśli używany jest adres URL inny niż domyślny, należy podać go w poniższych krokach.

4. Utwórz kolejkę `MSGQ`w menedżerze kolejek `QM1`. Ta kolejka jest używana na potrzeby przesyłania komunikatów. Zastosuj jedną z następujących metod:

- Użyj żądania POST dla zasobu `mqsc` w pliku administrative REST API, uwierzytelniając się jako użytkownik `mqadmin` :

```
curl -k https://localhost:9443/ibmmq/rest/v2/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\"}"
```

- Użyj komend MQSC:

  z/OSW systemie z/OSzamiast komendy **runmqsc** należy użyć źródła 2CR . Więcej informacji na ten temat zawiera sekcja Źródła, z których można wydawać komendy MQSC i PCF w systemie IBM MQ for z/OS.

  a. Uruchom program **runmqsc** dla menedżera kolejek, wprowadzając następującą komendę:

  ```
  runmqsc QM1
  ```

  b. Użyj komendy **DEFINE QLOCAL** MQSC, aby utworzyć kolejkę:

  ```
  DEFINE QLOCAL(MSGQ)
  ```

  c. Wyjdź z programu **runmqsc** , wprowadzając następującą komendę:

  ```
  end
  ```

5. Nadaj uprawnienie użytkownikowi, który został dodany do pliku `mqwebuser.xml` w kroku 5 sekcji Konfiguracja podstawowa dla serwera mqweb , aby uzyskać dostęp do kolejki `MSGQ`. Zastąp użytkownika w miejscu, w którym jest używany produkt `myuser` :

- z/OSW systemie z/OS:

  a. Nadaj użytkownikowi dostęp do kolejki:

```
RDEFINE MQQUEUE hlq.MSGQ UACC(NONE)
PERMIT hlq.MSGQ CLASS(MQQUEUE) ID(MYUSER) ACCESS(UPDATE)
```

b. Nadaj identyfikatorowi użytkownika uruchomionego zadania mqweb dostęp do ustawiania całego kontekstu w kolejce:

```
RDEFINE MQADMIN hlq.CONTEXT.MSGQ UACC(NONE)
PERMIT hlq.CONTEXT.MSGQ CLASS(MQADMIN) ID(mqwebStartedTaskID) ACCESS(CONTROL)
```

- Multi We wszystkich innych systemach operacyjnych, jeśli użytkownik należy do grupy mqm, uprawnienie jest już nadane. W przeciwnym razie wprowadź następujące komendy:

a. Uruchom program **runmqsc** dla menedżera kolejek, wprowadzając następującą komendę:

```
runmqsc QM1
```

b. Użyj komendy **SET AUTHREC** MQSC, aby nadać użytkownikowi uprawnienia do przeglądania, sprawdzania, pobierania i umieszczania w kolejce:

```
SET AUTHREC PROFILE(MSGQ) OBJTYPE(QUEUE) +
PRINCIPAL(myuser) AUTHADD(BROWSE, INQ, GET, PUT)
```

c. Wyjdź z programu **runmqsc** , wprowadzając następującą komendę:

```
end
```

6. Umieść komunikat z treścią `Hello World!` w kolejce `MSGQ` w menedżerze kolejek `QM1`, używając żądania POST dla zasobu `message` . Należy zastąpić identyfikator i hasło użytkownika z pliku `mqwebuser.xml` dla produktów `myuser` i `mypassword`:

   Używane jest uwierzytelnianie podstawowe, a nagłówek `ibm-mq-rest-csrf-token` HTTP z dowolną wartością jest ustawiany w żądaniu REST cURL . Ten dodatkowy nagłówek jest wymagany dla żądań POST, PATCH i DELETE.

```
curl -k https://localhost:9443/ibmmq/rest/v2/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. Niszcząco pobierz komunikat z kolejki `Hello World!` w kolejce `MSGQ` w menedżerze kolejek `QM1`, używając żądania DELETE dla zasobu `message` . Należy zastąpić identyfikator i hasło użytkownika z pliku `mqwebuser.xml` dla produktów `myuser` i `mypassword`:

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X DELETE
-u myuser:mypassword -H "ibm-mq-rest-csrf-token: value"
```

   Zostanie zwrócony komunikat `Hello World!` .

## Co dalej

- W przykładach do zabezpieczenia żądania użyto uwierzytelniania podstawowego. Zamiast tego można użyć uwierzytelniania opartego na znacznikach lub uwierzytelniania opartego na kliencie. Więcej informacji na ten temat zawiera sekcja Korzystanie z uwierzytelniania przy użyciu certyfikatu klienta za pomocą interfejsu REST API i konsoli IBM MQ Console oraz sekcja Korzystanie z uwierzytelniania opartego na znacznikach za pomocą interfejsu REST API.
- Więcej informacji na temat używania messaging REST API i tworzenia adresów URL z parametrami zapytania: “Korzystanie z messaging REST API” na stronie 731.
- Jeśli używana jest konsola messaging REST API, połączenia z menedżerem kolejek są zestawiane w celu zoptymalizowania wydajności. Można skonfigurować maksymalną wielkość puli oraz działanie, które jest wykonywane, gdy wszystkie połączenia w puli są używane: Konfigurowanie messaging REST API.
- Przejrzyj informacje uzupełniające dotyczące dostępnych zasobów messaging REST API i wszystkich dostępnych parametrów zapytania: messaging REST API odwołanie.

- Wykryj administrative REST API, interfejs RESTful dla administrowania IBM MQ : Administrowanie za pomocą REST API.
- Wykrywanie IBM MQ Console, interfejsu GUI opartego na przeglądarce: Administration za pomocą IBM MQ Console.

## Korzystanie z messaging REST API

Jeśli używana jest metoda messaging REST API, należy wywołać metody HTTP dla adresów URL w celu wysyłania i odbierania komunikatów IBM MQ . Metoda HTTP , na przykład POST, reprezentuje typ działania, które ma zostać wykonane na obiekcie reprezentowanym przez URL. Dodatkowe informacje o działaniu mogą być zakodowane w parametrach zapytania. Informacje o wyniku wykonania działania mogą zostać zwrócone jako treść odpowiedzi HTTP .

### Zanim rozpoczniesz

Przed użyciem programu messaging REST APInależy rozważyć następujące zagadnienia:

- Aby można było używać produktu messaging REST API, należy uwierzytelnić się na serwerze mqweb. Uwierzytelnianie można przeprowadzić przy użyciu podstawowego uwierzytelniania HTTP , uwierzytelniania przy użyciu certyfikatu klienta lub uwierzytelniania opartego na znaczniku. Więcej informacji na temat sposobu korzystania z tych metod uwierzytelniania zawiera sekcja IBM MQ Console i REST API zabezpieczenia.
- W pliku REST API rozróżniana jest wielkość liter. Na przykład żądanie HTTP POST dla następującego adresu URL powoduje błąd, jeśli menedżer kolejek ma nazwę `qmgr1`.

  ```
  /ibmmq/rest/v2/messaging/qmgr/QMGR1/queue/Q1/message
  ```

- V 9.3.3 W przypadku nawiązywania połączenia ze zdalnym menedżerem kolejek za pomocą programu messaging REST APIzamiast nazwy menedżera kolejek należy użyć nazwy unikalnej dla połączenia menedżera kolejek.
- Nie wszystkie znaki, które mogą być używane w nazwach obiektów IBM MQ , mogą być bezpośrednio zakodowane w URL. Aby poprawnie zakodować te znaki, należy użyć odpowiedniego kodowania adresu URL :
  - Ukośnik musi być zakodowany jako %2F.
  - Znak procentu musi być zakodowany jako %25.
  - Kropka musi być zakodowana jako %2E.
  - Znak zapytania musi być zakodowany jako %3F.
- Podczas odbierania lub przeglądania komunikatu obsługiwane są tylko komunikaty w formacie IBM MQ `MQSTR` i `JMS TextMessage` . Następnie wszystkie komunikaty są destrukcyjnie odbierane w punkcie synchronizacji, a wszystkie nieobsłużone komunikaty pozostają w kolejce. Kolejkę IBM MQ można skonfigurować w taki sposób, aby przenoś te komunikaty nieprzetwarzalne do alternatywnego miejsca docelowego. Więcej informacji zawiera sekcja “Obsługa komunikatów nieprzetwarzalnych w produkcie IBM MQ classes for JMS” na stronie 240.

### O tym zadaniu

Jeśli do wykonania działania przesyłania komunikatów na obiekcie kolejki produktu IBM MQ używany jest REST API , należy najpierw utworzyć URL reprezentujący ten obiekt. Każdy URL rozpoczyna się przedrostkiem opisującym nazwę hosta i port, do którego ma zostać wysłane żądanie. Pozostała część URL opisuje konkretny obiekt lub trasę do tego obiektu, znanego jako zasób.

Działanie przesyłania komunikatów, które ma zostać wykonane na zasobie, określa, czy URL wymaga parametrów zapytania, czy nie. Definiuje także używaną metodę HTTP oraz określa, czy dodatkowe informacje są wysyłane na adres URL, czy zwracane z tego adresu. Informacje dodatkowe mogą stanowić część żądania HTTP lub mogą zostać zwrócone jako część odpowiedzi HTTP .

Po utworzeniu URLmożna wysłać żądanie HTTP do IBM MQ. Żądanie można wysłać przy użyciu implementacji HTTP wbudowanej w wybrany język programowania. Żądanie można również wysłać za pomocą narzędzi wiersza komend, takich jak cURL, przeglądarki WWW lub programu dodatkowego przeglądarki WWW.

**Ważne:** Należy wykonać co najmniej kroki “1.a” na stronie 732 i “1.b” na stronie 732.

## Procedura

1. Utwórz URL:
   a) Określ przedrostek URL , wprowadzając następującą komendę:

   ```
   dspmqweb status
   ```

   URL , który ma być używany, zawiera frazę `/ibmmq/rest/` .

   b) Dodaj zasoby kolejki i powiązanego menedżera kolejek, które mają być używane na potrzeby przesyłania komunikatów, do ścieżki URL .

   W odwołaniu do mechanizmu przesyłania komunikatów segmenty zmiennych mogą być identyfikowane w polu URL za pomocą nawiasów otaczających *{ }*. Więcej informacji na ten temat zawiera sekcja /messaging/qmgr/{qmgrName}/queue/{queueName}/message.

   Na przykład w celu interakcji z kolejką *Q1* powiązaną z menedżerem kolejek *QM1*należy dodać łańcuch `/qmgr` i `/queue` do przedrostka URL w celu utworzenia następującego URL:

   ```
   https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message
   ```

   **Wskazówka:** V 9.3.3 Jeśli menedżer kolejek jest zdalnym menedżerem kolejek, należy użyć unikalnej nazwy menedżera kolejek zamiast nazwy menedżera kolejek. Zdalny menedżer kolejek musi być skonfigurowany, zanim będzie mógł być używany z programem messaging REST API. Więcej informacji na ten temat zawiera sekcja “Konfigurowanie zdalnego menedżera kolejek do użycia z programem messaging REST API” na stronie 732.

   c) Opcjonalne: Dodaj opcjonalny parametr zapytania do URL.

   Dodaj znak zapytania,?, parametr zapytania, znak równości = oraz wartość URL.

   Na przykład, aby poczekać maksymalnie 30 sekund na udostępnienie następnego komunikatu, utwórz następujący URL:

   ```
   https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message?wait=30000
   ```

   d) Opcjonalne: Dodaj dalsze opcjonalne parametry zapytania do adresu URL.

   Dodaj znak ampersand & do adresu URL, a następnie powtórz krok 1c.
2. Wywołaj odpowiednią metodę HTTP dla URL. Określ dowolny opcjonalny ładunek komunikatu i podaj odpowiednie referencje zabezpieczeń do uwierzytelnienia. Na przykład:
   - Użyj implementacji HTTP/REST wybranego języka programowania.
   - Użyj narzędzia, takiego jak program dodatkowy przeglądarki klienta REST lub program cURL.

## V 9.3.3 Konfigurowanie zdalnego menedżera kolejek do użycia z programem messaging REST API

Program messaging REST API umożliwia nawiązywanie połączeń ze zdalnymi menedżerami kolejek w celu przesyłania komunikatów. Przed nawiązaniem połączenia ze zdalnym menedżerem kolejek należy skonfigurować zdalny menedżer kolejek. Następnie można nawiązać połączenie ze zdalnym menedżerem kolejek przy użyciu nazwy unikalnej zdefiniowanej w informacjach konfiguracyjnych.

## Zanim rozpoczniesz

- Upewnij się, że serwer mqweb został skonfigurowany do użycia przez produkt administrative REST API, produkt administrative REST API dla systemu MFT, produkt messaging REST APIlub produkt IBM MQ Console.Więcej informacji na temat konfigurowania serwera mqweb z podstawowym rejestrem zawiera sekcja Podstawowa konfiguracja serwera mqweb.
- Jeśli serwer mqweb jest już skonfigurowany, upewnij się, że w kroku 5 sekcji Konfiguracja podstawowa dla serwera mqwebdodano odpowiednich użytkowników w celu włączenia przesyłania komunikatów. Użytkownicy messaging REST API muszą być członkami roli `MQWebUser` . Role `MQWebAdmin` i `MQWebAdminRO` nie mają zastosowania do messaging REST API.
  - Jeśli właściwość **mqRestMessagingAdoptWebUserContext** jest ustawiona na wartość `true` w konfiguracji serwera mqweb, użytkownicy o roli `MQWebUser` muszą być autoryzowani do uzyskiwania dostępu do kolejek i tematów, które są używane na potrzeby przesyłania komunikatów za pośrednictwem menedżera OAM lub produktu RACF.
  - Jeśli parametr **mqRestMessagingAdoptWebUserContext** ma wartość `false` w konfiguracji serwera mqweb, identyfikator użytkownika używany do uruchamiania serwera mqweb musi być autoryzowany do uzyskiwania dostępu do kolejek i tematów, które są używane na potrzeby przesyłania komunikatów za pośrednictwem menedżera OAM lub produktu RACF.
- Upewnij się, że program messaging REST API jest skonfigurowany do nawiązywania połączeń ze zdalnymi menedżerami kolejek. Więcej informacji na ten temat zawiera sekcja Konfigurowanie trybu połączenia dla messaging REST API.

## O tym zadaniu

Połączenie ze zdalnymi menedżerami kolejek można nawiązać za pomocą programu messaging REST API. Zdalny menedżer kolejek może być menedżerem kolejek w innym systemie, menedżerem kolejek w innej instalacji lub menedżerem kolejek w tej samej instalacji co serwer mqweb.

Aby nawiązać połączenie ze zdalnym menedżerem kolejek, należy wykonać następujące kroki konfiguracyjne:

- Skonfiguruj kanał połączenia z serwerem i program nasłuchujący.
- Nadaj odpowiednie uprawnienia użytkownikowi, aby mógł uzyskać dostęp do menedżera kolejek.
- Utwórz plik CCDT zawierający informacje o połączeniu dla menedżera kolejek.
- Dodaj informacje o połączeniu do pliku messaging REST API za pomocą komendy **setmqweb remote** .

Następnie można użyć zdalnego menedżera kolejek, podając unikalną nazwę w zasobie URL zamiast nazwy menedżera kolejek.

Można również skonfigurować zdalne menedżery kolejek jako część grupy menedżerów kolejek. Więcej informacji na ten temat zawiera “Konfigurowanie grupy menedżerów kolejek do użycia z interfejsem REST API przesyłania komunikatów” na stronie 736.

## Procedura

1. W zdalnym menedżerze kolejek utwórz kanał połączenia z serwerem, aby umożliwić zdalne połączenia z menedżerem kolejek. Kanały połączenia z serwerem można tworzyć za pomocą komendy **DEFINE CHANNEL** MQSC w wierszu komend.
   Na przykład, aby utworzyć kanał połączenia z serwerem `QM1.SVRCONN` dla menedżera kolejek QM1, wprowadź następującą komendę:

   ```
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   ```

   Więcej informacji na temat **DEFINE CHANNEL** i dostępnych opcji zawiera sekcja DEFINE CHANNEL.
2. Upewnij się, że odpowiedni użytkownik jest autoryzowany do uzyskania dostępu do menedżera kolejek. Ten użytkownik musi również mieć uprawnienia dostępu do wszystkich kolejek lub tematów używanych na potrzeby przesyłania komunikatów. Użytkownik musi mieć uprawnienia `connect`, `inquire`, `alternate user`i `set context` w menedżerze kolejek. W systemie UNIX, Linux, and

Windows należy użyć komendy sterującej **setmqaut** w wierszu komend. W systemie z/OSzdefiniuj profile RACF , aby nadać autoryzowanemu użytkownikowi dostęp do menedżera kolejek.
Na przykład w systemie UNIX, Linux, and Windows, aby autoryzować użytkownika `exampleUserw` celu uzyskania dostępu do menedżera kolejek QM1, należy wprowadzić następującą komendę:

```
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
```

Więcej informacji na temat użytkowników, którzy muszą być autoryzowani, zawiera sekcja “Określanie nazwy użytkownika używanej przez messaging REST API” na stronie 739.

3. ALW
Jeśli w zdalnym menedżerze kolejek nie istnieje program nasłuchujący, utwórz program nasłuchujący, który będzie akceptował przychodzące połączenia sieciowe, używając w wierszu komend komendy MQSC produktu **DEFINE LISTENER** .
Na przykład, aby utworzyć program nasłuchujący `REMOTE.LISTENER` na porcie 1414 dla zdalnego menedżera kolejek QM1, wprowadź następującą komendę:

```
runmqsc QM1
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
end
```

4. Upewnij się, że program nasłuchujący jest uruchomiony, używając komendy **START LISTENER** MQSC w wierszu komend:

ALW Na przykład w systemie AIX, Linux, and Windows , aby uruchomić program nasłuchujący `REMOTE.LISTENER` dla menedżera kolejek QM1, wprowadź następującą komendę:

```
runmqsc QM1
START LISTENER(REMOTE.LISTENER)
end
```

z/OS Na przykład w systemie z/OS, aby uruchomić program nasłuchujący, wprowadź następującą komendę:

```
runmqsc QM1
START LISTENER TRPTYPE(TCP) PORT(1414)
end
```

Przestrzeń adresowa inicjatora kanału musi zostać uruchomiona przed uruchomieniem programu nasłuchującego w systemie z/OS.

5. W systemie, w którym działa serwer mqweb udostępniający produkt messaging REST API , utwórz lub zaktualizuj plik JSON CCDT zawierający informacje o połączeniu z menedżerem kolejek.

Plik CCDT musi zawierać informacje `name`, `clientConnection`oni `type` . Opcjonalnie można dołączyć informacje dodatkowe, takie jak `transmissionSecurity` . Więcej informacji na temat wszystkich definicji atrybutów kanału CCDT zawiera sekcja Pełna lista definicji atrybutów kanału CCDT.

W poniższym przykładzie przedstawiono podstawowy plik JSON CCDT dla połączenia ze zdalnym menedżerem kolejek. Ustawia on nazwę kanału na taką samą nazwę, jak nazwa przykładowego kanału połączenia z serwerem utworzonego w kroku “1” na stronie 733. Port połączenia jest ustawiony na tę samą wartość, co port używany przez program nasłuchujący. Host połączenia jest ustawiony na nazwę hosta systemu, w którym działa zdalny menedżer kolejek QM1.

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "example.com",
            "port": 1414
          }],
        "queueManager": "QM1"
      },
      "type": "clientConnection"
```

```
    }]
}
```

6. Z poziomu instalacji, w której działa serwer mqweb udostępniający produkt messaging REST API, użyj komendy **setmqweb remote** , aby dodać informacje o zdalnym menedżerze kolejek do konfiguracji serwera mqweb.

   Należy podać co najmniej następujące parametry:

   - **-qmgrName**, w którym określana jest nazwa menedżera kolejek.
   - **-ccdtURL**, gdzie należy określić URL tabeli CCDT dla menedżera kolejek.
   - **-uniqueName**, gdzie należy podać unikalną nazwę dla menedżera kolejek. Nazwa unikalna jest używana do rozróżniania zdalnych menedżerów kolejek, które mogą mieć taką samą nazwę i dlatego nie może istnieć w celu zidentyfikowania innego menedżera kolejek.

   Można określić kilka innych opcji, takich jak nazwa użytkownika i hasło, które mają być używane na potrzeby połączenia ze zdalnym menedżerem kolejek, lub szczegóły magazynu zaufanych certyfikatów i magazynu kluczy. Pełną listę parametrów, które można podać w komendzie **setmqweb remote** , zawiera sekcja setmqweb remote.

   Aby na przykład dodać zdalny menedżer kolejek QM1z przykładowym plikiem CCDT, należy wprowadzić następującą komendę:

```
setmqweb remote add -uniqueName "RemoteQM1" -qmgrName "QM1" -ccdtURL "c:\myccdts\ccdt.json"
```

## Wyniki

Zdalny menedżer kolejek może być używany z programem messaging REST API przy użyciu nazwy unikalnej w polu URL zasobu zamiast nazwy menedżera kolejek.

### Przykład

W poniższym przykładzie przedstawiono konfigurowanie połączenia zdalnego menedżera kolejek dla menedżera kolejek QM1. Użytkownik IBM MQ Console autoryzowany do administrowania menedżerem kolejek na podstawie autoryzacji, która jest nadawana użytkownikowi exampleUser. Referencje tego użytkownika są udostępniane IBM MQ Console , gdy program **setmqweb remote** jest używany do konfigurowania połączenia z menedżerem kolejek.

1. W systemie, w którym znajduje się zdalny menedżer kolejek QM1 , zostanie utworzony kanał połączenia z serwerem i program nasłuchujący. Program nasłuchujący został uruchomiony, a użytkownik `exampleUser` ma autoryzację do nawiązania połączenia z menedżerem kolejek i uzyskania dostępu do kolejki używanej na potrzeby przesyłania komunikatów:

```
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
```

2. W systemie, w którym działa serwer mqweb, tworzony jest plik `QM1_ccdt.json` zawierający następujące informacje o połączeniu:

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "example.com",
            "port": 1414
          }],
```

```
        "queueManager": "QM1"
      },
      "type": "clientConnection"
    }]
}
```

3. W systemie, w którym działa serwer mqweb, informacje o połączeniu dla menedżera kolejek QM1 są dodawane do serwera mqweb. Informacje autoryzacyjne dla `exampleUser` są uwzględniane w informacjach o połączeniu:

```
setmqweb remote add -uniqueName "MACHINEAQM1" -qmgrName "QM1" -ccdtURL
"c:\myccdts\QM1_ccdt.json" -username "exampleUser" -password "password"
```

4. Program messaging REST API może nawiązać połączenie ze zdalnym menedżerem kolejek QM1 przy użyciu unikalnej nazwy połączenia menedżera kolejek zamiast nazwy menedżera kolejek w zasobie URL:

```
curl -k https://localhost:9443/ibmmq/rest/v2/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!"
```

## V 9.3.3 Konfigurowanie grupy menedżerów kolejek do użycia z interfejsem REST API przesyłania komunikatów

Program messaging REST API umożliwia nawiązywanie połączeń z grupami menedżerów kolejek w celu przesyłania komunikatów. Przed nawiązaniem połączenia z grupą menedżerów kolejek należy skonfigurować zdalny menedżer kolejek dla grupy. Następnie można nawiązać połączenie z grupą menedżerów kolejek przy użyciu nazwy unikalnej, która jest zdefiniowana w informacjach konfiguracyjnych.

### Zanim rozpoczniesz

- Upewnij się, że serwer mqweb został skonfigurowany do użycia przez produkt administrative REST API, produkt administrative REST API dla systemu MFT, produkt messaging REST APIlub produkt IBM MQ Console.Więcej informacji na temat konfigurowania serwera mqweb z podstawowym rejestrem zawiera sekcja Podstawowa konfiguracja serwera mqweb.
- Jeśli serwer mqweb jest już skonfigurowany, upewnij się, że w kroku 5 sekcji Konfiguracja podstawowa dla serwera mqwebdodano odpowiednich użytkowników w celu włączenia przesyłania komunikatów. Użytkownicy messaging REST API muszą być członkami roli `MQWebUser` . Role `MQWebAdmin` i `MQWebAdminRO` nie mają zastosowania do messaging REST API.
  - Jeśli właściwość **`mqRestMessagingAdoptWebUserContext`** ma wartość `true` w konfiguracji serwera mqweb, użytkownicy z rolą `MQWebUser` muszą być autoryzowani do uzyskiwania dostępu do kolejek i tematów używanych na potrzeby przesyłania komunikatów. Tych użytkowników można autoryzować za pomocą OAM lub RACF.
  - Jeśli parametr **`mqRestMessagingAdoptWebUserContext`** ma wartość `false` w konfiguracji serwera mqweb, identyfikator użytkownika, który uruchamia serwer mqweb, musi być autoryzowany do uzyskiwania dostępu do kolejek i tematów używanych na potrzeby przesyłania komunikatów. Tego użytkownika można autoryzować za pomocą OAM lub RACF.
- Upewnij się, że program messaging REST API jest skonfigurowany do nawiązywania połączeń ze zdalnymi menedżerami kolejek. Więcej informacji na ten temat zawiera sekcja Konfigurowanie trybu połączenia dla messaging REST API .

### O tym zadaniu

Grupa menedżerów kolejek umożliwia łączenie aplikacji z dowolnym menedżerem kolejek w grupie. Grupa jest definiowana jako zestaw połączeń w tabeli definicji kanału klienta (CCDT). W przypadku użycia wywołania MQCONN lub MQCONNX należy odwołać się do grupy, poprzedzając nazwą menedżera kolejek gwiazdką. Za pomocą parametru messaging REST APImożna odwołać się do grupy przy użyciu nazwy

unikalnej powiązanej z grupą menedżerów kolejek. Unikalna nazwa jest dołączana do zasobu URL zamiast nazwy menedżera kolejek. Więcej informacji na temat grup menedżerów kolejek zawiera sekcja “Grupy menedżerów kolejek w tabeli definicji kanału klienta” na stronie 952.

Można również indywidualnie skonfigurować zdalne menedżery kolejek. Więcej informacji na ten temat zawiera “Konfigurowanie zdalnego menedżera kolejek do użycia z programem messaging REST API” na stronie 732.

## Procedura

1. W każdym zdalnym menedżerze kolejek w grupie utwórz kanał połączenia z serwerem, aby umożliwić zdalne połączenia z menedżerem kolejek. Do utworzenia kanałów połączenia z serwerem można użyć komendy **DEFINE CHANNEL** MQSC w wierszu komend.
   Na przykład, aby utworzyć kanał połączenia z serwerem `QM1.SVRCONN` dla menedżera kolejek QM1, wprowadź następującą komendę:

   ```
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   ```

   Więcej informacji na temat **DEFINE CHANNEL** i dostępnych opcji zawiera sekcja DEFINE CHANNEL.
2. W każdym zdalnym menedżerze kolejek w grupie upewnij się, że odpowiedni użytkownik jest autoryzowany do uzyskania dostępu do menedżera kolejek. Ten użytkownik musi również mieć uprawnienia dostępu do wszystkich kolejek lub tematów używanych na potrzeby przesyłania komunikatów. Użytkownik musi mieć uprawnienia `connect`, `inquire`, `alternate user`i `set context` w menedżerze kolejek. W systemie UNIX, Linux, and Windows należy użyć komendy sterującej **setmqaut** w wierszu komend. W systemie z/OSzdefiniuj profile RACF , aby nadać autoryzowanemu użytkownikowi dostęp do menedżera kolejek.
   Na przykład w systemie UNIX, Linux, and Windowswprowadź następującą komendę, aby autoryzować użytkownika `exampleUser`w celu uzyskania dostępu do menedżera kolejek QM1:

   ```
   setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
   ```

   Więcej informacji na temat użytkowników, którzy muszą być autoryzowani, zawiera sekcja “Określanie nazwy użytkownika używanej przez messaging REST API” na stronie 739.
3. ALW
   Jeśli w każdym zdalnym menedżerze kolejek w grupie nie ma żadnego programu nasłuchującego, utwórz programy nasłuchujące, aby akceptowały przychodzące połączenia sieciowe. Do tworzenia obiektów nasłuchiwania można użyć komendy MQSC produktu **DEFINE LISTENER** w wierszu komend.
   Na przykład, aby utworzyć program nasłuchujący `REMOTE.LISTENER` na porcie 1414 dla zdalnego menedżera kolejek QM1, wprowadź następującą komendę:

   ```
   runmqsc QM1
   DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
   end
   ```
4. Dla każdego zdalnego menedżera kolejek w grupie upewnij się, że program nasłuchujący jest uruchomiony, używając w wierszu komend komendy MQSC produktu **START LISTENER** .

   ALW Na przykład w systemie AIX, Linux, and Windows , aby uruchomić program nasłuchujący `REMOTE.LISTENER` dla menedżera kolejek QM1, wprowadź następującą komendę:

   ```
   runmqsc QM1
   START LISTENER(REMOTE.LISTENER)
   end
   ```

   z/OS Na przykład w systemie z/OS, aby uruchomić program nasłuchujący, wprowadź następującą komendę:

   ```
   runmqsc QM1
   START LISTENER TRPTYPE(TCP) PORT(1414)
   end
   ```

Przestrzeń adresowa inicjatora kanału musi zostać uruchomiona przed uruchomieniem programu nasłuchującego w systemie z/OS.

5. W systemie, w którym działa serwer mqweb udostępniający produkt messaging REST API , utwórz plik JSON CCDT. Ten plik JSON zawiera informacje o połączeniu dla każdego menedżera kolejek w grupie.

   Plik CCDT musi zawierać informacje `name`, `clientConnection`i `type` dla każdego połączenia menedżera kolejek. Opcjonalnie można dołączyć informacje dodatkowe, takie jak `transmissionSecurity` . Więcej informacji na temat wszystkich definicji atrybutów kanału CCDT zawiera sekcja Pełna lista definicji atrybutów kanału CCDT.

   W poniższym przykładzie przedstawiono podstawowy plik JSON CCDT dla dwóch połączeń menedżera kolejek. Pierwsze połączenie dotyczy menedżera kolejek QM1. Ma kanał połączenia z serwerem `QM1.SVRCONN`, program nasłuchujący na porcie 1414i działa na hoście `QM1.example.com`. Drugie połączenie jest przeznaczone dla menedżera kolejek QM2. Ma kanał połączenia z serwerem `QM2.SVRCONN`, program nasłuchujący na porcie 1415i działa na hoście `QM2.example.com`. Ponieważ jednak połączenia są częścią grupy menedżerów kolejek QMGRP, w polu **queueManager** dla obu połączeń jest ustawiona nazwa grupy menedżerów kolejek.

```
{
  "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"
    }]
}
```

6. Z poziomu instalacji, w której działa serwer mqweb udostępniający produkt messaging REST API, użyj komendy **setmqweb remote** , aby dodać grupę menedżera kolejek do konfiguracji serwera mqweb.

   Należy podać co najmniej następujące parametry:

   - **-qmgrName**, gdzie należy podać nazwę grupy menedżera kolejek.
   - **-ccdtURL**, gdzie należy określić URL tabeli CCDT dla menedżerów kolejek.
   - **-uniqueName**, gdzie należy podać unikalną nazwę identyfikującą grupę menedżerów kolejek.
   - **-group**, aby ustawić informacje o menedżerze kolejek jako informacje dla grupy.

   Można określić kilka innych opcji, takich jak nazwa użytkownika i hasło, które mają być używane dla połączenia, lub szczegóły magazynu zaufanych certyfikatów i magazynu kluczy. Pełną listę parametrów, które można podać w komendzie **setmqweb remote** , zawiera sekcja setmqweb remote.

   Aby na przykład dodać grupę menedżerów kolejek QMGRP z przykładowym plikiem CCDT, należy wprowadzić następującą komendę:

```
setmqweb remote add  -uniqueName "MyQMGRP" -qmgrName "QMGRP" -ccdtURL
"c:\myccdts\group_ccdt.json" -group
```

## Wyniki

Grupa zdalnych menedżerów kolejek może być używana w połączeniu z programem messaging REST API przy użyciu nazwy unikalnej w URLzasobu. Menedżer kolejek z grupy jest wybierany w celu wykonania

żądania, a informacje o tym, który menedżer kolejek wykonał żądanie, są zwracane w nagłówku odpowiedzi `ibm-mq-resolved-qmgr`.

## V 9.3.3 Określanie nazwy użytkownika używanej przez messaging REST API

Jeśli używany jest program messaging REST API, odpowiedni użytkownik musi być autoryzowany w celu uzyskania dostępu do menedżerów kolejek, kolejek i tematów, z którymi ma zostać nawiązane połączenie w celu przesyłania komunikatów. To, który użytkownik musi zostać autoryzowany, zależy od sposobu skonfigurowania serwera mqweb oraz od tego, czy z produktem messaging REST APIsą używane zdalne menedżery kolejek.

Domyślnie nazwą użytkownika zabezpieczeń, która jest używana do autoryzowania dostępu do menedżera kolejek, jest użytkownik uruchamiający serwer mqweb, na którym działa produkt messaging REST API. Użytkownik, który jest używany do autoryzowania dostępu do kolejek i tematów, jest zalogowany do serwera messaging REST API. Jednak połączenie z serwerem mqweb lub zdalnym menedżerem kolejek może być skonfigurowane w taki sposób, że używana jest inna nazwa użytkownika zabezpieczeń.

### Określanie nazwy użytkownika zabezpieczeń, która jest używana do nawiązywania połączenia z menedżerem kolejek

W przypadku połączeń lokalnego menedżera kolejek nazwą użytkownika zabezpieczeń używaną do nawiązywania połączenia z menedżerem kolejek jest użytkownik uruchamiający serwer mqweb, na którym działa produkt messaging REST API. W przypadku połączeń zdalnego menedżera kolejek następujące nazwy użytkowników zabezpieczeń są używane przez messaging REST API do autoryzowania dostępu do menedżera kolejek w kolejności priorytetów. Oznacza to, że jeśli użytkownicy są określani na wiele sposobów w konfiguracji zdalnego menedżera kolejek, do autoryzacji używana jest pierwsza z listy.

1. Nazwa użytkownika jest kontekstem użytkownika adoptowanego z wyjścia zabezpieczeń.
2. Nazwa użytkownika zabezpieczeń jest kontekstem użytkownika adoptowanego w regule CHLAUTH w kanale połączenia z serwerem, który jest używany do nawiązywania połączenia ze zdalnym menedżerem kolejek.
3. Nazwa użytkownika zabezpieczeń to identyfikator użytkownika, który jest uwzględniany w konfiguracji zdalnego menedżera kolejek dla produktu messaging REST API. Ten ID użytkownika jest opcjonalnie dołączany do informacji o połączeniu menedżera kolejek podczas dodawania menedżera kolejek za pomocą komendy **setmqweb remote** .
4. Nazwa użytkownika zabezpieczeń to użytkownik, który uruchamia serwer mqweb, na którym działa produkt messaging REST API.

Więcej informacji na temat konfigurowania zdalnych menedżerów kolejek do użycia z programem messaging REST APIzawiera sekcja “Konfigurowanie zdalnego menedżera kolejek do użycia z programem messaging REST API” na stronie 732.

### Określanie nazwy użytkownika zabezpieczeń, która jest używana do nawiązywania połączeń z kolejkami i tematami

Istnieje możliwość ustawienia właściwości w konfiguracji serwera mqweb w celu określenia, która nazwa użytkownika zabezpieczeń jest używana do autoryzowania połączeń z kolejkami i tematami podczas korzystania z produktu messaging REST API. Ta właściwość jest właściwością **mqRestMessagingAdoptWebUserContext** . Ustawienie tej właściwości można wyświetlić za pomocą komendy **dspmqweb properties** .

- Jeśli parametr **mqRestMessagingAdoptWebUserContext** ma wartość true, serwer messaging REST API używa do autoryzacji identyfikatora użytkownika, który jest zalogowany do serwera messaging REST API . Dlatego ID użytkownika lub ID użytkownika istniejące w konfiguracji serwera mqweb do użycia z produktem messaging REST API są nazwami użytkowników zabezpieczeń, które muszą być autoryzowane w celu uzyskania dostępu do kolejek i tematów.

- Jeśli parametr **mqRestMessagingAdoptWebUserContext** ma wartość false, serwer messaging REST API używa do autoryzacji identyfikatora użytkownika, który uruchomił serwer mqweb obsługujący serwer messaging REST API . Dlatego identyfikator użytkownika, który jest taki sam jak identyfikator użytkownika uruchamiającego serwer mqweb, który udostępnia serwer messaging REST API , musi mieć uprawnienia dostępu do kolejek i tematów.

  Jeśli kolejki i tematy znajdują się w zdalnym menedżerze kolejek, nazwa użytkownika zabezpieczeń używana na potrzeby autoryzacji może być określona przez ustawienia w konfiguracji menedżera kolejek. W kolejności priorytetów mogą być używane następujące elementy główne zabezpieczeń:

  1. Nazwa użytkownika jest kontekstem użytkownika adoptowanego z wyjścia zabezpieczeń.
  2. Nazwa użytkownika zabezpieczeń jest kontekstem użytkownika adoptowanego w regule CHLAUTH w kanale połączenia z serwerem, który jest używany do nawiązywania połączenia ze zdalnym menedżerem kolejek. Na przykład można skonfigurować regułę CHLAUTH w kanale połączenia z serwerem w celu użycia parametru MCAUSER. Następnie wszystkie połączenia są odwzorowywane na identyfikator użytkownika, który jest autoryzowany do używania menedżera kolejek.
  3. Nazwa użytkownika zabezpieczeń jest kontekstem użytkownika adoptowanego z informacji AUTHINFO menedżera kolejek. Jeśli obiekt AUTHINFO, do którego odwołuje się atrybut CONNAUTH menedżera kolejek, jest skonfigurowany do używania produktu **ADOPTCTX(yes)**, do autoryzacji kolejek i tematów używany jest także element główny zabezpieczeń używany do autoryzowania połączeń z menedżerem kolejek. Nazwą użytkownika zabezpieczeń może być na przykład identyfikator użytkownika zawarty w informacjach o połączeniu ze zdalnym menedżerem kolejek w ramach komendy **setmqweb remote** .

**Informacje pokrewne**

CHLAUTH

KONNAUTH

Właściwości komendy dspmqweb

# Tworzenie aplikacji MQI przy użyciu produktu IBM MQ

IBM MQ zapewnia obsługę języków C, Visual Basic, COBOL, Assembler, RPG, pTALi PL/I. Te języki proceduralne korzystają z interfejsu kolejki komunikatów (MQI) w celu uzyskania dostępu do usług kolejkowania komunikatów.

Szczegółowe informacje na temat pisania aplikacji w wybranym języku można znaleźć w podtematach.

Przegląd interfejsu wywołań dla języków proceduralnych zawiera sekcja Opisy wywołań. Ta sekcja zawiera listę wywołań MQI, a każde wywołanie pokazuje sposób kodowania wywołań w każdym z tych języków.

IBM MQ udostępnia pliki definicji danych, które ułatwiają pisanie aplikacji. Pełny opis znajduje się w sekcji “Pliki definicji danych IBM MQ” na stronie 741.

Aby ułatwić wybór języka proceduralnego, w którym będą kodować programy, należy wziąć pod uwagę maksymalną długość komunikatów, które będą przetwarzane przez programy. Jeśli programy będą przetwarzać tylko komunikaty o znanej maksymalnej długości, można je zakodować w dowolnym z obsługiwanych języków. Jeśli nie znasz maksymalnej długości komunikatów, które będą musiały zostać przetworzone przez programy, wybrany język będzie zależał od tego, czy piszesz aplikację CICS, IMS, czy wsadową:

**IMS i zadanie wsadowe**
: Należy zakodować programy w języku C, PL/I lub w języku asemblera, aby korzystać z narzędzi oferowanych przez te języki w celu uzyskiwania i zwalniania dowolnych ilości pamięci. Alternatywnie można zakodować programy w języku COBOL, ale aby uzyskać i zwolnić pamięć masową, należy użyć języka asemblera, języka PL/I lub podprogramów w języku C.

**CICS**
: Napisz programy w dowolnym języku obsługiwanym przez CICS. Interfejs EXEC CICS udostępnia wywołania do zarządzania pamięcią, jeśli jest to konieczne.

**Pojęcia pokrewne**

“Aplikacje obiektowe” na stronie 16
IBM MQ zapewnia obsługę systemów JMS, Java, C + + i .NET. Te języki i środowiska korzystają z modelu obiektowego IBM MQ , który udostępnia klasy udostępniające te same funkcje, co wywołania i struktury IBM MQ .

Przegląd techniczny

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć języka proceduralnego lub obiektowego. Przed rozpoczęciem projektowania i pisania aplikacji IBM MQ należy zapoznać się z podstawowymi pojęciami związanymi z produktem IBM MQ .

**Odsyłacze pokrewne**

Informacje dodatkowe o tworzeniu aplikacji

# Pliki definicji danych IBM MQ

IBM MQ udostępnia pliki definicji danych, które ułatwiają pisanie aplikacji.

Pliki definicji danych są również nazywane:

| Język | Definicje danych |
|---|---|
| C | Dołącz pliki lub pliki nagłówkowe |
| Visual Basic | Pliki modułów (tylko wersje 32-bitowe) |
| kompilatory | Kopiowanie plików |
| Assembler | Makra |
| PL/I | Dołączaj pliki |

Pliki definicji danych, które ułatwiają zapisywanie wyjść kanałów, są opisane w sekcji IBM MQ COPY, header, include i module files.

Pliki definicji danych, które pomagają w zapisaniu możliwych do zainstalowania wyjść usług, zostały opisane w sekcji “Wyjścia użytkownika, wyjścia funkcji API i instalowalne usługi systemu IBM MQ” na stronie 965.

Pliki definicji danych obsługiwane w języku C + + znajdują się w sekcji Korzystanie z języka C++.

IBM i

Informacje na temat plików definicji danych obsługiwanych w języku RPG zawiera podręcznik IBM i Application Programming Reference (ILE/RPG).

Nazwy plików definicji danych mają przedrostek CMQ i przyrostek określony przez język programowania:

| Przyrostek | Język |
|---|---|
| a | Język asembler |
| b | Visual Basic |
| c | C |
| l | COBOL (bez zainicjowanych wartości) |
| p | PL/I |
| v | COBOL (z ustawionymi wartościami domyślnymi) |

## Biblioteka instalacyjna

Nazwa **thlqual** jest kwalifikatorem wysokiego poziomu biblioteki instalacji w systemie z/OS.

W tej sekcji przedstawiono pliki definicji danych IBM MQ pod następującymi nagłówkami:

- “Pliki włączane w języku C” na stronie 742
- “Pliki modułu Visual Basic” na stronie 742
- “Pliki kopii COBOL” na stronie 742
- z/OS “Makra w języku asembler System/390” na stronie 744
- z/OS “Pliki włączane języka PL/I” na stronie 744

## Pliki włączane w języku C

IBM MQ Pliki włączane w języku C są wymienione w sekcji Pliki nagłówkowe w języku C. Są one instalowane w następujących katalogach lub bibliotekach:

| Platforma | Katalog instalacyjny lub biblioteka |
|---|---|
| IBM i IBM i | QMQM/H |
| Linux AIX AIX and Linux | *MQ_INSTALLATION_PATH*/inc/ |
| Windows Windows | *MQ_INSTALLATION_PATH*\Tools\c\include |
| z/OS z/OS | **thlqual**, SCSQC370 |

gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym zainstalowano produkt IBM MQ .

**Uwaga:** W systemie AIX and Linuxpliki włączane są dowiązane symbolicznie do pliku `/usr/include`.

Więcej informacji na temat struktury katalogów zawiera sekcja Planowanie obsługi systemu plików.

## Pliki modułu Visual Basic

IBM MQ for Windows udostępnia cztery pliki modułu Visual Basic.

Są one wymienione w sekcji Pliki modułu Visual Basic i instalowane w

```
MQ_INSTALLATION_PATH\Tools\Samples\VB\Include
```

## Pliki kopii COBOL

W przypadku języka COBOL produkt IBM MQ udostępnia oddzielne pliki kopii zawierające stałe nazwane i dwa pliki kopii dla każdej struktury.

Istnieją dwa pliki kopii dla każdej struktury, ponieważ każda z nich jest dostarczana zarówno z wartościami początkowymi, jak i bez nich:

- W sekcji WORKING-STORAGE SECTION programu w języku COBOL użyj plików, które inicjują pola struktury na wartości domyślne. Struktury te są zdefiniowane w plikach kopii, które mają nazwy z przyrostkiem litery V (wartości).
- W sekcji LINKAGE programu w języku COBOL należy użyć struktur bez wartości początkowych. Struktury te są zdefiniowane w kopiowanych plikach, których nazwy mają przyrostek L (powiązanie).

IBM i Pliki kopii zawierające definicje danych i interfejsów dla produktu IBM i są dostarczane dla programów w języku ILE COBOL przy użyciu wywołań prototypowych dla interfejsu MQI. Zbiory

istnieją w QMQM/QCBLLESRC z nazwami podzbiorów, które mają przyrostek L (dla struktur bez wartości początkowych) lub przyrostek V (dla struktur z wartościami początkowymi).

Pliki kopii języka COBOL IBM MQ są wymienione w sekcji Pliki COPY języka COBOL. Są one instalowane w następujących katalogach:

| Platforma | Katalog instalacyjny lub biblioteka |
|---|---|
| Linux AIX AIX and Linux | *MQ_INSTALLATION_PATH*/inc/ |
| IBM i IBM i | QMQM/QCBLLESRC, |
| Windows Windows | *MQ_INSTALLATION_PATH*\Tools\cobol\copybook (dla Micro Focus COBOL) *MQ_INSTALLATION_PATH*\Tools\cobol\copybook\VAcobol (dla IBM VisualAge COBOL) |
| z/OS z/OS | **thlqual**.SCSQCOBC, |

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

W programie należy uwzględnić tylko te pliki, które są potrzebne. Można to zrobić za pomocą jednej lub większej liczby instrukcji COPY po deklaracji level-01 . Oznacza to, że w razie potrzeby można dołączyć do programu wiele wersji struktur. Należy zauważyć, że CMQV jest dużym plikiem.

Poniżej przedstawiono przykład kodu COBOL włączającego plik kopii CMQMDV:

```
01 MQM-MESSAGE-DESCRIPTOR.
COPY CMQMDV.
```

Każda deklaracja struktury rozpoczyna się od elementu level-01 ; można zadeklarować kilka instancji struktury, kodując deklarację level-01 , a następnie instrukcję COPY w celu skopiowania pozostałej części deklaracji struktury. Aby odwołać się do odpowiedniej instancji, należy użyć słowa kluczowego IN.

Poniżej przedstawiono przykład kodu COBOL uwzględniającego dwie instancje CMQMDV:

```
* Declare two instances of MQMD
01 MY-CMQMD.
COPY CMQMDV.
01 MY-OTHER-CMQMD.
COPY CMQMDV.
*
* Set MSGTYPE field in MY-OTHER-CMQMD
MOVE MQMT-REQUEST TO MQMD-MSGTYPE IN MY-OTHER-CMQMD.
```

Wyrównaj struktury na granicach 4-bajtowych. Jeśli instrukcja COPY jest używana w celu uwzględnienia struktury po elemencie, który nie jest elementem level-01 , upewnij się, że struktura jest wielokrotnością 4 bajtów od początku elementu level-01 . Jeśli tego nie zrobisz, możesz zmniejszyć wydajność aplikacji.

Struktury są opisane w sekcji Typy danych używane w MQI. Opisy pól w strukturach zawierają nazwy pól bez przedrostka. W programach w języku COBOL należy poprzedzić nazwy pól nazwą struktury, po której następuje łącznik, tak jak to pokazano w deklaracjach języka COBOL. Pola w plikach kopii struktury są w ten sposób poprzedzone przedrostkiem.

Nazwy pól w deklaracjach w plikach kopii struktury są zapisane wielkimi literami. Zamiast tego można używać małych i wielkich liter. Na przykład pole *StrucId* struktury MQGMO jest wyświetlane jako MQGMO-STRUCID w deklaracji COBOL i w pliku kopii.

Struktury przyrostka V są deklarowane z wartościami początkowymi dla wszystkich pól, dlatego należy ustawić tylko te pola, w których wymagana wartość różni się od wartości początkowej.

### Makra w języku asembler System/390

z/OS

IBM MQ for z/OS udostępnia dwa makra w języku asemblera zawierające stałe nazwane oraz jedno makro do generowania każdej struktury.

Są one wymienione w sekcji z/OS Assembler COPY files i instalowane w pliku **thlqual**.SCSQMACS.

Te makra są wywoływane przy użyciu kodu podobnego do następującego:

```
MY_MQMD CMQMDA EXPIRY=0,MSGTYPE=MQMT_DATAGRAM
```

### Pliki włączane języka PL/I

z/OS

IBM MQ for z/OS udostępnia pliki włączane, które zawierają wszystkie definicje potrzebne podczas pisania aplikacji IBM MQ w języku PL/I.

Pliki są wymienione w plikach włączanych języka PL/I i instalowane w katalogu **thlqual**.SCSQPLIC:

Te pliki należy dołączyć do programu, jeśli planowane jest dowiązanie kodu pośredniczącego IBM MQ do programu (patrz sekcja “Przygotowywanie programu do uruchomienia” na stronie 1053 ). Jeśli planowane jest dynamiczne łączenie wywołań IBM MQ , należy uwzględnić tylko CMQP (patrz sekcja “Dynamiczne wywoływanie kodu pośredniczącego IBM MQ” na stronie 1060 ). Konsolidacja dynamiczna może być wykonywana tylko dla programów wsadowych i IMS .

## Pisanie aplikacji proceduralnej dotyczącej kolejkowania

Ten temat zawiera informacje o pisaniu aplikacji kolejkowania, nawiązywaniu i rozłączaniu połączeń z menedżerem kolejek, publikowaniu i subskrybowaniu oraz otwieraniu i zamykaniu obiektów.

Aby dowiedzieć się więcej o pisaniu aplikacji, skorzystaj z następujących odsyłaczy:

- “Przegląd interfejsu kolejki komunikatów” na stronie 745
- “Nawiązywanie i rozłączanie połączenia z menedżerem kolejek” na stronie 758
- “Otwieranie i zamykanie obiektów” na stronie 766
- “Umieszczanie komunikatów w kolejce” na stronie 777
- “Pobieranie komunikatów z kolejki” na stronie 792
- “Pisanie aplikacji publikowania/subskrybowania” na stronie 834
- “Uzyskiwanie informacji o atrybutach obiektu i ustawianie ich” na stronie 876
- “Zatwierdzanie i wycofywanie jednostek pracy” na stronie 879
- “Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy” na stronie 891
- “Praca z funkcją MQI i klastrami” na stronie 911
- z/OS “Używanie i pisanie aplikacji w systemie IBM MQ for z/OS” na stronie 916
- z/OS “Aplikacje mostu IMS i IMS w systemie IBM MQ for z/OS” na stronie 71

**Pojęcia pokrewne**

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć języka proceduralnego lub obiektowego. Przed rozpoczęciem projektowania i pisania aplikacji IBM MQ należy zapoznać się z podstawowymi pojęciami związanymi z produktem IBM MQ .

“Tworzenie aplikacji dla składnika IBM MQ” na stronie 5
Użytkownik może tworzyć aplikacje do wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i zasobami pokrewnymi. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i środowiskach.

“Uwagi dotyczące projektowania dla aplikacji IBM MQ” na stronie 51
Po podjęciu decyzji, w jaki sposób aplikacje mogą korzystać z dostępnych platform i środowisk, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

“Pisanie aplikacji proceduralnych klienta” na stronie 941
Informacje potrzebne do pisania aplikacji klienckich w systemie IBM MQ przy użyciu języka proceduralnego.

“Budowanie aplikacji proceduralnej” na stronie 1029
Użytkownik może napisać aplikację IBM MQ w jednym z kilku języków proceduralnych i uruchomić ją na kilku różnych platformach.

“Obsługa błędów proceduralnych programu” na stronie 1069
Te informacje wyjaśniają błędy powiązane z wywołaniami MQI aplikacji podczas wykonywania wywołania lub po dostarczeniu komunikatu do miejsca docelowego.

**Zadania pokrewne**

“Korzystanie z przykładowych programów proceduralnych IBM MQ” na stronie 1089
Te przykładowe programy zostały napisane w językach proceduralnych i przedstawiają typowe zastosowania interfejsu kolejek komunikatów (Message Queue Interface-MQI). Programy IBM MQ na różnych platformach.

## Przegląd interfejsu kolejki komunikatów

Sekcja zawiera informacje na temat komponentów interfejsu kolejki komunikatów (Message Queue Interface-MQI).

Interfejs kolejki komunikatów składa się z następujących elementów:

- *Wywołania* , za pośrednictwem których programy mogą uzyskiwać dostęp do menedżera kolejek i jego narzędzi.
- *Struktury* używane przez programy do przekazywania danych do menedżera kolejek i pobierania danych z menedżera kolejek.
- *Elementarne typy danych* na potrzeby przekazywania danych do menedżera kolejek i pobierania danych z menedżera kolejek.

z/OS IBM MQ for z/OS dostarcza również:

- Dwa dodatkowe wywołania, za pomocą których programy wsadowe z/OS mogą zatwierdzać i wycofywać zmiany.
- *Pliki definicji danych* (czasami nazywane kopiowanymi plikami, makrami, plikami włączanymi i nagłówkami), które definiują wartości stałych dostarczanych z produktem IBM MQ for z/OS.
- *Programy pośredniczące* , aby połączyć je z aplikacjami.
- Pakiet przykładowych programów demonstrujących sposób użycia interfejsu MQI na platformie z/OS . Więcej informacji na temat tych przykładów zawiera sekcja “Korzystanie z przykładowych programów w systemie z/OS” na stronie 1194.

IBM i IBM MQ for IBM i dostarcza również:

- *Pliki definicji danych* (czasami nazywane kopiowanymi plikami, makrami, plikami włączanymi i nagłówkami), które definiują wartości stałych dostarczanych z produktem IBM MQ for IBM i.
- Trzy programy pośredniczące do konsolidacji z aplikacjami ILE C, ILE COBOL i ILE RPG.
- Pakiet przykładowych programów demonstrujących sposób użycia interfejsu MQI na platformie IBM i .

Systemy AIX, Linux, and Windows również dostarczają:

- Wywołania, za pośrednictwem których programy systemowe IBM MQ for AIX, Linux, and Windows mogą zatwierdzać i wycofywać zmiany.
- *Pliki włączane* definiujące wartości stałych dostarczanych na tych platformach.
- *Pliki biblioteki* , aby dowiązać aplikacje.

- Pakiet przykładowych programów demonstrujących sposób użycia interfejsu MQI na tych platformach. Więcej informacji na temat tych przykładów zawiera sekcja “Korzystanie z przykładowych programów w systemie Multiplatforms” na stronie 1090.
- Przykładowy kod źródłowy i wykonywalny dla powiązań z zewnętrznymi menedżerami transakcji.

Więcej informacji na temat interfejsu MQI można znaleźć, korzystając z następujących odsyłaczy:


- “Wywołania MQI” na stronie 746
- “Wywołania punktu synchronizacji” na stronie 747
- “Konwersja danych, typy danych, definicje danych i struktury” na stronie 748
- “Programy pośredniczące IBM MQ i pliki bibliotek” na stronie 749
- “Parametry wspólne dla wszystkich wywołań” na stronie 754
- “Określanie buforów” na stronie 755
- z/OS “Uwagi dotyczące zadań wsadowych dla systemu z/OS” na stronie 755
- “Obsługa sygnału AIX and Linux” na stronie 755


**Pojęcia pokrewne**

“Nawiązywanie i rozłączanie połączenia z menedżerem kolejek” na stronie 758
Aby można było używać usług programistycznych IBM MQ , program musi mieć połączenie z menedżerem kolejek. Ten temat zawiera informacje o nawiązywaniu połączenia z menedżerem kolejek i rozłączaniu się z nim.

“Otwieranie i zamykanie obiektów” na stronie 766
Informacje te udostępniają wgląd w otwieranie i zamykanie obiektów IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 777
Ta sekcja zawiera informacje na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 792
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Uzyskiwanie informacji o atrybutach obiektu i ustawianie ich” na stronie 876
Atrybuty są właściwościami definiującymi charakterystykę obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 879
W tej sekcji opisano sposób zatwierdzania i wycofywania wszelkich odtwarzalnych operacji pobierania i umieszczania, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy” na stronie 891
Informacje o wyzwalaczach i sposobie uruchamiania aplikacji IBM MQ za pomocą wyzwalaczy.

“Praca z funkcją MQI i klastrami” na stronie 911
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do łączenia w klastry.

“Używanie i pisanie aplikacji w systemie IBM MQ for z/OS” na stronie 916
Aplikacje IBM MQ for z/OS mogą być tworzone z programów działających w wielu różnych środowiskach. Oznacza to, że mogą korzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje mostu IMS i IMS w systemie IBM MQ for z/OS” na stronie 71
Informacje te są pomocne podczas pisania aplikacji IMS przy użyciu produktu IBM MQ.

## *Wywołania MQI*

Ten temat zawiera informacje o wywołaniach interfejsu Message Queue Interface (MQI).

Wywołania interfejsu MQI można pogrupować w następujący sposób:

**MQCONN, MQCONNX i MQDISC**
: Za pomocą tych wywołań można połączyć program z menedżerem kolejek (z opcjami lub bez opcji) i odłączyć program od menedżera kolejek. Jeśli piszesz programy CICS dla z/OS, nie musisz używać tych wywołań. Zaleca się jednak ich używanie, jeśli aplikacja ma być używana na innych platformach.

**MQOPEN i MQCLOSE**
: Te wywołania służą do otwierania i zamykania obiektu, takiego jak kolejka.

**MQPUT i MQPUT1**
Te wywołania służą do umieszczania komunikatu w kolejce.

**MQGET**
To wywołanie służy do przeglądania komunikatów w kolejce lub do usuwania komunikatów z kolejki.

**MQSUB, MQSUBRQ**
Te wywołania służą do rejestrowania subskrypcji w temacie i żądania publikacji zgodnych z subskrypcją.

**MQINQ**
To wywołanie umożliwia uzyskanie informacji o atrybutach obiektu.

**MQSET**
To wywołanie służy do ustawiania niektórych atrybutów kolejki. Nie można ustawić atrybutów innych typów obiektów.

**MQBEGIN, MQCMIT i MQBACK**
Użyj tych wywołań, gdy IBM MQ jest koordynatorem jednostki pracy. Komenda MQBEGIN uruchamia jednostkę pracy. Komendy MQCMIT i MQBACK kończą jednostkę pracy, zatwierdzając lub wycofując aktualizacje wprowadzone podczas tej jednostki pracy. IBM i IBM i Kontroler zatwierdzania jest używany do koordynowania globalnych jednostek pracy w systemie IBM MQ for IBM i. Używane są rodzime komendy kontroli transakcji, zatwierdzania i wycofywania zmian.

**MQCRTMH, MQBUFMH, MQMHBUF, MQDLTMH**
Za pomocą tych wywołań można utworzyć uchwyt komunikatu, przekształcić uchwyt komunikatu w bufor lub bufor w uchwyt komunikatu oraz usunąć uchwyt komunikatu.

**MQSETMP, MQINQMP, MQDLTMP**
Te wywołania służą do ustawiania właściwości komunikatu w uchwycie komunikatu, sprawdzania właściwości komunikatu i usuwania właściwości z uchwytu komunikatu.

**MQCB, MQCB_FUNCTION, MQCTL**
Te wywołania służą do rejestrowania i sterowania funkcją zwrotną.

**MQSTAT**
To wywołanie służy do pobierania informacji o statusie poprzednich asynchronicznych operacji umieszczania.

Opis wywołań MQI zawiera sekcja Opisy wywołań .

### *Wywołania punktu synchronizacji*
Ten temat zawiera informacje o wywołaniach punktów synchronizacji na różnych platformach.

Wywołania punktu synchronizacji są dostępne w następujący sposób:

## IBM MQ for z/OS wywołania
z/OS

Produkt IBM MQ for z/OS udostępnia wywołania MQCMIT i MQBACK.

Te wywołania w programach wsadowych systemu z/OS służą do poinformowania menedżera kolejek, że wszystkie operacje MQGET i MQPUT od ostatniego punktu synchronizacji mają zostać trwale (zatwierdzone) lub mają zostać wycofane. Aby zatwierdzić i wycofać zmiany w innych środowiskach:

**CICS**
Użyj komend, takich jak EXEC CICS SYNCPOINT i EXEC CICS SYNCPOINT ROLLBACK.

**IMS**
Użyj narzędzi punktu synchronizacji systemu IMS , takich jak GU (get unique) dla wywołań IOPCB, CHKP (checkpoint) i ROLB (rollback).

**RRS**
Użyj odpowiednio MQCMIT i MQBACK lub SRRCMIT i SRRBACK. (Patrz “Usługi zarządzania transakcjami i odtwarzalnego menedżera zasobów” na stronie 884).

**Uwaga:** SRRCMIT i SRRBACK to rodzime komendy RRS, nie są to wywołania MQI.

## IBM i wywołania

IBM i

IBM MQ for IBM i udostępnia komendy MQCMIT i MQBACK. Można również użyć komend COMMIT i ROLLBACK systemu IBM i lub dowolnych innych komend lub wywołań, które inicjują narzędzia kontroli transakcji IBM i (na przykład EXEC CICS SYNCPOINT).

## Wywołania IBM MQ na platformach AIX, Linux, and Windows

ALW

IBM MQ for AIX, Linux, and Windows udostępnia wywołania MQCMIT i MQBACK.

Wywołania punktu synchronizacji w programach służą do poinformowania menedżera kolejek, że wszystkie operacje MQGET i MQPUT od czasu ostatniego punktu synchronizacji mają być trwałe (zatwierdzone) lub mają zostać wycofane. Aby zatwierdzić i wycofać zmiany w środowisku CICS , należy użyć komend, takich jak EXEC CICS SYNCPOINT i EXEC CICS SYNCPOINT ROLLBACK.

## *Konwersja danych, typy danych, definicje danych i struktury*

Te informacje umożliwiają zapoznanie się z konwersjami danych, podstawowymi typami danych, definicjami danych IBM MQ i strukturami podczas korzystania z interfejsu kolejki komunikatów.

### Konwersja danych

Wywołanie MQXCNVC (konwersja znaków) przekształca dane znakowe komunikatu z jednego zestawu znaków na inny. Z wyjątkiem systemu IBM MQ for z/OS, to wywołanie jest używane tylko z wyjścia konwersji danych.

Sekcja MQXCNVC-Convert characters zawiera informacje o składni wywołania MQXCNVC oraz “Zapisywanie wyjść konwersji danych” na stronie 1013 zawiera wskazówki dotyczące zapisywania i wywoływania wyjść konwersji danych.

### Podstawowe typy danych

W przypadku obsługiwanych języków programowania MQI udostępnia podstawowe typy danych lub pola nieustrukturyzowane.

Te typy danych są w pełni opisane w sekcji Elementarne typy danych.

### IBM MQ Definicje danych

z/OS IBM MQ for z/OS udostępnia definicje danych w postaci plików kopii w języku COBOL, makr w języku asemblacji, pojedynczego pliku włączanego w języku PL/I, pojedynczego pliku włączanego w języku C oraz plików włączanych w języku C + +.

IBM i IBM MQ for IBM i dostarcza definicje danych w postaci plików kopii COBOL, plików kopii RPG, plików włączanych w języku C oraz plików włączanych w języku C + +.

Pliki definicji danych dostarczane z produktem IBM MQ zawierają:

- Definicje wszystkich stałych i kodów powrotu IBM MQ
- Definicje struktur i typów danych IBM MQ
- Definicje stałych do inicjowania struktur
- Prototypy funkcji dla każdego wywołania (tylko dla języka PL/I i języka C)

Pełny opis plików definicji danych systemu IBM MQ zawiera sekcja “Pliki definicji danych IBM MQ” na stronie 741.

### Struktury

Struktury używane z wywołaniami MQI wymienionymi w sekcji “Wywołania MQI” na stronie 746są dostarczane w plikach definicji danych dla każdego obsługiwanego języka programowania.

IBM i z/OS IBM MQ for z/OS i IBM MQ for IBM i dostarczają pliki zawierające stałe, których można użyć podczas wypełniania niektórych pól tych struktur. Więcej informacji na ten temat zawiera sekcja IBM MQ -definicje danych.

Podsumowanie struktur zawiera sekcja Typy danych struktury .

## *Programy pośredniczące IBM MQ i pliki bibliotek*

W tym miejscu wymieniono udostępnione programy pośredniczące i pliki bibliotek dla każdej platformy.

Więcej informacji na temat używania programów pośredniczących i plików bibliotek podczas budowania aplikacji wykonywalnej zawiera sekcja “Budowanie aplikacji proceduralnej” na stronie 1029. Informacje na temat dowiązywania do plików bibliotek C++ zawiera sekcja Korzystanie z języka C++ *IBM MQ Korzystanie z języka C++*.

### AIX *IBM MQ for AIX pliki biblioteki*

W systemie IBM MQ for AIXnależy połączyć program z plikami biblioteki MQI dostarczonymi dla środowiska, w którym działa aplikacja, oprócz plików udostępnianych przez system operacyjny.

W aplikacji niewielowątkowej należy utworzyć odsyłacz do jednej z następujących bibliotek:

*Tabela 106. Pliki bibliotek dla aplikacji AIX , które nie są wielowątkowe*

| **Plik biblioteki** | **Środowisko** |
|---|---|
| libmqm.a | Serwer dla języka C |
| libmqic.a i libmqm.a | Klient dla języka C |
| libmqmzf.a | Możliwe do zainstalowania wyjścia serwisowe dla języka C |
| libmqmxa.a | Interfejs XA serwera |
| libmqmxa64.a | Alternatywny interfejs XA serwera |
| libmqcxa.a | Interfejs XA klienta |
| libmqcxa64.a | Alternatywny interfejs XA klienta |
| libmqmcbrt.o | Biblioteka środowiska wykonawczego IBM MQ do obsługi języka Micro Focus COBOL |
| libmqmcb.a | Serwer dla języka COBOL |
| libmqicb.a | Klient dla języka COBOL |
| libimqc23ia.a | Klient dla C++ (XLC 16) |
| libimqs23ia.a | Serwer dla języka C++ (XLC 16) |
| V 9.3.5 libimqc23ca.a | Klient dla C++ (XLC 17) |
| V 9.3.5 libimqs23ca.a | Serwer dla C++ (XLC 17) |

V 9.3.5 Biblioteki zawierające "ia" zostały zbudowane za pomocą kompilatora XLC 16, a biblioteki z "ca" w nazwie zostały zbudowane za pomocą kompilatora XLC 17.

W aplikacji wielowątkowej należy utworzyć odsyłacz do jednej z następujących bibliotek:

| *Tabela 107. Pliki bibliotek dla aplikacji AIX z wątkami.*<br>Dwukolumnowa tabela zawierająca listę plików bibliotek i środowiska dla każdego pliku biblioteki. | |
|---|---|
| **Plik biblioteki** | **Środowisko** |
| libmqm_r.a | Serwer dla języka C |
| libmqic_r.a i libmqm_r.a | Klient dla języka C |
| libmqmzf_r.a | Możliwe do zainstalowania wyjścia serwisowe dla języka C |
| libmqmxa_r.a | Interfejs XA serwera |
| libmqmxa64_r.a | Alternatywny interfejs XA serwera |
| libmqcxa_r.a | Interfejs XA klienta |
| libmqcxa64_r.a | Alternatywny interfejs XA klienta |
| libimqc23ia_r.a | Klient dla C++ (XLC 16) |
| libimqs23ia_r.a | Serwer dla języka C++ (XLC 16) |
| V 9.3.5 libimqc23ca_r.a | Klient dla C++ (XLC 17) |
| V 9.3.5 libimqs23ca_r.a | Serwer dla C++ (XLC 17) |

V 9.3.5 Biblioteki zawierające "ia" zostały zbudowane za pomocą kompilatora XLC 16, a biblioteki z "ca" w nazwie zostały zbudowane za pomocą kompilatora XLC 17.

**Uwaga:** Nie można utworzyć odsyłacza do więcej niż jednej biblioteki. Oznacza to, że nie można jednocześnie utworzyć dowiązania do biblioteki wielowątkowej i niewielowątkowej.

IBM i *IBM MQ for IBM i pliki biblioteki*

W produkcie IBM MQ for IBM inależy połączyć program z plikami biblioteki MQI dostarczonymi dla środowiska, w którym działa aplikacja, oprócz plików udostępnianych przez system operacyjny.

W przypadku aplikacji niewielowątkowych:

| *Tabela 108. Pliki bibliotek dla aplikacji IBM i , które nie są wielowątkowe* | |
|---|---|
| **Plik biblioteki** | **Środowisko** |
| LIBMQM, | Program usługowy serwera i klienta |
| LIBMQICQC | Program usługowy klienta |
| IMQB23I4 | Podstawowy program usługowy C++ |
| IMQS23I4 | Program usługowy serwera C++ |
| LIBMQMZF, | Możliwe do zainstalowania wyjścia dla języka C |

W aplikacji wielowątkowej:

| *Tabela 109. Pliki bibliotek dla aplikacji IBM i z wątkami* | |
|---|---|
| **Plik biblioteki** | **Środowisko** |
| **LIBMQM_R** | Program usługowy serwera i klienta |
| **IMQB23I4_R** | Podstawowy program usługowy C++ |
| **IMQS23I4_R** | Program usługowy serwera C++ |

| *Tabela 109. Pliki bibliotek dla aplikacji IBM i z wątkami (kontynuacja)* | |
|---|---|
| **Plik biblioteki** | **Środowisko** |
| **LIBMQMZF_R** | Możliwe do zainstalowania wyjścia dla języka C |
| **LIBMQIC_R** | Program usługowy klienta |

W systemie IBM MQ for IBM imożna pisać aplikacje w języku C + +. Aby zapoznać się ze sposobem łączenia aplikacji C++ i uzyskać szczegółowe informacje na temat wszystkich aspektów używania języka C + +, należy zapoznać się z sekcją Korzystanie z języka C++.

Linux *Pliki biblioteki produktu IBM MQ for Linux*

W systemie IBM MQ for Linuxoprócz plików udostępnianych przez system operacyjny należy połączyć program z plikami biblioteki MQI dostarczonymi dla środowiska, w którym działa aplikacja.

W aplikacji niewielowątkowej należy utworzyć odsyłacz do jednej z następujących bibliotek:

| *Tabela 110. Pliki bibliotek dla aplikacji Linux , które nie są wielowątkowe* | |
|---|---|
| **Plik biblioteki** | **Środowisko** |
| libmqm.so | Serwer dla języka C |
| libmqic.so i libmqm.so | Klient dla języka C |
| libmqmzf.so | Możliwe do zainstalowania wyjścia serwisowe dla języka C |
| libmqmxa.so | Interfejs XA serwera |
| libmqmxa64.so | Alternatywny interfejs XA serwera |
| libmqcxa.so | Interfejs XA klienta |
| libmqcxa64.so | Alternatywny interfejs XA klienta |
| libimqc23gl.so | Klient dla języka C++ |
| libimqs23gl.so | Serwer dla języka C++ |

W aplikacji wielowątkowej należy utworzyć odsyłacz do jednej z następujących bibliotek:

| *Tabela 111. Pliki bibliotek dla aplikacji Linux z wątkami* | |
|---|---|
| **Plik biblioteki** | **Środowisko** |
| libmqm_r.so | Serwer dla języka C |
| libmqic_r.so i libmqm_r.so | Klient dla języka C |
| libmqmzf_r.so | Możliwe do zainstalowania wyjścia serwisowe dla języka C |
| libmqmxa_r.so | Interfejs XA serwera |
| libmqmxa64_r.so | Alternatywny interfejs XA serwera |
| libmqcxa_r.so | Interfejs XA klienta |
| libmqcxa64_r.so | Alternatywny interfejs XA klienta |
| libimqc23gl_r.so | Klient dla języka C++ |
| libimqs23gl_r.so | Serwer dla języka C++ |

**Uwaga:** Nie można utworzyć odsyłacza do więcej niż jednej biblioteki. Oznacza to, że nie można jednocześnie utworzyć dowiązania do biblioteki wielowątkowej i niewielowątkowej.

Windows *IBM MQ for Windows pliki biblioteki*

W systemie IBM MQ for Windowsoprócz plików udostępnianych przez system operacyjny należy połączyć program z plikami biblioteki MQI dostarczonymi dla środowiska, w którym działa aplikacja:

*Tabela 112. Pliki bibliotek dla aplikacji Windows*

| **Plik biblioteki** | **Środowisko** |
|---|---|
| *MQ_INSTALLATION_PATH*\Tools\Lib\mqm.lib | Server for C (32-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\mqic.lib | Client for C (32-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\mqmxa.lib | Server XA interface for C (32-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\mqcxa.lib | Interfejs XA klienta dla języka C (32-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\mqicxa.lib | Client MTS for C (32-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\mqmcics4.lib32 | Obsługa serwera TXSeries CICS dla języka C (32-bitowego) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\mqccics4.lib32 | Obsługa klienta TXSeries CICS dla języka C (32-bitowego) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\mqmzf.lib | Wyjścia instalowalnych usług dla języka C (32-bitowego) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\mqmcbb.lib | Server for IBM COBOL (32-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\mqmcb.lib | Server for Micro Focus COBOL (32-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\mqiccbb.lib | Client for IBM COBOL (32-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\mqiccb.lib | Client for Micro Focus COBOL (32-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\imqs23vn.lib | Server for C++ (32-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\imqc23vn.lib | Client for C++ (32-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\imqb23vn.lib | Base for C++ (wersja 32-bitowa) |
| *MQ_INSTALLATION_PATH*\Tools\Lib\imqx23vn.lib | Client MTS for C++ (32-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib64\mqm.lib | Serwer dla C (64-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib64\mqic.lib | Client for C (64-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib64\mqmxa.lib | Interfejs XA serwera dla języka C (64-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib64\mqcxa.lib | Client XA interface for C (64-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib64\mqicxa.lib | Client MTS for C (64-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib64\mqmcbb.lib | Server for IBM COBOL (64-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib64\mqmcb.lib | Serwer dla Micro Focus COBOL (64-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib64\mqiccbb.lib | Client for IBM COBOL (64-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib64\mqiccb.lib | Client for Micro Focus COBOL (64-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib64\imqs23vn.lib | Server for C++ (64-bitowy) |
| *MQ_INSTALLATION_PATH*\Tools\Lib64\imqc23vn.lib | Client for C++ (64-bitowy) |

| *Tabela 112. Pliki bibliotek dla aplikacji Windows (kontynuacja)* | |
|---|---|
| **Plik biblioteki** | **Środowisko** |
| *MQ_INSTALLATION_PATH*\Tools\Lib64\imqb23vn.lib | Base for C++ (wersja 64-bitowa) |
| *MQ_INSTALLATION_PATH*\Tools\Lib64\imqx23vn.lib | Client MTS for C++ (64-bitowy) |

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Do kompilowania programów .NET należy użyć parametru `amqmdnet.dll` . Więcej informacji na ten temat zawiera sekcja "Kompilowanie programów IBM MQ .NET" na stronie 631 w sekcji "Tworzenie aplikacji .NET" na stronie 573 .

Te pliki są dostarczane w celu zapewnienia zgodności z poprzednimi wersjami:

```
mqic32.lib
mqic32xa.lib
```

z/OS *IBM MQ for z/OS programy pośredniczące*

Przed uruchomieniem programu napisanego przy użyciu kodu IBM MQ for z/OSnależy go dowiązać do kodu pośredniczącego programu dostarczonego z produktem IBM MQ for z/OS dla środowiska, w którym uruchamiana jest aplikacja.

Program pośredniczący udostępnia pierwszy etap przetwarzania wywołań w żądania, które może przetworzyć produkt IBM MQ for z/OS .

Produkt IBM MQ for z/OS udostępnia następujące programy pośredniczące:

**CSQBSTUB**
Program pośredniczący dla programów wsadowych systemu z/OS

**CSQBRRSI**
Program pośredniczący dla programów wsadowych systemu z/OS używających usługi RRS za pośrednictwem interfejsu MQI

**CSQBRSTB**
Program pośredniczący dla programów wsadowych systemu z/OS używających bezpośrednio usługi RRS

**CSQCSTUB**
Program pośredniczący dla programów CICS

**CSQQSTUB**
Program pośredniczący dla programów IMS

**CSQXSTUB**
Program pośredniczący dla rozproszonego kolejkowania wyjść innych niżCICS

**KSQASTUB**
Program pośredniczący dla wyjść konwersji danych

**Ostrzeżenie:** Jeśli dla konkretnego środowiska używany jest program pośredniczący inny niż wymieniony, może to mieć nieprzewidywalne skutki.

**Uwaga:** Jeśli używany jest kod pośredniczący CSQBRSTB, należy dokonać konsolidacji z ATRSCSS z systemu SYS1.CSSLIB. (SYS1.CSSLIB jest również nazywana *biblioteką usług wywoływalnych* ). Więcej informacji na temat usługi RRS zawiera sekcja "Usługi zarządzania transakcjami i odtwarzalnego menedżera zasobów" na stronie 884.

Alternatywnie można dynamicznie wywołać kod pośredniczący z poziomu programu. Ta technika jest opisana w sekcji "Dynamiczne wywoływanie kodu pośredniczącego IBM MQ" na stronie 1060.

W systemie IMSmoże być również konieczne użycie specjalnego modułu interfejsu języka, który jest dostarczany z produktem IBM MQ.

Nie należy uruchamiać aplikacji, które są edytowane za pomocą odsyłaczy z CSQBSTUB i CSQQSTUB w tym samym regionie MPP IMS . Może to powodować problemy, takie jak komunikaty DFS3607I lub CSQQ005E . Pierwsze wywołanie MQCONN w przestrzeni adresowej określa, który interfejs jest używany, dlatego transakcje CSQQSTUB i CSQBSTUB muszą działać w różnych regionach komunikatów IMS .

### *Parametry wspólne dla wszystkich wywołań*

Istnieją dwa typy parametrów wspólnych dla wszystkich wywołań: uchwyty i kody powrotu.

## Korzystanie z uchwytów

Wszystkie wywołania MQI używają jednego lub kilku *uchwytów*. Identyfikują one menedżera kolejek, kolejkę lub inny obiekt, komunikat lub subskrypcję odpowiednio do wywołania.

Aby program mógł komunikować się z menedżerem kolejek, musi mieć unikalny identyfikator, na podstawie którego zna ten menedżer kolejek. Ten identyfikator jest nazywany *uchwytem połączenia*, czasem nazywany *Hconn*. W przypadku programów CICS uchwyt połączenia zawsze ma wartość zero. W przypadku wszystkich innych platform lub stylów programów uchwyt połączenia jest zwracany przez wywołanie MQCONN lub MQCONNX, gdy program łączy się z menedżerem kolejek. Programy przekazują uchwyt połączenia jako parametr wejściowy, gdy używają innych wywołań.

Aby program mógł pracować z obiektem IBM MQ , musi mieć unikalny identyfikator, za pomocą którego rozpoznaje ten obiekt. Ten identyfikator jest nazywany *uchwytem obiektu*, czasem nazywany *Hobj*. Uchwyt jest zwracany przez wywołanie MQOPEN, gdy program otwiera obiekt do pracy z nim. Programy przekazują uchwyt obiektu jako parametr wejściowy, gdy używają kolejnych wywołań MQPUT, MQGET, MQINQ, MQSET lub MQCLOSE.

Podobnie wywołanie MQSUB zwraca *uchwyt subskrypcji* lub *Hsub*, który jest używany do identyfikowania subskrypcji w kolejnych wywołaniach MQGET, MQCB lub MQSUBRQ, a niektóre wywołania przetwarzające właściwości komunikatu używają *uchwytu komunikatu* lub *Hmsg*.

## Kody powrotu-podstawy

Kod zakończenia i kod przyczyny są zwracane jako parametry wyjściowe przez każde wywołanie. Są one określane zbiorczo jako *kody powrotu*.

Aby pokazać, czy wywołanie zakończyło się pomyślnie, każde wywołanie zwraca *kod zakończenia* po zakończeniu wywołania. Kod zakończenia jest zwykle kodem MQCC_OK wskazującym powodzenie lub kodem MQCC_FAILED wskazującym niepowodzenie. Niektóre wywołania mogą zwracać stan pośredni (MQCC_WARNING) wskazujący częściowy sukces.

Każde wywołanie zwraca również *kod przyczyny* , który przedstawia przyczynę niepowodzenia lub częściowego powodzenia wywołania. Istnieje wiele kodów przyczyny, obejmujących takie okoliczności, jak zapełnienie kolejki, niedozwolone operacje pobierania dla kolejki oraz brak definicji konkretnej kolejki dla menedżera kolejek. Programy mogą korzystać z kodu przyczyny, aby zdecydować, w jaki sposób kontynuować. Na przykład mogą oni poprosić użytkowników o zmianę danych wejściowych, a następnie ponownie wykonać wywołanie lub mogą zwrócić komunikat o błędzie do użytkownika.

Jeśli kodem zakończenia jest MQCC_OK, kodem przyczyny jest zawsze MQRC_NONE.

Kody zakończenia i przyczyny dla każdego wywołania są wymienione z opisem tego wywołania. Patrz sekcja Opisy połączeń i wybierz odpowiednie połączenie z listy.

Bardziej szczegółowe informacje, w tym pomysły na działania naprawcze, znajdują się w następujących sekcjach:

- z/OS Komunikaty systemu IBM MQ for z/OS , kody zakończenia i kody przyczyny dla IBM MQ for z/OS
- Komunikaty i kody przyczyny dla wszystkich innych platform IBM MQ

### *Określanie buforów*

Menedżer kolejek odwołuje się do buforów tylko wtedy, gdy są one wymagane. Jeśli w wywołaniu nie jest wymagany bufor lub bufor ma zerową długość, można użyć wskaźnika pustego do buforu.

Podczas określania wymaganej wielkości buforu należy zawsze używać długości danych.

Jeśli do przechowywania danych wyjściowych wywołania używany jest bufor (na przykład do przechowywania danych komunikatu dla wywołania MQGET lub wartości atrybutów, których dotyczy wywołanie MQINQ), menedżer kolejek próbuje zwrócić kod przyczyny, jeśli podany bufor jest niepoprawny lub znajduje się w pamięci tylko do odczytu. Może jednak nie zawsze być w stanie zwrócić kod przyczyny.

## z/OS *Uwagi dotyczące zadań wsadowych dla systemu z/OS*

Programy wsadowe z/OS , które wywołują MQI, mogą być w stanie nadzorującym lub problemowym.

Muszą one jednak spełniać następujące warunki:

- Muszą one być w trybie zadania, a nie w trybie bloku zgłoszenia serwisowego (SRB).
- Muszą być w trybie ASC (Primary address space control) (a nie Access Register ASC).
- Nie mogą one być w trybie międzypamięciowym. Podstawowy numer przestrzeni adresowej (ASN) musi być taki sam jak dodatkowy numer ASN i podstawowy numer ASN.
- Nie mogą być one używane jako programy obsługi wyjścia MPF.
- Nie można wstrzymywać żadnych blokad systemu z/OS .
- W stosie FRR nie mogą być dostępne żadne procedury odtwarzania funkcji (FRR).
- Dla wywołania MQCONN lub MQCONNX może obowiązywać dowolny klucz PSW (PSW) (pod warunkiem, że klucz jest zgodny z użyciem pamięci masowej, która znajduje się w kluczu TCB), ale kolejne wywołania używające uchwytu połączenia zwróconego przez MQCONN lub MQCONNX:
  - Musi mieć ten sam klucz PSW, który został użyty w wywołaniu MQCONN lub MQCONNX
  - Parametry muszą być dostępne (w razie potrzeby dla zapisu) w tym samym kluczu PSW
  - Musi być wystawiona w ramach tego samego zadania (TCB), ale nie w żadnym podzadaniu zadania
- Mogą one być w trybie adresowania 24-bitowego lub 31-bitowego. Jeśli jednak obowiązuje 24-bitowy tryb adresowania, adresy parametrów muszą być interpretowane jako poprawne adresy 31-bitowe.

Jeśli którykolwiek z tych warunków nie jest spełniony, może zostać wykonane sprawdzenie programu. W niektórych przypadkach wywołanie nie powiedzie się i zostanie zwrócony kod przyczyny.

## Linux AIX *Uwagi dotyczące produktu AIX and Linux*

Zagadnienia, o których należy pamiętać podczas tworzenia aplikacji AIX and Linux .

### Linux AIX *Wywołanie systemowe fork w systemach AIX and Linux*

Należy zapoznać się z tymi uwagami podczas używania wywołania systemowego fork w aplikacjach IBM MQ .

Jeśli aplikacja ma używać języka `fork`, proces nadrzędny tej aplikacji powinien wywołać funkcję `fork` przed wykonaniem jakichkolwiek wywołań IBM MQ , na przykład `MQCONN`lub utworzeniem obiektu IBM MQ przy użyciu języka **`ImqQueueManager`**.

Jeśli aplikacja chce utworzyć proces potomny po wykonaniu wywołań IBM MQ , kod aplikacji musi używać kodu `fork()` z wartością `exec()` , aby upewnić się, że element potomny jest nową instancją, a nie dokładną kopią elementu nadrzędnego.

Jeśli aplikacja nie używa produktu `exec()`, wywołanie funkcji API języka IBM MQ wykonane w procesie potomnym zwraca błąd MQRC_ENVIRONMENT_ERROR.

### Linux AIX *Obsługa sygnału AIX and Linux*

Ogólnie rzecz biorąc, systemy AIX and Linux zostały przeniesione ze środowiska niewielowątkowego (procesowego) do środowiska wielowątkowego. W wielu przypadkach sygnały i obsługa sygnałów, chociaż są obsługiwane, nie mieszczą się dobrze w środowisku wielowątkowym i istnieją różne ograniczenia.

Ogólnie rzecz biorąc, systemy AIX and Linux zostały przeniesione ze środowiska niewielowątkowego (procesowego) do środowiska wielowątkowego. W środowisku niewielowątkowym niektóre funkcje mogły być zaimplementowane tylko przy użyciu sygnałów, chociaż większość aplikacji nie musiała być świadoma sygnałów i obsługi sygnałów. W środowisku wielowątkowym operacje podstawowe oparte na wątkach obsługują niektóre funkcje, które były implementowane w środowiskach niewielowątkowych przy użyciu sygnałów.

W wielu przypadkach sygnały i obsługa sygnałów, chociaż są obsługiwane, nie mieszczą się dobrze w środowisku wielowątkowym i istnieją różne ograniczenia. Może to być problematyczne podczas integrowania kodu aplikacji z różnymi bibliotekami oprogramowania pośredniego (działającymi jako część aplikacji) w środowisku wielowątkowym, w którym każdy z nich próbuje obsłużyć sygnały. Tradycyjne podejście do zapisywania i odtwarzania procedur obsługi sygnałów (zdefiniowane dla każdego procesu), które działało, gdy w obrębie procesu istniał tylko jeden wątek wykonywania, nie działa w środowisku wielowątkowym. Jest to spowodowane tym, że wiele wątków wykonywania może próbować zapisać i odtworzyć zasób całego procesu, co może mieć nieprzewidywalne skutki.

Linux AIX *Aplikacje niewątkowe*

Każda funkcja MQI konfiguruje własną procedurę obsługi sygnałów dla sygnałów. Procedury obsługi użytkowników dla tych funkcji są zastępowane na czas trwania wywołania funkcji MQI. Inne sygnały mogą być wychwycone w normalny sposób przez procedury obsługi napisane przez użytkownika.

Każda funkcja MQI ustawia własną procedurę obsługi sygnałów dla sygnałów:

PODPISZ
SIGBUS
SIGFPE,
SIGSEGV
SIGILL

Procedury obsługi użytkowników dla tych funkcji są zastępowane na czas trwania wywołania funkcji MQI. Inne sygnały mogą być wychwycone w normalny sposób przez procedury obsługi napisane przez użytkownika. Jeśli procedura obsługi nie zostanie zainstalowana, zostaną pozostawione domyślne działania (na przykład ignorowanie, zrzut pamięci lub wyjście).

Po tym, jak program IBM MQ obsłuży sygnał synchroniczny (SIGSEGV, SIGBUS, SIGFPE, SIGILL), próbuje on przekazać sygnał do dowolnej zarejestrowanej procedury obsługi sygnału przed wywołaniem funkcji MQI.

Linux AIX *Aplikacje wielowątkowe*

Wątek jest traktowany jako połączony z programem IBM MQ z MQCONN (lub MQCONNX) do MQDISC.

## Sygnały synchroniczne

Sygnały synchroniczne pojawiają się w określonym wątku.

Systemy AIX and Linux umożliwiają bezpieczne skonfigurowanie procedury obsługi sygnałów dla całego procesu. Jednak program IBM MQ konfiguruje własną procedurę obsługi dla następujących sygnałów w procesie aplikacji, gdy każdy wątek jest połączony z programem IBM MQ:

SIGBUS
SIGFPE,
SIGSEGV
SIGILL

W przypadku pisania aplikacji wielowątkowych dla każdego sygnału istnieje tylko jedna procedura obsługi sygnału dla całego procesu. Gdy IBM MQ konfiguruje swoje własne procedury obsługi sygnału synchronicznego, zapisuje wszystkie wcześniej zarejestrowane procedury obsługi dla każdego sygnału. Po tym, jak program IBM MQ obsłuży jeden z wymienionych sygnałów, program IBM MQ podejmie próbę wywołania procedury obsługi sygnału, która była aktywna w czasie pierwszego połączenia IBM

MQ w procesie. Poprzednio zarejestrowane procedury obsługi są odtwarzane po odłączeniu wszystkich wątków aplikacji od programu IBM MQ.

Ponieważ procedury obsługi sygnałów są zapisywane i odtwarzane przez program IBM MQ, wątki aplikacji nie mogą ustanawiać procedur obsługi sygnałów dla tych sygnałów, jeśli istnieje możliwość, że inny wątek tego samego procesu jest również połączony z programem IBM MQ.

**Uwaga:** Gdy aplikacja lub biblioteka oprogramowania pośredniego (działająca jako część aplikacji) ustanawia procedurę obsługi sygnału, gdy wątek jest połączony z serwerem IBM MQ, procedura obsługi sygnału aplikacji musi wywołać odpowiednią procedurę obsługi IBM MQ podczas przetwarzania tego sygnału.

Przy ustanawianiu i odtwarzaniu procedur obsługi sygnału, ogólną zasadą jest, że ostatnia procedura obsługi sygnału, która ma zostać zapisana, musi być pierwszą procedurą, która zostanie odtworzona:

- Gdy aplikacja ustanawia procedurę obsługi sygnału po nawiązaniu połączenia z serwerem IBM MQ, należy odtworzyć poprzednią procedurę obsługi sygnału, zanim aplikacja rozłączy się z serwerem IBM MQ.
- Gdy aplikacja ustanawia procedurę obsługi sygnału przed nawiązaniem połączenia z programem IBM MQ, musi ona odłączyć się od programu IBM MQ przed odtworzeniem swojej procedury obsługi sygnału.

**Uwaga:** Nieprzestrzeganie ogólnej zasady, zgodnie z którą ostatnia procedura obsługi sygnału, która ma zostać zapisana, musi być pierwszą procedurą, która ma zostać przywrócona, może skutkować nieoczekiwaną obsługą sygnału w aplikacji i, potencjalnie, utratą sygnałów przez aplikację.

## Sygnały asynchroniczne

IBM MQ nie używa żadnych sygnałów asynchronicznych w aplikacjach wielowątkowych, chyba że są to aplikacje klienckie.

## Dodatkowe uwagi dotyczące wielowątkowych aplikacji klienckich

Program IBM MQ obsługuje następujące sygnały podczas operacji we/wy na serwerze. Sygnały te są definiowane przez stos komunikacji. Aplikacja nie może ustanowić procedury obsługi sygnału dla tych sygnałów, gdy wątek jest połączony z menedżerem kolejek:

SIGPIPE (dla TCP/IP)

Linux AIX *Dodatkowe uwagi dotyczące korzystania z obsługi sygnałów AIX and Linux w MQI*

W przypadku używania interfejsu MQI do obsługi sygnałów w systemie AIX and Linuxistnieją dodatkowe uwagi dotyczące aplikacji krótkiej ścieżki, wywołań funkcji MQI w procedurach obsługi sygnałów, sygnałów podczas wywołań MQI, programów zewnętrznych i instalowalnych usług oraz programów obsługi wyjścia VMS.

## Krótka ścieżka (zaufane) aplikacje

Aplikacje krótkiej ścieżki działają w tym samym procesie, co produkt IBM MQ i działają w środowisku wielowątkowym.

W tym środowisku IBM MQ obsługuje sygnały synchroniczne SIGSEGV, SIGBUS, SIGFPE i SIGILL. Wszystkie inne sygnały nie mogą być dostarczane do aplikacji krótkiej ścieżki, gdy jest ona połączona z serwerem IBM MQ. Zamiast tego muszą być zablokowane lub obsługiwane przez aplikację. Jeśli aplikacja krótkiej ścieżki przechwyci takie zdarzenie, należy zatrzymać i zrestartować menedżer kolejek lub pozostawić go w stanie niezdefiniowanym. Pełną listę ograniczeń dotyczących aplikacji krótkiej ścieżki w produkcie MQCONNX zawiera sekcja “Nawiązywanie połączenia z menedżerem kolejek przy użyciu wywołania MQCONNX” na stronie 761.

## Wywołania funkcji MQI w procedurach obsługi sygnałów

Podczas pracy w procedurze obsługi sygnału nie należy wywoływać funkcji MQI.

Próba wywołania funkcji MQI z procedury obsługi sygnału, gdy inna funkcja MQI jest aktywna, powoduje zwrócenie wartości MQRC_CALL_IN_PROGRESS. Próba wywołania funkcji MQI z procedury obsługi sygnału, gdy żadna inna funkcja MQI nie jest aktywna, prawdopodobnie zakończy się niepowodzeniem w czasie wykonywania operacji z powodu ograniczeń systemu operacyjnego, w którym tylko wywołania selektywne mogą być wysyłane z procedury obsługi lub w jej obrębie.

W przypadku metod destruktora języka C + +, które mogą być wywoływane automatycznie podczas wyjścia programu, może nie być możliwe zatrzymanie wywołania funkcji MQI. Zignoruj wszystkie błędy dotyczące komendy MQRC_CALL_IN_PROGRESS. Jeśli procedura obsługi sygnału wywoła funkcję exit (), program IBM MQ wycofa niezatwierdzone komunikaty w punkcie synchronizacji i zamknie wszystkie otwarte kolejki.

### Sygnały podczas wywołań MQI

Funkcje MQI nie zwracają kodu EINTR ani żadnego odpowiednika dla aplikacji.

Jeśli sygnał wystąpi podczas wywołania MQI, a procedura obsługi wywoła funkcję *return*, wywołanie nadal będzie działać tak, jakby sygnał nie wystąpił. W szczególności komenda MQGET nie może zostać przerwana przez sygnał, który natychmiast zwróci sterowanie do aplikacji. Aby przerwać operację MQGET, należy ustawić kolejkę na wartość GET_DISABLED; można także użyć pętli wokół wywołania MQGET z skończonym czasem utraty ważności (MQGMO_WAIT z ustawioną wartością gmo.WaitInterval ) i użyć procedury obsługi sygnału (w środowisku niewielowątkowym) lub równoważnej funkcji w środowisku wielowątkowym, aby ustawić flagę, która przerywa pętlę.

AIX W środowisku AIX program IBM MQ wymaga, aby wywołania systemowe przerywane przez sygnały były restartowane. Podczas ustanawiania własnej procedury obsługi sygnału z sigaction (2), należy ustawić flagę SA_RESTART w polu sa_flags nowej struktury działania, w przeciwnym razie IBM MQ może nie być w stanie zakończyć żadnego wywołania przerwanego przez sygnał.

### Procedury zewnętrzne i usługi instalowalne

Procedury zewnętrzne i instalowalne usługi, które działają jako część procesu IBM MQ w środowisku wielowątkowym, mają takie same ograniczenia jak w przypadku aplikacji krótkiej ścieżki. Należy wziąć pod uwagę, że są one trwale połączone z programem IBM MQ i nie używają sygnałów ani niewątkowo bezpiecznych wywołań systemu operacyjnego.

## Nawiązywanie i rozłączanie połączenia z menedżerem kolejek

Aby można było używać usług programistycznych IBM MQ , program musi mieć połączenie z menedżerem kolejek. Ten temat zawiera informacje o nawiązywaniu połączenia z menedżerem kolejek i rozłączaniu się z nim.

Sposób nawiązywania tego połączenia zależy od platformy i środowiska, w którym działa program:

Multi **IBM MQ for Multiplatforms**
Programy działające w tych środowiskach mogą używać wywołania MQCONN MQI w celu nawiązania połączenia z menedżerem kolejek i wywołania MQDISC w celu rozłączenia się z menedżerem kolejek. Alternatywnie programy mogą używać wywołania MQCONNX.

z/OS **IBM MQ for z/OS wsadowe**
Programy działające w tym środowisku mogą używać wywołania MQCONN MQI w celu nawiązania połączenia z menedżerem kolejek i wywołania MQDISC w celu rozłączenia się z menedżerem kolejek. Alternatywnie programy mogą używać wywołania MQCONNX.

Programy wsadowe z/OS mogą łączyć się, kolejno lub współbieżnie, z wieloma menedżerami kolejek w tej samej bazie TCB.

z/OS **IMS**
Podczas uruchamiania region sterujący IMS jest połączony z co najmniej jednym menedżerem kolejek. To połączenie jest kontrolowane przez komendy systemu IMS . Informacje na temat sterowania adapterem IMS w systemie z/OSzawiera sekcja Administrowanie programem IBM MQ for z/OS.

Jednak programy piszące kolejkowania komunikatów IMS muszą użyć wywołania MQI MQCONN w celu określenia menedżera kolejek, z którym mają się połączyć. Mogą oni używać wywołania MQDISC w celu rozłączenia się z tym menedżerem kolejek.

Po wywołaniu programu IMS , które ustanawia punkt synchronizacji, a przed przetworzeniem komunikatu dla innego użytkownika, adapter IMS zapewnia, że aplikacja zamknie uchwyty i rozłączy się z menedżerem kolejek. Patrz “Punkty synchronizacji w aplikacjach IMS” na stronie 883.

Programy IMS mogą łączyć się, kolejno lub współbieżnie, z wieloma menedżerami kolejek w tej samej bazie TCB.

**z/OS CICS Transaction Server dla systemu z/OS**

Programy CICS nie muszą wykonywać żadnych działań w celu nawiązania połączenia z menedżerem kolejek, ponieważ sam system CICS jest połączony. To połączenie jest zwykle nawiązywane automatycznie podczas inicjowania, ale można również użyć transakcji CKQC, która jest dostarczana z produktem IBM MQ for z/OS. Więcej informacji na temat CKQC zawiera sekcja Administrowanie programem IBM MQ for z/OS.

Zadania programu CICS mogą łączyć się tylko z menedżerem kolejek, z którym jest połączony region programu CICS .

Programy CICS mogą również używać wywołań połączenia i rozłączenia MQI (MQCONN i MQDISC). Można to zrobić, aby umożliwić przeniesienie tych aplikacji do środowisk innych niżCICS przy minimalnym rekodowaniu. Jednak w środowisku CICS wywołania te *zawsze* są pomyślnie kończone. Oznacza to, że kod powrotu może nie odzwierciedlać rzeczywistego stanu połączenia z menedżerem kolejek.

**TXSeries dla systemu Windows i systemów otwartych**

Programy te nie muszą wykonywać żadnej pracy w celu nawiązania połączenia z menedżerem kolejek, ponieważ sam system CICS jest połączony. Dlatego w danym momencie obsługiwane jest tylko jedno połączenie. Aplikacje CICS muszą wywołać wywołanie MQCONN w celu uzyskania uchwytu połączenia oraz wywołanie MQDISC przed wyjściem.

Aby uzyskać więcej informacji na temat nawiązywania i rozłączania połączenia z menedżerem kolejek, należy użyć następujących odsyłaczy:

- “Nawiązywanie połączenia z menedżerem kolejek przy użyciu wywołania MQCONN” na stronie 760
- “Nawiązywanie połączenia z menedżerem kolejek przy użyciu wywołania MQCONNX” na stronie 761
- “Odłączanie programów od menedżera kolejek za pomocą komendy MQDISC” na stronie 765

**Pojęcia pokrewne**

“Przegląd interfejsu kolejki komunikatów” na stronie 745
Sekcja zawiera informacje na temat komponentów interfejsu kolejki komunikatów (Message Queue Interface-MQI).

“Otwieranie i zamykanie obiektów” na stronie 766
Informacje te udostępniają wgląd w otwieranie i zamykanie obiektów IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 777
Ta sekcja zawiera informacje na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 792
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Uzyskiwanie informacji o atrybutach obiektu i ustawianie ich” na stronie 876
Atrybuty są właściwościami definiującymi charakterystykę obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 879
W tej sekcji opisano sposób zatwierdzania i wycofywania wszelkich odtwarzalnych operacji pobierania i umieszczania, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy” na stronie 891
Informacje o wyzwalaczach i sposobie uruchamiania aplikacji IBM MQ za pomocą wyzwalaczy.

“Praca z funkcją MQI i klastrami” na stronie 911
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do łączenia w klastry.

“Używanie i pisanie aplikacji w systemie IBM MQ for z/OS” na stronie 916
Aplikacje IBM MQ for z/OS mogą być tworzone z programów działających w wielu różnych środowiskach. Oznacza to, że mogą korzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje mostu IMS i IMS w systemie IBM MQ for z/OS” na stronie 71
Informacje te są pomocne podczas pisania aplikacji IMS przy użyciu produktu IBM MQ.

### *Nawiązywanie połączenia z menedżerem kolejek przy użyciu wywołania MQCONN*

Ten temat zawiera informacje o nawiązywaniu połączenia z menedżerem kolejek przy użyciu wywołania MQCONN.

Ogólnie można nawiązać połączenie z konkretnym menedżerem kolejek lub z domyślnym menedżerem kolejek:

- W przypadku bazy danych IBM MQ for z/OSw środowisku zadań wsadowych domyślny menedżer kolejek jest określony w module CSQBDEFV.
- W przypadku produktu IBM MQ for Multiplatformsdomyślny menedżer kolejek jest określony w pliku mqs.ini .

Alternatywnie w środowiskach zadań wsadowych, TSO i RRS systemu z/OS MVS można nawiązać połączenie z dowolnym menedżerem kolejek w obrębie grupy współużytkowania kolejek. Żądanie MQCONN lub MQCONNX wybiera dowolny z aktywnych członków grupy.

Po nawiązaniu połączenia z menedżerem kolejek musi to być zadanie lokalne. Musi on należeć do tego samego systemu, co aplikacja IBM MQ .

W środowisku IMS menedżer kolejek musi być połączony z regionem sterującym IMS oraz z regionem zależnym używanym przez program. Domyślny menedżer kolejek jest określany w module CSQQDEFV podczas instalowania produktu IBM MQ for z/OS .

W środowisku produktu TXSeries CICS oraz w produkcie TXSeries dla systemów Windows i AIXmenedżer kolejek musi być zdefiniowany jako zasób XA w produkcie CICS.

Aby nawiązać połączenie z domyślnym menedżerem kolejek, wywołaj komendę MQCONN, podając nazwę składającą się całkowicie z odstępów lub rozpoczynającą się znakiem o kodzie zero (X'00 ').

Aby aplikacja mogła pomyślnie nawiązać połączenie z menedżerem kolejek, musi być autoryzowana. Więcej informacji zawiera sekcja Bezpieczeństwo.

Dane wyjściowe komendy MQCONN to:

- Uchwyt połączenia ( **Hconn** )
- Kod zakończenia
- Kod przyczyny

Użyj uchwytu połączenia w kolejnych wywołaniach MQI.

Jeśli kod przyczyny wskazuje, że aplikacja jest już połączona z tym menedżerem kolejek, zwrócony uchwyt połączenia jest taki sam, jak zwrócony podczas pierwszego połączenia aplikacji. W tej sytuacji aplikacja nie może wywołać wywołania MQDISC, ponieważ oczekuje, że aplikacja wywołująca pozostanie połączona.

Zasięg uchwytu połączenia jest taki sam, jak zasięg uchwytu obiektu (patrz sekcja “Otwieranie obiektów przy użyciu wywołania MQOPEN” na stronie 767 ).

Opisy parametrów są podane w opisie wywołania MQCONN w tabeli MQCONN.

Wywołanie MQCONN kończy się niepowodzeniem, jeśli menedżer kolejek jest w stanie wyciszania podczas wywoływania wywołania lub jeśli menedżer kolejek jest zamykany.

## Zasięg MQCONN lub MQCONNX

Zasięgiem wywołania MQCONN lub MQCONNX jest zazwyczaj wątek, który je wywołał. Oznacza to, że uchwyt połączenia zwrócony z wywołania jest poprawny tylko w wątku, który wywołał wywołanie. Za pomocą uchwytu można wykonać tylko jedno wywołanie w danym momencie. Jeśli jest używany z innego

wątku, jest odrzucany jako niepoprawny. Jeśli w aplikacji istnieje wiele wątków, a każdy z nich ma używać wywołań IBM MQ , każdy z nich musi wywołać komendę MQCONN lub MQCONNX.

Nie jest konieczne, aby każde wywołanie do tego samego menedżera kolejek było wykonywane, gdy proces wykonuje wiele wywołań MQCONN. Jednak w danym momencie z wątku można nawiązać tylko jedno połączenie IBM MQ . Alternatywnie można rozważyć użycie opcji "Współużytkowane (niezależne od wątku) połączenia z MQCONNX" na stronie 762 , aby zezwolić na używanie wielu połączeń IBM MQ z jednego wątku i połączenia IBM MQ z dowolnego wątku.[7]

Jeśli aplikacja działa jako klient, może nawiązać połączenie z więcej niż jednym menedżerem kolejek w obrębie wątku.

### *Nawiązywanie połączenia z menedżerem kolejek przy użyciu wywołania MQCONNX*

Wywołanie MQCONNX jest podobne do wywołania MQCONN, ale zawiera opcje sterujące sposobem działania wywołania.

Jako dane wejściowe dla MQCONNX można podać nazwę menedżera kolejek z/OS lub nazwę grupy współużytkowania kolejek w z/OS systemach kolejek współużytkowanych. Opcje sterujące nawiązaniem połączenia z menedżerem kolejek są udostępniane w strukturze nazywanej MQCNO.

Dane wyjściowe komendy MQCONNX są następujące:

- Uchwyt połączenia (Hconn)
- Kod zakończenia
- Kod przyczyny

Uchwyt połączenia jest używany w kolejnych wywołaniach MQI.

Opcje połączenia ustawione w polu *Options* struktury MQCNO umożliwiają sterowanie kilkoma atrybutami połączenia. Szczególną uwagę należy zwrócić na następujące grupy opcji:

- Opcje powiązania umożliwiają tworzenie *zaufanych aplikacji* . Zaufane aplikacje oznaczają, że aplikacja IBM MQ i agent lokalnego menedżera kolejek stają się tym samym procesem. Ponieważ proces agenta nie musi już używać interfejsu w celu uzyskania dostępu do menedżera kolejek, te aplikacje stają się rozszerzeniem menedżera kolejek. To zachowanie jest żądane przez określenie opcji MQCNO_FASTPATH_BINDING . Więcej informacji na temat ograniczeń, które mają zastosowanie do zaufanych aplikacji, zawiera sekcja "Ograniczenia dotyczące zaufanych aplikacji" na stronie 761.
- Opcje współużytkowania uchwytów umożliwiają tworzenie połączeń współużytkowanych. Współużytkowane połączenia mogą współużytkować uchwyty między różnymi wątkami w ramach tego samego procesu. Więcej informacji na temat połączeń współużytkowanych zawiera sekcja "Współużytkowane (niezależne od wątku) połączenia z MQCONNX" na stronie 762.

Komenda MQCNO umożliwia również aplikacji sterowanie uwierzytelnianiem połączenia z menedżerem kolejek. Referencje uwierzytelniania można określić w strukturze MQCSP przywoływanej ze struktury MQCNO.

Pełny opis parametrów wywołania MQCONNX i atrybutów połączenia, które mogą być kontrolowane, zawiera sekcja Menedżer kolejek MQCONNX-Connect (rozszerzony).

*Ograniczenia dotyczące zaufanych aplikacji*

Ograniczenia dotyczące zaufanych aplikacji. Niektóre ograniczenia mają zastosowanie do wszystkich platform, a inne są specyficzne dla platformy.

T

- Należy jawnie odłączyć zaufane aplikacje od menedżera kolejek.

[7] W przypadku korzystania z aplikacji wielowątkowych w systemach IBM MQ for AIX or Linux należy upewnić się, że aplikacje mają wystarczającą wielkość stosu dla wątków. Należy rozważyć użycie stosu o wielkości 256 kB lub większej, gdy aplikacje wielowątkowe wywołują wywołania MQI samodzielnie lub z innymi procedurami obsługi sygnałów (na przykład CICS ).

- Przed zakończeniem działania menedżera kolejek za pomocą komendy **endmqm** należy zatrzymać zaufane aplikacje.
- Z powiązaniem MQCNO_FASTPATH_BINDING nie można używać asynchronicznych sygnałów ani przerwań licznika czasu (takich jak `sigkill`).
- Na wszystkich platformach wątek w zaufanej aplikacji nie może nawiązać połączenia z menedżerem kolejek, gdy inny wątek w tym samym procesie jest połączony z innym menedżerem kolejek.
- Linux AIX W systemach AIX and Linux należy użyć systemu mqm jako efektywny identyfikator userID i groupID dla wszystkich wywołań MQI. Identyfikatory te można zmienić przed wykonaniem wywołania innego niż MQI wymagającego uwierzytelnienia (na przykład otwarcia pliku), ale przed wykonaniem następnego wywołania MQI należy zmienić je z powrotem na mqm.
- IBM i W systemie IBM i:
  1. Zaufane aplikacje muszą być uruchamiane z profilem użytkownika QMQM. Nie wystarczy, że profil użytkownika jest członkiem grupy QMQM lub że program adoptuje uprawnienie QMQM. Profil użytkownika QMQM może nie być używany do wpisywania się do zadań interaktywnych lub może być określony w opisie zadania dla zadań uruchamianych zaufanych aplikacji. W takim przypadku należy użyć funkcji API wymiany profili systemu IBM i , QSYGETPH, QWTSETP i QSYRLSPH, aby tymczasowo zmienić bieżącego użytkownika zadania na QMQM podczas działania programów IBM MQ . Szczegółowe informacje na temat tych funkcji, wraz z przykładem ich użycia, zawiera sekcja Security APIs w dokumentacji IBM i*Application programming interfaces* (Aplikacyjne interfejsy programistyczne).
  2. Nie należy anulować zaufanych aplikacji za pomocą opcji System-Request 2 lub zakończyć zadania, w których są one uruchomione, za pomocą komendy ENDJOB.
- ALW W systemach AIX, Linux, and Windows zaufane aplikacje 32-bitowe nie są obsługiwane. Jeśli zostanie podjęta próba uruchomienia zaufanej aplikacji 32-bitowej, zostanie ona zdegradowana do standardowego połączenia powiązanego.

*Współużytkowane (niezależne od wątku) połączenia z MQCONNX*
Te informacje umożliwiają zapoznanie się ze współużytkowanymi połączeniami z usługą MQCONNX oraz z niektórymi uwagami dotyczącymi użycia, które należy wziąć pod uwagę.

**Uwaga:** Nieobsługiwane w systemie IBM MQ for z/OS.

Na platformach IBM MQ innych niż IBM MQ for z/OSpołączenie nawiązane za pomocą programu MQCONN jest dostępne tylko dla wątku, który nawiąże połączenie. Opcje wywołania MQCONNX umożliwiają utworzenie połączenia, które może być współużytkowane przez wszystkie wątki w procesie. Jeśli aplikacja działa w środowisku transakcyjnym, które wymaga wywołania MQI w tym samym wątku, należy użyć następującej opcji domyślnej:

**MQCNO_HANDLE_SHARE_NONE**
Tworzy połączenie niewspółużytkowane.

W większości innych środowisk można użyć jednej z następujących opcji połączeń współużytkowanych niezależnych od wątku:

**MQCNO_HANDLE_SHARE_BLOCK**
Tworzy połączenie współużytkowane. W przypadku połączenia MQCNO_HANDLE_SHARE_BLOCK , jeśli połączenie jest obecnie używane przez wywołanie MQI w innym wątku, wywołanie MQI oczekuje na zakończenie bieżącego wywołania MQI.

**MQCNO_HANDLE_SHARE_NO_BLOCK**
Tworzy połączenie współużytkowane. W przypadku połączenia MQCNO_HANDLE_SHARE_NO_BLOCK , jeśli połączenie jest obecnie używane przez wywołanie MQI w innym wątku, wywołanie MQI natychmiast kończy się niepowodzeniem z powodu MQRC_CALL_IN_PROGRESS.

Z wyjątkiem środowiska MTS ( Microsoft Transaction Server), wartością domyślną jest MQCNO_HANDLE_SHARE_NONE. W środowisku MTS wartością domyślną jest MQCNO_HANDLE_SHARE_BLOCK.

Uchwyt połączenia jest zwracany przez wywołanie MQCONNX . Uchwyt może być używany przez kolejne wywołania MQI z dowolnego wątku w procesie, wiążąc te wywołania z uchwytem zwróconym z MQCONNX. Wywołania MQI używające pojedynczego uchwytu współużytkowanego są szeregowane między wątkami.

Na przykład można wykonać następującą sekwencję działań przy użyciu współużytkowanego uchwytu:

1. Wątek 1 zgłasza MQCONNX i uzyskuje współużytkowany uchwyt *h1*
2. Wątek 1 otwiera kolejkę i wysyła żądanie pobrania za pomocą *h1*
3. Wątek 2 wysyła żądanie umieszczenia przy użyciu *h1*
4. Wątek 3 wysyła żądanie umieszczenia przy użyciu *h1*
5. Problemy z wątkiem 2 MQDISC przy korzystaniu z *h1*

Gdy uchwyt jest używany przez dowolny wątek, dostęp do połączenia jest niedostępny dla innych wątków. W sytuacjach, w których jest dopuszczalne, aby wątek oczekiwał na zakończenie poprzedniego wywołania z innego wątku, należy użyć parametru MQCONNX z opcją MQCNO_HANDLE_SHARE_BLOCK.

Jednak blokowanie może powodować trudności. Załóżmy, że w kroku “2” na stronie 763wątek 1 wysyła żądanie pobrania, które oczekuje na komunikaty, które nie zostały jeszcze wysłane (pobranie z oczekiwaniem). W takim przypadku wątki 2 i 3 pozostają w stanie oczekiwania (zablokowane) tak długo, jak długo trwa żądanie pobrania w wątku 1. Jeśli wolisz, aby wywołanie MQI zwróciło błąd, jeśli inne wywołanie MQI jest już uruchomione na uchwycie, użyj opcji MQCONNX z opcją MQCNO_HANDLE_SHARE_NO_BLOCK.

## Uwagi dotyczące użycia połączenia współużytkowanego

1. Wszystkie uchwyty obiektów (Hobj) utworzone przez otwarcie obiektu są powiązane z Hconn; tak więc dla współużytkowanego Hconn, Hobjs są również współużytkowane i używane przez dowolny wątek używający Hconn. Podobnie, każda jednostka pracy uruchomiona w ramach Hconn jest powiązana z tym Hconn; dlatego też jest ona współużytkowana przez wątki ze współużytkowanym Hconn.
2. *Każdy wątek* może wywołać komendę MQDISC w celu rozłączenia współużytkowanego połączenia Hconn, a nie tylko wątek, który wywołał odpowiednią komendę MQCONNX. Komenda MQDISC kończy działanie programu Hconn, co powoduje, że jest on niedostępny dla wszystkich wątków.
3. Pojedynczy wątek może używać wielu współużytkowanych połączeń Hconns szeregowo, na przykład użyć wywołania MQPUT do umieszczenia jednego komunikatu w jednym współużytkowanym połączeniu Hconn, a następnie umieścić inny komunikat przy użyciu innego współużytkowanego połączenia Hconn, przy czym każda operacja jest w innej lokalnej jednostce pracy.
4. Współużytkowane połączenia typu Hconns nie mogą być używane w globalnej jednostce pracy.

Multi *Użycie opcji wywołania MQCONNX z wartością MQ_CONNECT_TYPE*

Ten temat zawiera informacje o różnych opcjach wywołania MQCONNX i sposobie ich używania ze zmienną środowiskową **MQ_CONNECT_TYPE** .

**Uwaga: MQ_CONNECT_TYPE** ma wpływ tylko na powiązania STANDARD. W przypadku innych powiązań parametr **MQ_CONNECT_TYPE** jest ignorowany.

W systemie IBM MQ for Multiplatformsmożna użyć zmiennej środowiskowej **MQ_CONNECT_TYPE** w połączeniu z typem powiązania określonym w polu Options struktury MQCNO używanej w wywołaniu MQCONNX.

*Tabela 113. Sposób użycia opcji wywołania MQCONNX ze zmienną środowiskową **MQ_CONNECT_TYPE***

| Opcja wywołania MQCONNX | MQ_CONNECT_TYPE, zmienna środowiskowa | Wynik |
|---|---|---|
| STANDARDOWA | UNDEFINED | STANDARDOWA |
| STANDARDOWA | STANDARDOWA | STANDARDOWA |
| STANDARDOWA | Krótka ścieżka | STANDARDOWA |

| *Tabela 113. Sposób użycia opcji wywołania MQCONNX ze zmienną środowiskową **MQ_CONNECT_TYPE** (kontynuacja)* | | |
|---|---|---|
| **Opcja wywołania MQCONNX** | **MQ_CONNECT_TYPE, zmienna środowiskowa** | **Wynik** |
| STANDARDOWA | KLIENT | KLIENT |
| STANDARDOWA | LOKALNA | STANDARDOWA |

Jeśli opcja MQCNO_STANDARD_BINDING nie jest określona, można użyć opcji MQCNO_NONE, która domyślnie ma wartość MQCNO_STANDARD_BINDING.

### *Uwierzytelnianie i tożsamość dla MQCONN i MQCONNX*

To zadanie umożliwia poznanie sposobu, w jaki aplikacje mogą dostarczać referencje używane do uwierzytelniania podczas nawiązywania połączenia z produktem IBM MQ.

## Domyślna tożsamość użytkownika

Jeśli aplikacja używa interfejsu kolejki komunikatów (MQI) do nawiązania połączenia z produktem IBM MQ za pomocą wywołania MQCONN lub MQCONNX, tożsamość użytkownika jest zawsze ustanawiana i wiązana z połączeniem.

Domyślnie początkową tożsamością użytkownika jest zawsze tożsamość procesu systemu operacyjnego, w którym działa aplikacja. Ta początkowa tożsamość może być wystarczająca dla lokalnie powiązanych lub zaufanych połączeń aplikacji.

Gdy aplikacja łączy się z menedżerem kolejek za pomocą wywołania MQCONN, nie może modyfikować domyślnego identyfikatora użytkownika. Jednak następujące mechanizmy mogą zmienić identyfikator użytkownika, który jest powiązany z połączeniem:

- Wyjście zabezpieczeń po stronie klienta lub po stronie serwera.
- Reguły uwierzytelniania kanału w menedżerze kolejek.
- Identyfikator użytkownika klienta ustanowiony podczas wzajemnego uwierzytelniania TLS.

## Używanie usługi MQCONNX do dostarczania referencji

Funkcja MQCONNX zapewnia aplikacji większą kontrolę nad tożsamością, która jest powiązana z połączeniem. Aplikacja może dostarczyć strukturę MQCSP w ramach opcji połączenia, które są określone w parametrze **ConnectOpts** dla MQCONNX. Struktura MQCSP może zawierać referencje używane do ustanawiania tożsamości użytkownika. Produkt IBM MQ obsługuje następujące referencje w strukturze MQCSP:

- Identyfikator użytkownika i hasło.
- V 9.3.4 Z programu IBM MQ 9.3.4-znacznik uwierzytelniania, jeśli aplikacja nawiązuje połączenie z menedżerem kolejek, który działa w systemie AIX lub Linux .

Konfiguracja uwierzytelniania połączenia menedżera kolejek i uwierzytelniania kanału steruje sposobem przetwarzania referencji dostarczanych przez aplikację. Na przykład ta konfiguracja ma wpływ na następujące aspekty:

- Określa, czy poprawność referencji w strukturze MQCSP jest sprawdzana i w jaki sposób.
- Określa, czy identyfikator użytkownika w referencjach w strukturze MQCSP jest odwzorowany na inny identyfikator użytkownika.
- Określa, czy uwierzytelniony użytkownik jest następnie adoptowany jako kontekst dla aplikacji.

Więcej informacji na temat uwierzytelniania połączenia zawiera sekcja Uwierzytelnianie połączenia. Więcej informacji na temat uwierzytelniania kanału zawiera sekcja Rekordy uwierzytelniania kanału.

Kilka przykładowych programów napisanych w języku C, które używają interfejsu MQI, demonstruje sposób, w jaki struktura MQCSP jest używana do udostępniania referencji uwierzytelniających. Więcej informacji na ten temat zawierają następujące programy przykładowe:

- “Przykładowe programy Get” na stronie 1123
- “Przykładowe programy Put” na stronie 1135
- “Przykładowy program przeglądarki” na stronie 1111
- “Przykładowy program TLS” na stronie 1152

**Informacje pokrewne**

Identyfikowanie i uwierzytelnianie użytkowników przy użyciu struktury MQCSP

MQCSP-parametry zabezpieczeń

Identyfikowanie i uwierzytelnianie użytkowników

## *Odłączanie programów od menedżera kolejek za pomocą komendy MQDISC*

Ten temat zawiera informacje o odłączaniu programów od menedżera kolejek za pomocą komendy MQDISC.

Jeśli program, który nawiązał połączenie z menedżerem kolejek za pomocą wywołania MQCONN lub MQCONNX, zakończył całą interakcję z menedżerem kolejek, przerywa połączenie za pomocą wywołania MQDISC, z wyjątkiem następujących sytuacji:

- W przypadku aplikacji produktu CICS Transaction Server for z/OS , gdzie wywołanie jest opcjonalne, chyba że użyto wywołania MQCONNX i użytkownik chce usunąć znacznik połączenia przed zakończeniem działania aplikacji.
- W systemie IBM MQ for IBM i , w którym po wylogowaniu się z systemu operacyjnego wykonywane jest niejawne wywołanie MQDISC.

Jako dane wejściowe wywołania MQDISC należy podać uchwyt połączenia (Hconn), który został zwrócony przez MQCONN lub MQCONNX podczas nawiązywania połączenia z menedżerem kolejek.

Z wyjątkiem systemu CICS na platformie z/OS, po wywołaniu komendy MQDISC uchwyt połączenia (Hconn) nie jest już poprawny i nie można wywoływać kolejnych wywołań MQI, dopóki nie zostanie ponownie wywołana komenda MQCONN lub MQCONNX. Komenda MQDISC wykonuje niejawną komendę MQCLOSE dla wszystkich obiektów, które są nadal otwarte przy użyciu tego uchwytu.

z/OS W przypadku klienta połączonego z programem z/OS, gdy zostanie wysłane wywołanie MQDISC, wykonywane jest niejawne zatwierdzenie, ale wszystkie uchwyty kolejek, które są nadal otwarte, nie są zamykane do momentu rzeczywistego zakończenia kanału.

Jeśli do nawiązywania połączenia w systemie IBM MQ for z/OSużywany jest program MQCONNX, program MQDISC kończy również zasięg znacznika połączenia ustanowionego przez program MQCONNX. Jeśli jednak w aplikacji CICS, IMSlub RRS istnieje aktywna jednostka odtwarzania powiązana ze znacznikiem połączenia, usługa MQDISC jest odrzucana z kodem przyczyny MQRC_CONN_TAG_NOT_RELEASED.

Opisy parametrów są podane w opisie wywołania MQDISC w MQDISC.

### Jeśli nie wydano komendy MQDISC

Standardowe, niewspółużytkowane połączenie (Hconn) jest czyszczone po zakończeniu wątku tworzącego. Połączenie współużytkowane jest niejawnie wycofywane i rozłączane tylko wtedy, gdy cały proces kończy działanie. Jeśli wątek, który utworzył współużytkowany Hconn, zostanie zakończony, gdy Hconn nadal istnieje, nadal będzie można korzystać z Hconn.

### Sprawdzanie uprawnień

Wywołania MQCLOSE i MQDISC zwykle nie wykonują sprawdzania uprawnień.

W normalnym toku zdarzeń zadanie, które ma uprawnienia do otwierania lub łączenia się z obiektem IBM MQ , zamyka lub rozłącza się z tym obiektem. Nawet jeśli uprawnienie do zadania, które nawiązało połączenie z obiektem IBM MQ lub otworzyło ten obiekt, wywołania MQCLOSE i MQDISC są akceptowane.

## Otwieranie i zamykanie obiektów

Informacje te udostępniają wgląd w otwieranie i zamykanie obiektów IBM MQ .

Aby wykonać dowolną z następujących operacji, należy najpierw *otworzyć* odpowiedni obiekt IBM MQ :

- Umieszczają komunikaty w kolejce
- Pobieranie (przeglądanie lub pobieranie) komunikatów z kolejki
- Ustawianie atrybutów obiektu
- Uzyskiwanie informacji o atrybutach dowolnego obiektu

Użyj wywołania MQOPEN, aby otworzyć obiekt, korzystając z opcji wywołania, aby określić, co chcesz zrobić z obiektem. Jedynym wyjątkiem jest umieszczenie pojedynczego komunikatu w kolejce, a następnie natychmiastowe zamknięcie kolejki. W takim przypadku można pominąć etap *otwierania* za pomocą wywołania MQPUT1 (patrz sekcja “Umieszczanie jednego komunikatu w kolejce za pomocą wywołania MQPUT1” na stronie 786 ).

Przed otwarciem obiektu za pomocą wywołania MQOPEN należy połączyć program z menedżerem kolejek. Zostało to szczegółowo wyjaśnione dla wszystkich środowisk w sekcji “Nawiązywanie i rozłączanie połączenia z menedżerem kolejek” na stronie 758.

Istnieją cztery typy obiektów IBM MQ , które można otworzyć:

- Kolejka
- Lista nazw
- Definicja procesu
- Menedżer kolejek

Wszystkie te obiekty są otwierane w podobny sposób przy użyciu wywołania MQOPEN. Więcej informacji na temat obiektów IBM MQ zawiera sekcja Typy obiektów.

Ten sam obiekt można otworzyć więcej niż jeden raz i za każdym razem, gdy jest dostępny nowy uchwyt obiektu. Istnieje możliwość przeglądania komunikatów w kolejce za pomocą jednego uchwytu i usuwania komunikatów z tej samej kolejki za pomocą innego uchwytu. Dzięki temu zasoby nie są wykorzystywane do zamykania i ponownego otwierania tego samego obiektu. Można również otworzyć kolejkę w celu przeglądania komunikatów *i* usuwania w tym samym czasie.

Ponadto można otworzyć wiele obiektów za pomocą jednej komendy MQOPEN i zamknąć je za pomocą komendy MQCLOSE. Więcej informacji na ten temat zawiera sekcja “Lista dystrybucyjna” na stronie 787 .

Podczas próby otwarcia obiektu menedżer kolejek sprawdza, czy użytkownik ma uprawnienia do otwarcia tego obiektu dla opcji określonych w wywołaniu MQOPEN.

Obiekty są zamykane automatycznie po odłączeniu się programu od menedżera kolejek. W środowisku IMS rozłączenie jest wymuszane, gdy program rozpoczyna przetwarzanie dla nowego użytkownika po wywołaniu GU (get unique) IMS . Na platformie IBM i obiekty są zamykane automatycznie po zakończeniu zadania.

Dobrą praktyką programowania jest zamykanie otwartych obiektów. W tym celu należy użyć wywołania MQCLOSE.

Aby uzyskać więcej informacji na temat otwierania i zamykania obiektów, należy skorzystać z następujących odsyłaczy:

- “Otwieranie obiektów przy użyciu wywołania MQOPEN” na stronie 767
- “Tworzenie kolejek dynamicznych” na stronie 775
- “Otwieranie kolejek zdalnych” na stronie 776

- “Zamykanie obiektów przy użyciu wywołania MQCLOSE” na stronie 776

**Pojęcia pokrewne**

“Przegląd interfejsu kolejki komunikatów” na stronie 745
Sekcja zawiera informacje na temat komponentów interfejsu kolejki komunikatów (Message Queue Interface-MQI).

“Nawiązywanie i rozłączanie połączenia z menedżerem kolejek” na stronie 758
Aby można było używać usług programistycznych IBM MQ , program musi mieć połączenie z menedżerem kolejek. Ten temat zawiera informacje o nawiązywaniu połączenia z menedżerem kolejek i rozłączaniu się z nim.

“Umieszczanie komunikatów w kolejce” na stronie 777
Ta sekcja zawiera informacje na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 792
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Uzyskiwanie informacji o atrybutach obiektu i ustawianie ich” na stronie 876
Atrybuty są właściwościami definiującymi charakterystykę obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 879
W tej sekcji opisano sposób zatwierdzania i wycofywania wszelkich odtwarzalnych operacji pobierania i umieszczania, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy” na stronie 891
Informacje o wyzwalaczach i sposobie uruchamiania aplikacji IBM MQ za pomocą wyzwalaczy.

“Praca z funkcją MQI i klastrami” na stronie 911
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do łączenia w klastry.

“Używanie i pisanie aplikacji w systemie IBM MQ for z/OS” na stronie 916
Aplikacje IBM MQ for z/OS mogą być tworzone z programów działających w wielu różnych środowiskach. Oznacza to, że mogą korzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje mostu IMS i IMS w systemie IBM MQ for z/OS” na stronie 71
Informacje te są pomocne podczas pisania aplikacji IMS przy użyciu produktu IBM MQ.

### *Otwieranie obiektów przy użyciu wywołania MQOPEN*

Ten temat zawiera informacje o otwieraniu obiektów za pomocą wywołania MQOPEN.

Jako dane wejściowe wywołania MQOPEN należy podać:

- Uchwyt połączenia. W przypadku aplikacji CICS w systemie z/OSmożna określić stałą MQHC_DEF_HCONN (która ma wartość zero) lub użyć uchwytu połączenia zwróconego przez wywołanie MQCONN lub MQCONNX. W przypadku innych programów należy zawsze używać uchwytu połączenia zwróconego przez wywołanie MQCONN lub MQCONNX.
- Opis obiektu, który ma zostać otwarty przy użyciu struktury deskryptora obiektu (MQOD).
- Co najmniej jedna opcja sterująca działaniem wywołania.

Dane wyjściowe komendy MQOPEN są następujące:

- Uchwyt obiektu reprezentujący dostęp użytkownika do obiektu. Tej opcji należy użyć na wejściu dla kolejnych wywołań MQI.
- Zmodyfikowana struktura deskryptora obiektu, jeśli tworzona jest kolejka dynamiczna (która jest obsługiwana na platformie użytkownika).
- Kod zakończenia.
- Kod przyczyny.

## Zasięg uchwytu obiektu

Zasięg uchwytu obiektu (Hobj) jest taki sam jak zasięg uchwytu połączenia (Hconn).

Jest to omówione w sekcji “Zasięg MQCONN lub MQCONNX” na stronie 760 i “Współużytkowane (niezależne od wątku) połączenia z MQCONNX” na stronie 762. Jednak w niektórych środowiskach należy wziąć pod uwagę następujące kwestie:

**CICS**
W programie CICS uchwytu można używać tylko w ramach tego samego zadania CICS , z którego wykonano wywołanie MQOPEN.

**Zadanie wsadowe IMS i z/OS**
W środowisku IMS i w środowisku wsadowym można użyć uchwytu w ramach tej samej czynności, ale nie w ramach podczynności.

Opisy parametrów wywołania MQOPEN zawiera sekcja MQOPEN.

W poniższych sekcjach opisano informacje, które należy podać jako dane wejściowe dla MQOPEN.

## Identyfikowanie obiektów (struktura MQOD)

Użyj struktury MQOD, aby zidentyfikować obiekt, który ma zostać otwarty. Ta struktura jest parametrem wejściowym wywołania MQOPEN. (Struktura jest modyfikowana przez menedżer kolejek, gdy do utworzenia kolejki dynamicznej używane jest wywołanie MQOPEN).

Szczegółowe informacje na temat struktury MQOD zawiera sekcja MQOD.

Informacje na temat używania struktury MQOD dla list dystrybucyjnych zawiera sekcja “Korzystanie ze struktury MQOD” na stronie 789 w sekcji “Lista dystrybucyjna” na stronie 787.

*Tłumaczenie nazw*
Sposób rozstrzygania nazw kolejek i menedżerów kolejek przez wywołanie MQOPEN.

**Uwaga:** Alias menedżera kolejek to definicja kolejki zdalnej bez pola RNAME .

Po otwarciu kolejki IBM MQ wywołanie MQOPEN wykonuje funkcję tłumaczenia nazw dla podanej nazwy kolejki. Określa, w której kolejce menedżer kolejek wykonuje kolejne operacje. Oznacza to, że po określeniu nazwy kolejki aliasowej lub kolejki zdalnej w deskryptorze obiektu (MQOD) wywołanie tłumaczy nazwę na kolejkę lokalną lub kolejkę transmisji. Jeśli kolejka jest otwarta dla dowolnego typu danych wejściowych, przeglądania lub ustawiania, jest rozstrzygana jako kolejka lokalna, jeśli istnieje, i kończy się niepowodzeniem, jeśli nie istnieje. Jest on rozstrzygany jako kolejka nielokalna tylko wtedy, gdy jest otwarty tylko dla danych wyjściowych, tylko dla zapytań lub tylko dla danych wyjściowych i tylko dla zapytań. Przegląd procesu tłumaczenia nazw zawiera sekcja Tabela 114 na stronie 768 . Nazwa podana w pliku *ObjectQMgrName* jest tłumaczona *przed* nazwą podaną w pliku *ObjectName*.

Tabela 114 na stronie 768 przedstawia również sposób użycia lokalnej definicji kolejki zdalnej do zdefiniowania aliasu dla nazwy menedżera kolejek. Dzięki temu można wybrać, która kolejka transmisji jest używana podczas umieszczania komunikatów w kolejce zdalnej. Dzięki temu można na przykład użyć pojedynczej kolejki transmisji dla komunikatów przeznaczonych dla wielu zdalnych menedżerów kolejek.

Aby skorzystać z poniższej tabeli, należy najpierw przeczytać dwie lewe kolumny pod nagłówkiem **Dane wejściowe dla MQOD**i wybrać odpowiednią wielkość liter. Następnie należy przeczytać odpowiedni wiersz, postępując zgodnie z instrukcjami. Postępując zgodnie z instrukcjami w kolumnach **Rozstrzygnięte nazwy** , można wrócić do kolumn **Dane wejściowe do MQOD** i wstawić wartości zgodnie z instrukcjami lub wyjść z tabeli z podanymi wynikami. Na przykład może być wymagane wprowadzenie wartości *ObjectName*.

*Tabela 114. Rozstrzyganie nazw kolejek podczas korzystania z MQOPEN*

| Dane wejściowe dla MQOD | Dane wejściowe dla MQOD | Przetłumaczone nazwy | Przetłumaczone nazwy | Przetłumaczone nazwy |
|---|---|---|---|---|
| *ObjectQMgrName* | *ObjectName* | *ObjectQMgrName* | *ObjectName* | Transmission queue |

*Tabela 114. Rozstrzyganie nazw kolejek podczas korzystania z MQOPEN (kontynuacja)*

| **Dane wejściowe dla MQOD** | **Dane wejściowe dla MQOD** | **Przetłumaczone nazwy** | **Przetłumaczone nazwy** | **Przetłumaczone nazwy** |
|---|---|---|---|---|
| Pusty lub lokalny menedżer kolejek | Kolejka lokalna bez atrybutu CLUSTER | Lokalny menedżer kolejek | Dane wejściowe *ObjectName* | Nie dotyczy (używana kolejka lokalna) |
| Pusty menedżer kolejek | Kolejka lokalna z atrybutem CLUSTER | W operacji PUT wybrano menedżera kolejek klastra lub konkretnego menedżera kolejek klastra. | Dane wejściowe *ObjectName* | SYSTEM SYSTEM.CLUSTER.TRANSMIT.QUEUE i kolejka lokalna<br>SYSTEM SYSTEM.QSG.TRANSMIT.QUEUE (patrz uwaga) |
| Lokalny menedżer kolejek | Kolejka lokalna z atrybutem CLUSTER | Lokalny menedżer kolejek | Dane wejściowe *ObjectName* | Nie dotyczy (używana kolejka lokalna) |
| Pusty lub lokalny menedżer kolejek | Kolejka modelowa | Lokalny menedżer kolejek | Wygenerowana nazwa | Nie dotyczy (używana kolejka lokalna) |
| Pusty lub lokalny menedżer kolejek | Kolejka aliasowa z atrybutem CLUSTER lub bez tego atrybutu | Ponownie przeprowadź tłumaczenie nazw z niezmienionym atrybutem *ObjectQMgrName* i wartością wejściową *ObjectName* ustawioną na *BaseQName* w obiekcie definicji kolejki aliasowej.<br>Nie można go przetłumaczyć na alias zdefiniowany lokalnie, w którym określono nazwę *ObjectQMgr* , ale można go przetłumaczyć na alias klastrowy (udostępniany przez inne menedżery kolejek), w którym pole *ObjectQMgrNazwa* jest puste. | | |
| Lokalny menedżer kolejek | Kolejka aliasowa z atrybutem CLUSTER | Alias nie może być tłumaczony na kolejkę klastra, która nie jest zdefiniowana lokalnie, lub na kolejkę klastra, która ma taką samą *ObjectName* jak alias. | | |

*Tabela 114. Rozstrzyganie nazw kolejek podczas korzystania z MQOPEN (kontynuacja)*

| **Dane wejściowe dla MQOD** | **Dane wejściowe dla MQOD** | **Przetłumaczone nazwy** | **Przetłumaczone nazwy** | **Przetłumaczone nazwy** |
|---|---|---|---|---|
| Pusty menedżer kolejek | Kolejka aliasowa z atrybutem CLUSTER | Alias może zostać przetłumaczony na kolejkę klastra z tym samym *ObjectName* co alias. | | |
| Pusty lub lokalny menedżer kolejek | lokalna definicja kolejki zdalnej | Wykonaj ponownie tłumaczenie nazw z parametrem *ObjectQMgrName* ustawionym na wartość *RemoteQMgrName*i para metrem *ObjectName* ustawionym na wartość *RemoteQName*. Nie można rozstrzygać kolejek zdalnych | | Nazwa atrybutu *XmitQName* , jeśli nie jest określona, w przeciwnym razie *RemoteQMgrNazwa* w obiekcie definicji kolejki zdalnej.<br>SYSTEM SYSTEM.QSG.TRANSMIT.QUEUE (patrz uwaga) |
| Pusty menedżer kolejek | Brak zgodnego obiektu lokalnego; znaleziono kolejkę klastra | W operacji PUT wybrano menedżera kolejek klastra lub konkretnego menedżera kolejek klastra. | Dane wejściowe *ObjectName* | SYSTEM.CLUSTER.TRANSMIT.QUEUE<br>SYSTEM SYSTEM.QSG.TRANSMIT.QUEUE (patrz uwaga) |
| Pusty lub lokalny menedżer kolejek | Brak zgodnego obiektu lokalnego; nie znaleziono kolejki klastra | | Błąd, nie znaleziono kolejki | Nie dotyczy |
| Nazwa menedżera kolejek w tej samej grupie współużytkowania kolejek co menedżer kolejek lokalnych | Lokalna kolejka współużytkowana | Lokalny menedżer kolejek | Dane wejściowe *ObjectName* | Nie dotyczy |
| Nazwa lokalnej kolejki transmisji | (nierozstrzygnięte) | Dane wejściowe *ObjectQMgrNazwa* | Dane wejściowe *ObjectName* | Dane wejściowe *ObjectQMgrNazwa*<br>SYSTEM SYSTEM.QSG.TRANSMIT.QUEUE (patrz uwaga) |

| *Tabela 114. Rozstrzyganie nazw kolejek podczas korzystania z MQOPEN (kontynuacja)* | | | | |
|---|---|---|---|---|
| **Dane wejściowe dla MQOD** | **Dane wejściowe dla MQOD** | **Przetłumaczone nazwy** | **Przetłumaczone nazwy** | **Przetłumaczone nazwy** |
| Definicja aliasu menedżera kolejek (*RemoteQMgrNazwa* może być lokalnym menedżerem kolejek) | (Nie rozstrzygnięto, kolejka zdalna) | Ponownie wykonaj tłumaczenie nazw, jeśli parametr *ObjectQMgrName* ma wartość *RemoteQMgrName*. Nie można przetłumaczyć na kolejki zdalne | Dane wejściowe *ObjectName* | Nazwa atrybutu *XmitQName* , jeśli nie jest określona, w przeciwnym razie *RemoteQMgrNazwa* w obiekcie definicji kolejki zdalnej.<br>SYSTEM SYSTEM.QSG.TRANSMIT.QUEUE (patrz uwaga) |
| Menedżer kolejek nie jest nazwą żadnego obiektu lokalnego; znaleziono menedżery kolejek klastra lub alias menedżera kolejek | (nierozstrzygnięte) | *ObjectQMgrNazwa* lub konkretny menedżer kolejek klastra wybrany w operacji PUT | Dane wejściowe *ObjectName* | SYSTEM.CLUSTER.TRANSMIT.QUEUE<br>SYSTEM SYSTEM.QSG.TRANSMIT.QUEUE (patrz uwaga) |
| Menedżer kolejek nie jest nazwą żadnego obiektu lokalnego; nie znaleziono obiektów klastra | (nierozstrzygnięte) | Dane wejściowe *ObjectQMgrNazwa* | Dane wejściowe *ObjectName* | *DefXmitQName* atrybut menedżera kolejek, w którym obsługiwana jest nazwa QName *DefXmit* .<br>SYSTEM SYSTEM.QSG.TRANSMIT.QUEUE (patrz uwaga) |

**Uwagi:**

1. *BaseQName* jest nazwą kolejki podstawowej z definicji kolejki aliasowej.
2. *RemoteQName* jest nazwą kolejki zdalnej z lokalnej definicji kolejki zdalnej.
3. *RemoteQMgrName* jest nazwą menedżera kolejek zdalnych z lokalnej definicji kolejki zdalnej.
4. *XmitQName* jest nazwą kolejki transmisji z lokalnej definicji kolejki zdalnej.
5. W przypadku używania menedżerów kolejek systemu IBM MQ for z/OS , które są częścią grupy współużytkowania kolejek (QSG), w programie Tabela 114 na stronie 768można użyć nazwy grupy współużytkowania kolejek zamiast nazwy lokalnego menedżera kolejek.

   Jeśli lokalny menedżer kolejek nie może otworzyć kolejki docelowej lub umieścić komunikatu w kolejce, komunikat jest przesyłany do określonej nazwy ObjectQMgrza pośrednictwem kolejki wewnątrz grupy lub kanału IBM MQ .
6. W kolumnie *ObjectName* tabeli CLUSTER odnosi się zarówno do atrybutów CLUSTER, jak i CLUSNL kolejki.
7. SYSTEM SYSTEM.QSG.TRANSMIT.QUEUE jest używany, jeśli lokalne i zdalne menedżery kolejek znajdują się w tej samej grupie współużytkowania kolejek. Kolejkowanie wewnątrz grupy jest włączone.
8. Jeśli do każdego kanału nadawczego klastra przypisano inną kolejkę transmisji klastra, SYSTEM.CLUSTER.TRANSMIT.QUEUE może nie być nazwą kolejki transmisji klastra. Więcej informacji na temat wielu kolejek transmisji klastra zawiera sekcja Grupowanie: Planowanie konfigurowania kolejek transmisji klastra.

9. W sytuacji, gdy menedżer kolejek nie jest nazwą żadnego obiektu lokalnego, znaleziono menedżery kolejek klastra lub alias menedżera kolejek.

   Jeśli nazwa menedżera kolejek została podana przy użyciu parametru **ObjectQMgrName**, a istnieje wiele kanałów klastra z różnymi nazwami klastrów znanymi lokalnemu menedżerowi kolejek, które mogłyby osiągnąć to miejsce docelowe, to każdy z tych kanałów może zostać użyty do przeniesienia komunikatu bez względu na nazwę klastra kolejki docelowej.

   Może to być nieoczekiwane, jeśli użytkownik spodziewał się, że komunikaty dla tej kolejki będą wysyłane tylko przez kanał, który ma taką samą nazwę klastra jak kolejka.

   Jednak w tym przypadku pierwszeństwo ma parametr **ObjectQMgrName** , a funkcja równoważenia obciążenia klastra uwzględnia wszystkie kanały, które mogą dotrzeć do tego menedżera kolejek, niezależnie od nazwy klastra, w którym się znajdują.

Otwarcie kolejki aliasowej powoduje również otwarcie kolejki podstawowej, do której alias jest tłumaczony, a otwarcie kolejki zdalnej powoduje również otwarcie kolejki transmisji. Dlatego nie można usunąć określonej kolejki ani kolejki, do której jest ona rozstrzygana, gdy druga kolejka jest otwarta.

Podczas gdy kolejka aliasowa nie może zostać przetłumaczona na inną lokalnie zdefiniowaną kolejkę aliasową (współużytkowaną w klastrze lub nie), rozstrzyganie na zdalnie zdefiniowaną kolejkę aliasową klastra jest dozwolone i dlatego można ją określić jako kolejkę podstawową.

Rozstrzygnięta nazwa kolejki i rozstrzygnięta nazwa menedżera kolejek są przechowywane w polach *ResolvedQName* i *ResolvedQMgrName* w programie MQOD.

Więcej informacji na temat tłumaczenia nazw w środowisku kolejkowania rozproszonego zawiera sekcja Czym jest tłumaczenie nazw kolejek?.

*Korzystanie z opcji wywołania MQOPEN*

W parametrze **Options** wywołania MQOPEN należy wybrać co najmniej jedną opcję, aby kontrolować dostęp do otwieranego obiektu. Za pomocą tych opcji można:

- Otwórz kolejkę i określ, że wszystkie komunikaty umieszczone w tej kolejce muszą być kierowane do tej samej instancji tej kolejki.
- Otwórz kolejkę, aby umożliwić umieszczanie w niej komunikatów
- Otwórz kolejkę, aby umożliwić przeglądanie znajdujących się w niej komunikatów
- Otwórz kolejkę, aby umożliwić usuwanie z niej komunikatów
- Otwórz obiekt, aby umożliwić sprawdzenie i ustawienie jego atrybutów (ale można ustawić tylko atrybuty kolejek)
- Otwórz temat lub łańcuch tematu, aby opublikować w nim komunikaty
- Powiąż informacje o kontekście z komunikatem
- Wyznacz alternatywny identyfikator użytkownika, który będzie używany do sprawdzania bezpieczeństwa
- Steruj wywołaniem, jeśli menedżer kolejek jest w stanie wyciszania

*Opcja MQOPEN dla kolejki klastra*
Powiązanie używane dla uchwytu kolejki jest pobierane z atrybutu kolejki **DefBind** , który może mieć wartość MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXEDlub MQBND_BIND_ON_GROUP.

Aby skierować wszystkie komunikaty umieszczone w kolejce przy użyciu funkcji MQPUT do tego samego menedżera kolejek przy użyciu tej samej trasy, należy użyć opcji MQOO_BIND_ON_OPEN w wywołaniu funkcji MQOPEN .

Aby określić, że miejsce docelowe ma zostać wybrane w czasie MQPUT, czyli dla poszczególnych komunikatów, należy użyć opcji MQOO_BIND_NOT_FIXED w wywołaniu MQOPEN .

Aby określić, że wszystkie komunikaty w grupach komunikatów umieszczane w kolejce za pomocą funkcji MQPUT są przydzielane do tej samej instancji docelowej, należy użyć opcji MQOO_BIND_ON_GROUP w wywołaniu funkcji MQOPEN .

W przypadku korzystania z grup komunikatów z klastrami należy określić parametr MQOO_BIND_ON_OPEN lub MQOO_BIND_ON_GROUP , aby zapewnić, że wszystkie komunikaty w grupie będą przetwarzane w tym samym miejscu docelowym.

Jeśli żadna z tych opcji nie zostanie podana, zostanie użyta wartość domyślna MQOO_BIND_AS_Q_DEF.

Jeśli nazwa menedżera kolejek zostanie określona w programie MQOD, wybrana zostanie kolejka w tym menedżerze kolejek. Jeśli nazwa menedżera kolejek jest pusta, można wybrać dowolną instancję. Więcej informacji zawiera sekcja “MQOPEN i klastry” na stronie 912.

Jeśli kolejka klastra jest otwierana za pomocą definicji QALIAS , niektóre atrybuty kolejki są definiowane przez kolejkę aliasową, a nie przez kolejkę podstawową. Atrybuty klastra należą do atrybutów definicji kolejki podstawowej, które są nadpisywane przez kolejkę aliasową. Na przykład w poniższym fragmencie kodu kolejka klastra jest otwierana za pomocą kodu MQOO_BIND_NOT_FIXED , a nie za pomocą kodu MQOO_BIND_ON_OPEN. Definicja kolejki klastra jest ogłaszana w całym klastrze, a definicja kolejki aliasowej jest lokalna względem menedżera kolejek.

```
DEFINE QLOCAL(CLQ1) CLUSTER(MYCLUSTER) DEFBIND(OPEN) REPLACE
DEFINE QALIAS(ACLQ1) TARGET(CLQ1) DEFBIND(NOTFIXED) REPLACE
```

*Opcja MQOPEN do umieszczania komunikatów*
Aby otworzyć kolejkę lub temat w celu umieszczenia w nim komunikatów, należy użyć opcji MQOO_OUTPUT.

*Opcja MQOPEN do przeglądania komunikatów*
Aby otworzyć kolejkę w celu *przeglądania* jej komunikatów, należy użyć wywołania MQOPEN z opcją MQOO_BROWSE.

Spowoduje to utworzenie *kursora przeglądania* , którego menedżer kolejek używa do identyfikowania następnego komunikatu w kolejce. Więcej informacji na ten temat zawiera sekcja “Przeglądanie komunikatów w kolejce” na stronie 828.

**Uwaga:**

1. Nie można przeglądać komunikatów w kolejce zdalnej; nie należy otwierać kolejki zdalnej za pomocą opcji MQOO_BROWSE.
2. Nie można określić tej opcji podczas otwierania listy dystrybucyjnej. Więcej informacji na temat list dystrybucyjnych zawiera sekcja “Lista dystrybucyjna” na stronie 787.
3. W przypadku korzystania z przeglądania kooperatywnego należy użyć opcji MQOO_CO_OP w połączeniu z opcją MQOO_BROWSE. Więcej informacji na ten temat zawiera sekcja Opcje .

*Opcje MQOPEN dotyczące usuwania komunikatów*
Trzy opcje sterują otwieraniem kolejki w celu usunięcia z niej komunikatów.

W każdym wywołaniu MQOPEN można użyć tylko jednego z nich. Opcje te określają, czy program ma wyłączny, czy współużytkowany dostęp do kolejki. *Dostęp na wyłączność* oznacza, że do momentu zamknięcia kolejki tylko Ty możesz usunąć z niego wiadomości. Jeśli inny program spróbuje otworzyć kolejkę w celu usunięcia komunikatów, wywołanie MQOPEN nie powiedzie się. *Dostęp współużytkowany* oznacza, że więcej niż jeden program może usunąć komunikatów z kolejki.

Najbardziej zaleconym podejściem jest zaakceptowanie typu dostępu, który był przeznaczony dla kolejki podczas jej definiowania. Definicja kolejki obejmowała ustawienie parametru **Shareability** i parametru Atrybuty **DefInputOpenOption** . Aby zaakceptować ten dostęp, należy użyć opcji MQOO_INPUT_AS_Q_DEF. Patrz sekcja Tabela 115 na stronie 773 , aby dowiedzieć się, w jaki sposób ustawienie tych atrybutów wpływa na typ dostępu, który zostanie nadany podczas korzystania z tej opcji.

*Tabela 115. Wpływ atrybutów i opcji kolejki wywołania MQOPEN na dostęp do kolejek*

| **Kolejka - atrybuty** | | **Typ dostępu z opcjami MQOPEN** | | |
|---|---|---|---|---|
| **Shareability** | **DefInputOpenOption** | **AS_Q_DEF** | **WSPÓŁUŻYTKOWANY** | **EXCLUSIVE** |

| *Tabela 115. Wpływ atrybutów i opcji kolejki wywołania MQOPEN na dostęp do kolejek (kontynuacja)* | | | | |
|---|---|---|---|---|
| **Kolejka - atrybuty** | | **Typ dostępu z opcjami MQOPEN** | | |
| Do współużytkowania | WSPÓŁUŻYTKOWANY | współużytkowany | współużytkowany | wykluczająca |
| Do współużytkowania | EXCLUSIVE | wykluczająca | współużytkowany | wykluczająca |
| NIESHAREABLE* | WSPÓŁUŻYTKOWANIE * | wykluczająca | wykluczająca | wykluczająca |
| NIESHAREABLE | EXCLUSIVE | wykluczająca | wykluczająca | wykluczająca |

**Uwaga:** * Chociaż można zdefiniować kolejkę tak, aby miała taką kombinację atrybutów, domyślna opcja otwarcia wejścia jest nadpisywana przez atrybut współużytkowalności.

Lub:

- Jeśli wiadomo, że aplikacja może działać pomyślnie, nawet jeśli inne programy mogą usuwać komunikaty z kolejki w tym samym czasie, należy użyć opcji MQOO_INPUT_SHARED. Tabela 115 na stronie 773 pokazuje, w jaki sposób, w niektórych przypadkach, użytkownik będzie miał wyłączny dostęp do kolejki, nawet z tą opcją.
- Jeśli wiadomo, że aplikacja może działać poprawnie tylko wtedy, gdy inne programy nie mogą jednocześnie usuwać komunikatów z kolejki, należy użyć opcji MQOO_INPUT_EXCLUSIVE.

**Uwaga:**

1. Nie można usuwać komunikatów ze zdalnej kolejki. Dlatego nie można otworzyć kolejki zdalnej przy użyciu żadnej z opcji MQOO_INPUT_ *.
2. Nie można określić tej opcji podczas otwierania listy dystrybucyjnej. Więcej informacji zawiera sekcja "Lista dystrybucyjna" na stronie 787.

*Opcje MQOPEN służące do ustawiania atrybutów i uzyskiwania informacji o nich*
Aby otworzyć kolejkę w celu ustawienia jej atrybutów, należy użyć opcji MQOO_SET.

Nie można ustawić atrybutów żadnego innego typu obiektu (patrz sekcja "Uzyskiwanie informacji o atrybutach obiektu i ustawianie ich" na stronie 876 ).

Aby otworzyć obiekt w celu uzyskania informacji o jego atrybutach, należy użyć opcji MQOO_INQUIRE.

**Uwaga:** Nie można określić tej opcji podczas otwierania listy dystrybucyjnej.

*Opcje MQOPEN dotyczące kontekstu komunikatu*
Aby można było powiązać informacje o kontekście z komunikatem umieszczonym w kolejce, podczas otwierania kolejki należy użyć jednej z opcji kontekstu komunikatu.

Opcje umożliwiają rozróżnienie między informacjami o kontekście, które odnoszą się do *użytkownika* , który wygenerował komunikat, a tymi, które odnoszą się do *aplikacji* , która wygenerowała komunikat. Można również ustawić informacje o kontekście podczas umieszczania komunikatu w kolejce lub wybrać opcję automatycznego zabierania kontekstu z innego uchwytu kolejki.

**Pojęcia pokrewne**
"Kontekst komunikatu" na stronie 48
Informacje *Kontekst komunikatu* umożliwiają aplikacji, która pobiera komunikat, uzyskanie informacji o twórcy komunikatu.

"Sterowanie informacjami o kontekście komunikatu" na stronie 784
Jeśli do umieszczenia komunikatu w kolejce używane jest wywołanie MQPUT lub MQPUT1 , można określić, że menedżer kolejek ma dodać do deskryptora komunikatu pewne domyślne informacje o kontekście. Aplikacje, które mają odpowiedni poziom uprawnień, mogą dodawać dodatkowe informacje o kontekście. Do sterowania informacjami o kontekście można użyć pola opcji w strukturze MQPMO.

*Opcja MQOPEN dla alternatywnego uprawnienia użytkownika*
Podczas próby otwarcia obiektu za pomocą wywołania MQOPEN menedżer kolejek sprawdza, czy użytkownik ma uprawnienia do otwarcia tego obiektu. Jeśli użytkownik nie jest autoryzowany, wywołanie nie powiedzie się.

Jednak programy serwera mogą chcieć, aby menedżer kolejek sprawdził autoryzację użytkownika, dla którego pracuje, a nie własną autoryzację serwera. W tym celu należy użyć opcji MQOO_ALTERNATE_USER_AUTHORITY wywołania MQOPEN i określić alternatywny identyfikator użytkownika w polu *AlternateUserId* struktury MQOD. Zwykle serwer otrzymuje identyfikator użytkownika z informacji o kontekście w przetwarzanym komunikacie.

z/OS *Opcja MQOPEN dla wyciszania menedżera kolejek*
Jeśli używane jest wywołanie MQOPEN, gdy menedżer kolejek jest w stanie wyciszania, wywołanie może zakończyć się niepowodzeniem w zależności od używanego środowiska.

W środowisku CICS w systemie z/OS, jeśli używane jest wywołanie MQOPEN, gdy menedżer kolejek jest w stanie wyciszania, wywołanie zawsze kończy się niepowodzeniem.

W innych środowiskach z/OS i wieloplatformowych wywołanie kończy się niepowodzeniem, gdy menedżer kolejek jest wyciszany tylko wtedy, gdy używana jest opcja MQOO_FAIL_IF_QUIESCING wywołania MQOPEN.

*Opcja MQOPEN do tłumaczenia nazw kolejek lokalnych*
Po otwarciu kolejki lokalnej, kolejki aliasowej lub kolejki modelowej zwracana jest kolejka lokalna.

Jednak po otwarciu kolejki zdalnej lub kolejki klastra pola *ResolvedQName* i *ResolvedQMgrName* struktury MQOD są wypełniane nazwami kolejki zdalnej i menedżera kolejek zdalnych znalezionymi w definicji kolejki zdalnej lub z wybraną zdalną kolejką klastra.

Użyj opcji MQOO_RESOLVE_LOCAL_Q wywołania MQOPEN, aby wypełnić pole *ResolvedQName* w strukturze MQOD nazwą kolejki lokalnej, która została otwarta. Pole *ResolvedQMgrName* jest podobnie wypełnione nazwą lokalnego menedżera kolejek, który udostępnia kolejkę lokalną. To pole jest dostępne tylko w wersji 3 struktury MQOD. Jeśli struktura jest niższa niż wersja 3, opcja MQOO_RESOLVE_LOCAL_Q jest ignorowana bez zwracania błędu.

Jeśli podczas otwierania kolejki, na przykład kolejki zdalnej, zostanie podana wartość MQOO_RESOLVE_LOCAL_Q, *ResolvedQName* jest nazwą kolejki transmisji, do której będą umieszczane komunikaty. *ResolvedQMgrName* to nazwa lokalnego menedżera kolejek udostępniającego kolejkę transmisji.

## *Tworzenie kolejek dynamicznych*
Kolejki dynamicznej należy używać wtedy, gdy nie jest ona potrzebna po zakończeniu działania aplikacji.

Na przykład można użyć kolejki dynamicznej dla kolejki odpowiedzi. Nazwę kolejki zwrotnej określa się w polu *ReplyToQ* struktury MQMD podczas umieszczania komunikatu w kolejce (patrz sekcja “Definiowanie komunikatów przy użyciu struktury MQMD” na stronie 779 ).

Aby utworzyć kolejkę dynamiczną, należy użyć szablonu znanego jako kolejka modelowa wraz z wywołaniem MQOPEN. Kolejkę modelową tworzy się za pomocą komend IBM MQ lub za pomocą operacji i paneli sterujących. Tworzona kolejka dynamiczna przyjmuje atrybuty kolejki modelowej.

Podczas wywoływania funkcji MQOPEN należy określić nazwę kolejki modelowej w polu *ObjectName* struktury MQOD. Po zakończeniu wywołania w polu *ObjectName* zostanie ustawiona nazwa tworzonej kolejki dynamicznej. Ponadto w polu *ObjectQMgrName* jest ustawiona nazwa lokalnego menedżera kolejek.

Nazwę tworzonej kolejki dynamicznej można określić na trzy sposoby:

- Podaj pełną nazwę w polu *DynamicQName* struktury MQOD.
- Podaj przedrostek (mniej niż 33 znaki) dla nazwy i zezwól menedżerowi kolejek na wygenerowanie pozostałej części nazwy. Oznacza to, że menedżer kolejek generuje unikalną nazwę, ale nadal ma pewne sterowanie (na przykład każdy użytkownik może używać określonego przedrostka lub może chcieć

nadać specjalną klasyfikację zabezpieczeń kolejkom z określonym przedrostkiem w nazwie). Aby użyć tej metody, należy wpisać gwiazdkę (*) jako ostatni niepusty znak w polu *DynamicQName* . Nie należy podawać pojedynczej gwiazdki (*) dla nazwy kolejki dynamicznej.

- Zezwalaj menedżerowi kolejek na generowanie pełnej nazwy. Aby użyć tej metody, należy podać gwiazdkę (*) na pierwszej pozycji znaku w polu *DynamicQName* .

Więcej informacji na temat tych metod zawiera opis pola DynamicQName .

Więcej informacji na temat kolejek dynamicznych zawiera sekcja Kolejki dynamiczne i modelowe.

### *Otwieranie kolejek zdalnych*

Kolejka zdalna to kolejka, której właścicielem jest menedżer kolejek inny niż ten, z którym połączona jest aplikacja.

Aby otworzyć kolejkę zdalną, należy użyć wywołania MQOPEN dla kolejki lokalnej. Nazwę kolejki można określić w następujący sposób:

1. W polu *ObjectName* struktury MQOD określ nazwę kolejki zdalnej, która jest znana w *lokalnym* menedżerze kolejek.

   **Uwaga:** W tym przypadku pole *ObjectQMgrName* należy pozostawić puste.
2. W polu *ObjectName* struktury MQOD określ nazwę kolejki zdalnej, która jest znana w *zdalnym* menedżerze kolejek. W polu *ObjectQMgrName* podaj jedną z następujących wartości:

   - Nazwa kolejki transmisji, która ma taką samą nazwę jak zdalny menedżer kolejek. Nazwa i wielkość liter (wielkie litery, małe litery lub mieszanka) muszą być *dokładnie*zgodne.
   - Nazwa obiektu aliasu menedżera kolejek, który jest tłumaczony na docelowy menedżer kolejek lub kolejkę transmisji.

   W ten sposób menedżer kolejek jest informowany o miejscu docelowym komunikatu, a także o kolejce transmisji, w której musi on zostać umieszczony, aby można było się tam dostać.
3. Jeśli parametr *DefXmitQname* jest obsługiwany, w polu *ObjectName* struktury MQOD należy określić nazwę kolejki zdalnej, która jest znana menedżerowi kolejek *zdalnych* .

   **Uwaga:** W polu *ObjectQMgrName* wpisz nazwę zdalnego menedżera kolejek (w tym przypadku nie można pozostawić pustego pola).

Podczas wywoływania komendy MQOPEN sprawdzana jest poprawność tylko nazw lokalnych; ostatnie sprawdzenie dotyczy istnienia kolejki transmisji, która ma być używana.

Tabela 114 na stronie 768zawiera podsumowanie tych metod.

### *Zamykanie obiektów przy użyciu wywołania MQCLOSE*

Aby zamknąć obiekt, należy użyć wywołania MQCLOSE.

Jeśli obiekt jest kolejką, należy zwrócić uwagę na następujące informacje:

- Przed zamknięciem tymczasowej kolejki dynamicznej nie trzeba jej opróżniać.

  Po zamknięciu tymczasowej kolejki dynamicznej jest ona usuwana wraz z komunikatami, które mogą się w niej znajdować. Ma to miejsce nawet wtedy, gdy dla kolejki istnieją niezatwierdzone wywołania MQGET, MQPUT lub MQPUT1 .
- W systemie IBM MQ for z/OS, jeśli istnieją jakiekolwiek żądania MQGET z opcją MQGMO_SET_SIGNAL oczekujące dla tej kolejki, zostaną one anulowane.
- Jeśli kolejka została otwarta za pomocą opcji MQOO_BROWSE, kursor przeglądania zostanie zniszczony.

Zamknięcie nie jest związane z punktem synchronizacji, dlatego można zamykać kolejki przed punktem synchronizacji lub po nim.

Jako dane wejściowe wywołania MQCLOSE należy podać:

- Uchwyt połączenia. Użyj tego samego uchwytu połączenia, który został użyty do jego otwarcia, lub w przypadku aplikacji CICS w systemie z/OSmożna określić stałą MQHC_DEF_HCONN (która ma wartość zero).
- Uchwyt obiektu, który ma zostać zamknięty. Pobierz to z danych wyjściowych wywołania MQOPEN.
- MQCO_NONE w polu *Options* (chyba że zamykasz trwałą kolejkę dynamiczną).
- Opcja sterująca służąca do określania, czy menedżer kolejek powinien usunąć kolejkę, nawet jeśli nadal są w niej komunikaty (podczas zamykania trwałej kolejki dynamicznej).

Dane wyjściowe komendy MQCLOSE to:

- Kod zakończenia
- Kod przyczyny
- Uchwyt obiektu, zresetowany do wartości MQHO_UNUSABLE_HOBJ

Opisy parametrów wywołania MQCLOSE znajdują się w sekcji MQCLOSE.

## Umieszczanie komunikatów w kolejce

Ta sekcja zawiera informacje na temat umieszczania komunikatów w kolejce.

Użyj wywołania MQPUT, aby umieścić komunikaty w kolejce. Za pomocą wywołania MQPUT można wielokrotnie umieszczać wiele komunikatów w tej samej kolejce po początkowym wywołaniu MQOPEN. Po zakończeniu umieszczania wszystkich komunikatów w kolejce wywołaj komendę MQCLOSE.

Aby umieścić pojedynczy komunikat w kolejce i zamknąć kolejkę natychmiast po zakończeniu, można użyć wywołania MQPUT1 . Komenda MQPUT1 wykonuje te same funkcje, co następująca sekwencja wywołań:

- MQOPEN
- MQPUT
- MQCLOSE

Jeśli jednak istnieje więcej niż jeden komunikat do umieszczenia w kolejce, bardziej wydajne jest użycie wywołania MQPUT. Zależy to od wielkości komunikatu i platformy, na której pracuje użytkownik.

Więcej informacji na temat umieszczania komunikatów w kolejce można uzyskać, korzystając z następujących odsyłaczy:

- “Umieszczanie komunikatów w kolejce lokalnej przy użyciu wywołania MQPUT” na stronie 778
- “Umieszczanie komunikatów w kolejce zdalnej” na stronie 783
- “Ustawianie właściwości komunikatu” na stronie 783
- “Sterowanie informacjami o kontekście komunikatu” na stronie 784
- “Umieszczanie jednego komunikatu w kolejce za pomocą wywołania MQPUT1” na stronie 786
- “Lista dystrybucyjna” na stronie 787
- “Niektóre przypadki, w których wywołania put nie powiodły się” na stronie 792

**Pojęcia pokrewne**

“Przegląd interfejsu kolejki komunikatów” na stronie 745
Sekcja zawiera informacje na temat komponentów interfejsu kolejki komunikatów (Message Queue Interface-MQI).

“Nawiązywanie i rozłączanie połączenia z menedżerem kolejek” na stronie 758
Aby można było używać usług programistycznych IBM MQ , program musi mieć połączenie z menedżerem kolejek. Ten temat zawiera informacje o nawiązywaniu połączenia z menedżerem kolejek i rozłączaniu się z nim.

“Otwieranie i zamykanie obiektów” na stronie 766
Informacje te udostępniają wgląd w otwieranie i zamykanie obiektów IBM MQ .

“Pobieranie komunikatów z kolejki” na stronie 792
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Uzyskiwanie informacji o atrybutach obiektu i ustawianie ich” na stronie 876
Atrybuty są właściwościami definiującymi charakterystykę obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 879
W tej sekcji opisano sposób zatwierdzania i wycofywania wszelkich odtwarzalnych operacji pobierania i umieszczania, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy” na stronie 891
Informacje o wyzwalaczach i sposobie uruchamiania aplikacji IBM MQ za pomocą wyzwalaczy.

“Praca z funkcją MQI i klastrami” na stronie 911
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do łączenia w klastry.

“Używanie i pisanie aplikacji w systemie IBM MQ for z/OS” na stronie 916
Aplikacje IBM MQ for z/OS mogą być tworzone z programów działających w wielu różnych środowiskach. Oznacza to, że mogą korzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje mostu IMS i IMS w systemie IBM MQ for z/OS” na stronie 71
Informacje te są pomocne podczas pisania aplikacji IMS przy użyciu produktu IBM MQ.

## *Umieszczanie komunikatów w kolejce lokalnej przy użyciu wywołania MQPUT*

Ten temat zawiera informacje o umieszczaniu komunikatów w kolejce lokalnej przy użyciu wywołania MQPUT.

Jako dane wejściowe wywołania MQPUT należy podać:

- Uchwyt połączenia (Hconn).
- Uchwyt kolejki (Hobj).
- Opis komunikatu, który ma zostać umieszczony w kolejce. Jest to struktura deskryptora komunikatu (MQMD).
- Informacje sterujące w postaci struktury opcji umieszczania komunikatu (MQPMO).
- Długość danych zawartych w komunikacie (MQLONG).
- Same dane komunikatu.

Dane wyjściowe wywołania MQPUT są następujące:

- Kod przyczyny (MQLONG)
- Kod zakończenia (MQLONG)

Jeśli wywołanie zakończy się pomyślnie, zwróci również strukturę opcji i strukturę deskryptora komunikatu. Wywołanie modyfikuje strukturę opcji, wyświetlając nazwę kolejki i menedżera kolejek, do którego został wysłany komunikat. W przypadku żądania wygenerowania przez menedżer kolejek unikalnej wartości dla identyfikatora umieszczanego komunikatu (przez określenie zera binarnego w polu *MsgId* struktury MQMD), wywołanie wstawia wartość w polu *MsgId* przed zwróceniem tej struktury do użytkownika. Tę wartość należy zresetować przed wysłaniem kolejnego wywołania MQPUT.

Opis wywołania MQPUT znajduje się w sekcji MQPUT.

Więcej informacji na temat informacji wymaganych jako dane wejściowe wywołania MQPUT zawierają następujące odsyłacze:

- “Określanie uchwytów” na stronie 779
- “Definiowanie komunikatów przy użyciu struktury MQMD” na stronie 779
- “Określanie opcji przy użyciu struktury MQPMO” na stronie 779
- “Dane w komunikacie” na stronie 782
- “Umieszczanie komunikatów: korzystanie z uchwytów komunikatów” na stronie 783

## Określanie uchwytów

W przypadku uchwytu połączenia (*Hconn*) w produkcie CICS w aplikacjach z/OS można określić stałą MQHC_DEF_HCONN (która ma wartość zero) lub użyć uchwytu połączenia zwróconego przez wywołanie MQCONN lub MQCONNX. W przypadku innych aplikacji należy zawsze używać uchwytu połączenia zwróconego przez wywołanie MQCONN lub MQCONNX.

Niezależnie od używanego środowiska należy użyć tego samego uchwytu kolejki (*Hobj*), który jest zwracany przez wywołanie MQOPEN.

## Definiowanie komunikatów przy użyciu struktury MQMD

Struktura deskryptora komunikatu (MQMD) jest parametrem wejścia/wyjścia dla wywołań MQPUT i MQPUT1 . Służy on do definiowania komunikatu umieszczanego w kolejce.

Jeśli dla komunikatu określono opcję MQPRI_PRIORITY_AS_Q_DEF lub MQPER_PERSISTENCE_AS_Q_DEF, a kolejka jest kolejką klastra, używane są wartości kolejki, do której jest rozstrzygana operacja MQPUT. Jeśli ta kolejka jest wyłączona dla MQPUT, wywołanie nie powiedzie się. Więcej informacji na ten temat zawiera sekcja Konfigurowanie klastra menedżera kolejek .

**Uwaga:** Użyj opcji MQPMO_NEW_MSG_ID i MQPMO_NEW_CORREL_ID przed umieszczeniem nowego komunikatu, aby upewnić się, że *MsgId* i *CorrelId* są unikalne. Wartości w tych polach są zwracane po pomyślnym wykonaniu operacji MQPUT.

Istnieje wprowadzenie do właściwości komunikatu, które MQMD opisuje w “Komunikaty produktu IBM MQ” na stronie 18, oraz opis samej struktury w MQMD.

## Określanie opcji przy użyciu struktury MQPMO

Struktura MQPMO (Put Message Option) służy do przekazywania opcji do wywołań MQPUT i MQPUT1 .

Poniższe sekcje zawierają informacje pomocne przy wypełnianiu pól tej struktury. Istnieje opis struktury w MQPMO.

Struktura obejmuje następujące pola:

- *StrucId*
- *Version*
- *Options*
- *Context*
- *ResolvedQName*
- *ResolvedQMgrName*
- *RecsPresent*
- *PutMsgRecsFields*
- *ResponseRecOffset and ResponseRecPtr*
- *OriginalMsgHandle*
- *NewMsgHandle*
- *Action*
- *PubLevel*

Zawartość tych pól jest następująca:

**StrucId**
Identyfikuje strukturę jako strukturę opcji put-message. Jest to pole 4-znakowe. Zawsze podawaj wartość MQPMO_STRUC_ID.

**Wersja**
Opisuje numer wersji struktury. Wartość domyślna to MQPMO_VERSION_1. Jeśli zostanie wprowadzona wartość MQPMO_VERSION_2, można użyć list dystrybucyjnych (patrz “Lista

dystrybucyjna” na stronie 787 ). W przypadku wprowadzenia wartości MQPMO_VERSION_3można użyć uchwytów komunikatów i właściwości komunikatów. Jeśli zostanie wprowadzona wartość MQPMO_CURRENT_VERSION, aplikacja będzie zawsze używać najnowszego poziomu.

**Opcje**
Steruje to następującymi elementami:

- Informacja o tym, czy operacja umieszczania (put) jest uwzględniona w jednostce pracy
- Ilość informacji o kontekście powiązanych z komunikatem
- Miejsce, z którego pobierane są informacje kontekstowe
- Określa, czy wywołanie nie powiedzie się, jeśli menedżer kolejek jest w stanie wyciszania.
- Określa, czy grupowanie lub segmentacja jest dozwolona
- Generowanie nowego identyfikatora komunikatu i identyfikatora korelacji
- Kolejność umieszczania komunikatów i segmentów w kolejce
- Czy rozstrzygać nazwy kolejek lokalnych

Jeśli pole *Options* pozostanie ustawione na wartość domyślną (MQPMO_NONE), z umieszczonym komunikatem będą powiązane domyślne informacje o kontekście.

Ponadto sposób działania wywołania z punktami synchronizacji jest określany przez platformę. Wartością domyślną elementu sterującego punktu synchronizacji w produkcie z/OS jest yes. W przypadku innych platform jest to wartość "nie".

**Kontekst**
Określa nazwę uchwytu kolejki, z którego mają być kopiowane informacje o kontekście (jeśli jest to wymagane w polu *Options* ).

Wprowadzenie do kontekstu komunikatu zawiera sekcja “Kontekst komunikatu” na stronie 48. Informacje na temat używania struktury MQPMO do sterowania informacjami o kontekście w komunikacie zawiera sekcja “Sterowanie informacjami o kontekście komunikatu” na stronie 784.

**ResolvedQName**
Zawiera nazwę (po przetłumaczeniu dowolnej nazwy aliasu) kolejki, która została otwarta w celu odebrania komunikatu. Jest to pole wyjściowe.

**ResolvedQMgrNazwa**
Zawiera on nazwę (po przetłumaczeniu dowolnej nazwy aliasu) menedżera kolejek, który jest właścicielem kolejki w programie *ResolvedQName*. Jest to pole wyjściowe.

Program MQPMO może również pomieścić pola wymagane dla list dystrybucyjnych (patrz sekcja “Lista dystrybucyjna” na stronie 787 ). Jeśli ma być używane to narzędzie, używana jest wersja 2 struktury MQPMO. Obejmuje to następujące pola:

**RecsPresent**
To pole zawiera liczbę kolejek na liście dystrybucyjnej, czyli liczbę rekordów umieszczania komunikatów (MQPMR) i odpowiadających im rekordów odpowiedzi (MQRR).

Wprowadzona wartość może być taka sama jak liczba rekordów obiektu podana w MQOPEN. Jeśli jednak wartość jest mniejsza niż liczba rekordów obiektów podana w wywołaniu MQOPEN lub jeśli nie zostanie podana opcja Umieść rekordy komunikatów, wartości kolejek, które nie zostały zdefiniowane, są pobierane z wartości domyślnych podanych przez deskryptor komunikatu. Ponadto, jeśli wartość jest większa niż podana liczba rekordów obiektów, nadmiarowe rekordy komunikatów umieszczania (Put Message Records) są ignorowane.

Zaleca się wykonanie jednej z następujących czynności:

- Aby otrzymać raport lub odpowiedź z każdego miejsca docelowego, należy wprowadzić tę samą wartość, która jest wyświetlana w strukturze MQOR, i użyć MQPMR zawierających pola *MsgId* . Zainicjuj te pola *MsgId* na zera lub podaj wartość MQPMO_NEW_MSG_ID.

  Po umieszczeniu komunikatu w kolejce wartości *MsgId* utworzone przez menedżer kolejek stają się dostępne w MQPMR. Można ich użyć do określenia, które miejsce docelowe jest powiązane z każdym raportem lub odpowiedzią.

- Jeśli nie chcesz otrzymywać raportów ani odpowiedzi, wybierz jedną z następujących opcji:
  1. Aby zidentyfikować miejsca docelowe, które nie powiodły się natychmiast, można wprowadzić tę samą wartość w polu *RecsPresent* , która jest wyświetlana w strukturze MQOR, i udostępnić wywołania MQRR w celu zidentyfikowania tych miejsc docelowych. Nie podawaj żadnych MQPMR.
  2. Aby nie identyfikować miejsc docelowych, których przetwarzanie nie powiodło się, należy wprowadzić wartość zero w polu *RecsPresent* i nie podawać MQPMR ani MQRR.

  **Uwaga:** Jeśli używana jest komenda MQPUT1, liczba wskaźników rekordów odpowiedzi i przesunięć rekordów odpowiedzi musi wynosić zero.

  Pełny opis opcji Put Message Records (MQPMR) i Response Records (MQRR) zawiera sekcja MQPMR i MQRR.

**PutMsgRecFields**
Wskazuje, które pola są obecne w każdym rekordzie umieszczania komunikatu (MQPMR). Listę tych pól zawiera sekcja "Korzystanie ze struktury MQPMR" na stronie 791.

***PutMsgRecOffset* i *PutMsgRecPtr***
Wskaźniki (zwykle w języku C) i przesunięcia (zazwyczaj w języku COBOL) są używane do adresowania rekordów umieszczania komunikatów (przegląd struktury MQPMR zawiera sekcja "Korzystanie ze struktury MQPMR" na stronie 791 ).

Użyj pola *PutMsgRecPtr* , aby określić wskaźnik do pierwszego rekordu umieszczenia komunikatu lub pola *PutMsgRecOffset* , aby określić przesunięcie pierwszego rekordu umieszczenia komunikatu. Jest to przesunięcie od początku MQPMO. W zależności od pola *PutMsgRecFields* wprowadź wartość inną niż null dla *PutMsgRecOffset* lub *PutMsgRecPtr*.

**ResponseRecprzesunięcie i ResponseRecPtr**
Za pomocą wskaźników i przesunięć można również adresować rekordy odpowiedzi (więcej informacji na temat rekordów odpowiedzi zawiera sekcja "Korzystanie ze struktury MQRR" na stronie 790 ).

Użyj pola *ResponseRecPtr* , aby określić wskaźnik do pierwszego rekordu odpowiedzi lub pola *ResponseRecOffset* , aby określić przesunięcie pierwszego rekordu odpowiedzi. Jest to przesunięcie od początku struktury MQPMO. Wprowadź wartość inną niż null dla *ResponseRecOffset* lub *ResponseRecPtr*.

**Uwaga:** Jeśli komunikaty są umieszczane na liście dystrybucyjnej za pomocą komendy MQPUT1 , parametr *ResponseRecPtr* musi mieć wartość NULL lub zero, a parametr *ResponseRecOffset* musi mieć wartość zero.

Wersja 3 struktury MQPMO zawiera dodatkowo następujące pola:

**OriginalMsgUchwyt**
Sposób użycia tego pola zależy od wartości pola *Działanie* . Jeśli umieszczany jest nowy komunikat z powiązanymi właściwościami komunikatu, należy ustawić w tym polu uchwyt komunikatu, który został wcześniej utworzony, i ustawić właściwości. W przypadku przekazywania, odpowiadania lub generowania raportu w odpowiedzi na wcześniej pobrany komunikat, to pole zawiera uchwyt tego komunikatu.

**NewMsgUchwyt**
Jeśli zostanie podany uchwyt *NewMsg*, wszystkie właściwości powiązane z uchwytem przesłonią właściwości powiązane z uchwytem *OriginalMsg*. Więcej informacji na ten temat zawiera sekcja Działanie (MQLONG).

**Działanie**
To pole służy do określania typu wykonywanego umieszczania. Możliwe wartości i ich znaczenia są następujące:

**MQACTP_NOWA**
To jest nowa wiadomość niezwiązana z żadną inną.

**MQACTP_FORWARD**
Ten komunikat został wcześniej pobrany i jest teraz przekazywany.

**ODPOWIEDŹ MQACTP_REPLY**
Ten komunikat jest odpowiedzią na poprzednio pobrany komunikat.

**RAPORT MQACTP_REPORT**
Ten komunikat jest raportem wygenerowanym w wyniku wcześniej pobranego komunikatu.

Więcej informacji na ten temat zawiera sekcja Działanie (MQLONG).

**PubLevel**
Jeśli ten komunikat jest publikacją, można ustawić to pole, aby określić, które subskrypcje odbierają ten komunikat. Tylko subskrypcje z poziomem *SubLevel* mniejszym lub równym tej wartości otrzymają tę publikację. Wartością domyślną jest 9, która jest najwyższym poziomem i oznacza, że subskrypcje z dowolnym *SubLevel* mogą otrzymać tę publikację.

## Dane w komunikacie

Podaj adres buforu zawierającego dane w parametrze **Buffer** wywołania MQPUT. Do wiadomości można dołączyć wszystko, co znajduje się w danych. Ilość danych w komunikatach wpływa jednak na wydajność aplikacji, która je przetwarza.

Maksymalna wielkość danych jest określana przez:

- Atrybut **MaxMsgLength** menedżera kolejek
- Atrybut **MaxMsgLength** kolejki, w której umieszczany jest komunikat.
- Wielkość dowolnego nagłówka komunikatu dodanego przez IBM MQ (w tym nagłówka niedostarczonego komunikatu, nagłówka MQDLH i nagłówka listy dystrybucyjnej, MQDH)

Atrybut **MaxMsgLength** menedżera kolejek przechowuje wielkość komunikatu, który może zostać przetworzona przez menedżer kolejek. Wartość domyślna wynosi 100 MB dla wszystkich produktów IBM MQ w wersji V6 lub nowszej.

Aby określić wartość tego atrybutu, należy użyć wywołania MQINQ dla obiektu menedżera kolejek. W przypadku dużych komunikatów można zmienić tę wartość.

Atrybut **MaxMsgLength** kolejki określa maksymalną wielkość komunikatu, który można umieścić w kolejce. Jeśli zostanie podjęta próba umieszczenia komunikatu o wielkości większej niż wartość tego atrybutu, wywołanie MQPUT nie powiedzie się. Jeśli komunikat jest umieszczany w kolejce zdalnej, maksymalna wielkość komunikatu, który można pomyślnie umieścić, jest określana przez atrybut **MaxMsgLength** kolejki zdalnej, wszystkich pośrednich kolejek transmisji, w których komunikat jest umieszczany, wzdłuż trasy do miejsca docelowego oraz używanych kanałów.

W przypadku operacji MQPUT wielkość komunikatu musi być mniejsza lub równa wartości atrybutu **MaxMsgLength** zarówno kolejki, jak i menedżera kolejek. Wartości tych atrybutów są niezależne, ale zaleca się ustawienie parametru *MaxMsgLength* kolejki na wartość mniejszą lub równą wartości menedżera kolejek.

IBM MQ dodaje informacje nagłówka do komunikatów w następujących okolicznościach:

- Po umieszczeniu komunikatu w kolejce zdalnej program IBM MQ dodaje do komunikatu strukturę nagłówka transmisji (MQXQH). Ta struktura zawiera nazwę kolejki docelowej i menedżera kolejek będącego jej właścicielem.
- Jeśli program IBM MQ nie może dostarczyć komunikatu do kolejki zdalnej, próbuje umieścić komunikat w kolejce niedostarczonych komunikatów (niedostarczonych komunikatów). Dodaje strukturę MQDLH do komunikatu. Ta struktura zawiera nazwę kolejki docelowej i przyczynę umieszczenia komunikatu w kolejce niedostarczonych komunikatów.
- Aby wysłać komunikat do wielu kolejek docelowych, produkt IBM MQ dodaje do komunikatu nagłówek MQDH. Opisuje dane znajdujące się w komunikacie, należącym do listy dystrybucyjnej, w kolejce transmisji. Należy to wziąć pod uwagę przy wyborze optymalnej wartości dla maksymalnej długości komunikatu.
- Jeśli komunikat jest segmentem lub komunikatem w grupie, produkt IBM MQ może dodać komunikat MQMDE.

Struktury te są opisane w sekcjach MQDH i MQMDE.

Jeśli komunikaty mają maksymalną wielkość dozwoloną dla tych kolejek, dodanie tych nagłówków oznacza, że operacje umieszczania nie powiodą się, ponieważ komunikaty są teraz zbyt duże. Aby zmniejszyć prawdopodobieństwo niepowodzenia operacji put:

- Ustaw wielkość komunikatów mniejszą niż wartość atrybutu **MaxMsgLength** w kolejkach transmisji i niewysłanych wiadomości. Zezwalaj na co najmniej wartość stałej MQ_MSG_HEADER_LENGTH (więcej dla dużych list dystrybucyjnych).
- Upewnij się, że atrybut **MaxMsgLength** kolejki niedostarczonych komunikatów jest ustawiony na taką samą wartość, jak atrybut *MaxMsgLength* menedżera kolejek, do którego należy kolejka niedostarczonych komunikatów.

Atrybuty menedżera kolejek i stałe kolejkowania komunikatów są opisane w sekcji Atrybuty menedżera kolejek.

z/OS Informacje na temat obsługi niedostarczonych komunikatów w rozproszonym środowisku kolejkowania zawiera sekcja Niedostarczone/nieprzetworzone komunikaty.

## Umieszczanie komunikatów: korzystanie z uchwytów komunikatów

W strukturze MQPMO dostępne są dwa uchwyty komunikatów: *OriginalMsgHandle* i *NewMsgHandle*. Relacja między tymi uchwytami komunikatów jest definiowana przez wartość pola *Działanie* MQPMO.

Szczegółowe informacje na ten temat zawiera sekcja Działanie (MQLONG). Uchwyt komunikatu nie jest wymagany do umieszczenia komunikatu. Jego celem jest powiązanie właściwości z komunikatem, dlatego jest on wymagany tylko wtedy, gdy używane są właściwości komunikatu.

### *Umieszczanie komunikatów w kolejce zdalnej*

Jeśli komunikat ma zostać umieszczony w kolejce zdalnej (czyli w kolejce, której właścicielem jest menedżer kolejek inny niż ten, z którym połączona jest aplikacja), a nie w kolejce lokalnej, jedyną dodatkową możliwością jest określenie nazwy kolejki podczas jej otwierania.Zostało to opisane w sekcji “Otwieranie kolejek zdalnych” na stronie 776. Nie ma zmian w sposobie użycia wywołania MQPUT lub MQPUT1 dla kolejki lokalnej.

Więcej informacji na temat używania kolejek zdalnych i kolejek transmisji zawiera sekcja Techniki kolejkowania rozproszonego IBM MQ.

### *Ustawianie właściwości komunikatu*

Wywołaj komendę MQSETMP dla każdej właściwości, która ma zostać ustawiona. Po umieszczeniu zestawu komunikatów w polach uchwytu komunikatu i działania struktury MQPMO.

Aby powiązać właściwości z komunikatem, komunikat musi mieć uchwyt komunikatu. Utwórz uchwyt komunikatu przy użyciu wywołania funkcji MQCRTMH. Wywołaj komendę MQSETMP, określając ten uchwyt komunikatu dla każdej właściwości, która ma zostać ustawiona. W celu zilustrowania użycia komendy MQSETMP udostępniono przykładowy program amqsstma.c.

Jeśli jest to nowy komunikat, po umieszczeniu go w kolejce przy użyciu wywołania MQPUT lub MQPUT1należy ustawić pole uchwytu OriginalMsgw programie MQPMO na wartość tego uchwytu komunikatu i ustawić pole działania MQPMO na wartość MQACTP_NEW (jest to wartość domyślna).

Jeśli jest to komunikat, który został wcześniej pobrany, i jest on teraz przesyłany lub na niego odpowiadany, lub wysyłany jest raport w odpowiedzi na ten komunikat, należy umieścić oryginalny uchwyt komunikatu w polu OriginalMsguchwytu MQPMO i nowy uchwyt komunikatu w polu NewMsguchwytu. W polu Działanie ustaw odpowiednio wartości MQACTP_FORWARD, MQACTP_REPLY lub MQACTP_REPORT.

Jeśli w nagłówku MQRFH2 znajdują się właściwości z wcześniej pobranego komunikatu, można je przekształcić w właściwości uchwytu komunikatu za pomocą wywołania MQBUFMH.

Jeśli komunikat jest umieszczany w kolejce w menedżerze kolejek na poziomie wcześniejszym niż IBM WebSphere MQ 7.0, który nie może przetworzyć właściwości komunikatu, można ustawić parametr PropertyControl w definicji kanału, aby określić sposób traktowania właściwości.

### *Sterowanie informacjami o kontekście komunikatu*

Jeśli do umieszczenia komunikatu w kolejce używane jest wywołanie MQPUT lub MQPUT1 , można określić, że menedżer kolejek ma dodać do deskryptora komunikatu pewne domyślne informacje o kontekście. Aplikacje, które mają odpowiedni poziom uprawnień, mogą dodawać dodatkowe informacje o kontekście. Do sterowania informacjami o kontekście można użyć pola opcji w strukturze MQPMO.

Informacje o kontekście komunikatu umożliwiają aplikacji, która pobiera komunikat, uzyskanie informacji o nadawcy komunikatu. Wszystkie informacje o kontekście są przechowywane w polach kontekstu deskryptora komunikatu. Typ informacji odpowiada informacjom o tożsamości, pochodzeniu i kontekście użytkownika.

Aby sterować informacjami o kontekście, należy użyć pola *Options* w strukturze MQPMO.

Jeśli nie zostaną określone żadne opcje informacji o kontekście, menedżer kolejek nadpisze informacje o kontekście, które mogą już znajdować się w deskryptorze komunikatu, przy użyciu informacji o tożsamości i kontekście, które zostały wygenerowane dla komunikatu. Jest to równoważne określeniu opcji MQPMO_DEFAULT_CONTEXT.Te domyślne informacje o kontekście mogą być potrzebne podczas tworzenia nowego komunikatu (na przykład podczas przetwarzania danych wejściowych użytkownika z ekranu zapytania).

Jeśli z komunikatem nie mają być powiązane żadne informacje o kontekście, należy użyć opcji MQPMO_NO_CONTEXT. Podczas umieszczania komunikatu bez kontekstu wszystkie sprawdzenia uprawnień wykonywane przez program IBM MQ są wykonywane przy użyciu pustego identyfikatora użytkownika. Pustego identyfikatora użytkownika nie można przypisać jawnego uprawnienia do zasobów IBM MQ , ale jest on traktowany jako członek grupy specjalnej 'nobody' (nikt). Więcej informacji na temat grupy specjalnej nobodyzawiera sekcja Informacje dodatkowe o interfejsie instalowalnych usług.

Ustawienia kontekstu można ustawić za pomocą komendy MQOPEN, a następnie komendy MQPUT z użyciem opcji MQOO_ i opcji MQPMO_ wskazanych w poniższych sekcjach. Ustawienie kontekstu można również wykonać przy użyciu tylko komendy MQPUT1. W takim przypadku wystarczy wybrać opcję MQPMO_ wskazaną w poniższych sekcjach.

W poniższych sekcjach tego tematu wyjaśniono użycie kontekstu tożsamości, kontekstu użytkownika i całego kontekstu.


- “Przekazywanie kontekstu tożsamości” na stronie 784
- “Przekazywanie kontekstu użytkownika” na stronie 785
- “Przekazywanie całego kontekstu” na stronie 785
- “Ustawianie kontekstu tożsamości” na stronie 785
- “Ustawianie kontekstu użytkownika” na stronie 785
- “Ustawianie całego kontekstu” na stronie 786


## Przekazywanie kontekstu tożsamości

Ogólnie rzecz biorąc, programy powinny przekazywać informacje o kontekście tożsamości z komunikatu do komunikatu, dopóki dane nie dotrą do miejsca docelowego.

Programy powinny zmieniać informacje o kontekście źródłowym za każdym razem, gdy zmieniają dane. Jednak aplikacje, które chcą zmienić lub ustawić informacje o kontekście, muszą mieć odpowiedni poziom uprawnień. Menedżer kolejek sprawdza to uprawnienie, gdy aplikacje otwierają kolejki. Muszą mieć uprawnienia do używania odpowiednich opcji kontekstu dla wywołania MQOPEN.

Jeśli aplikacja otrzyma komunikat, przetworzy dane z komunikatu, a następnie umieści zmienione dane w innym komunikacie (na potrzeby przetwarzania przez inną aplikację), aplikacja musi przekazać informacje o kontekście tożsamości z oryginalnego komunikatu do nowego komunikatu. Można zezwolić menedżerowi kolejek na tworzenie informacji o kontekście źródłowym.

Aby zapisać informacje o kontekście z oryginalnego komunikatu, należy użyć opcji MQOO_SAVE_ALL_CONTEXT podczas otwierania kolejki w celu uzyskania komunikatu. Jest to uzupełnienie innych opcji używanych w wywołaniu MQOPEN. Należy jednak zauważyć, że nie można zapisać informacji o kontekście, jeśli przeglądany jest tylko komunikat.

Podczas tworzenia drugiego komunikatu:

- Otwórz kolejkę za pomocą opcji MQOO_PASS_IDENTITY_CONTEXT (oprócz opcji MQOO_OUTPUT).
- W polu *Context* struktury opcji put-message podaj uchwyt kolejki, z której zostały zapisane informacje o kontekście.
- W polu *Options* struktury opcji put-message określ opcję MQPMO_PASS_IDENTITY_CONTEXT.

## Przekazywanie kontekstu użytkownika

Nie można przekazać tylko kontekstu użytkownika. Aby przekazać kontekst użytkownika podczas umieszczania komunikatu, należy określić parametr MQPMO_PASS_ALL_CONTEXT. Wszystkie właściwości w kontekście użytkownika są przekazywane w taki sam sposób, jak kontekst źródłowy.

Gdy ma miejsce operacja MQPUT lub MQPUT1 , a kontekst jest przekazywany, wszystkie właściwości w kontekście użytkownika są przekazywane z pobranego komunikatu do komunikatu umieszczania. Wszystkie właściwości kontekstu użytkownika, które zostały zmienione przez aplikację umieszczającą, są umieszczane z ich pierwotnymi wartościami. Wszystkie właściwości kontekstu użytkownika, które zostały usunięte przez aplikację umieszczającą, są odtwarzane w komunikacie umieszczanym. Wszystkie właściwości kontekstu użytkownika, które aplikacja wstawiająca dodała do komunikatu, są zachowywane.

## Przekazywanie całego kontekstu

Jeśli aplikacja otrzyma komunikat i umieści dane komunikatu (bez zmian) w innym komunikacie, aplikacja musi przekazać wszystkie informacje o kontekście (tożsamość, pochodzenie i użytkownik) z oryginalnego komunikatu do nowego komunikatu. Przykładem aplikacji, która może to zrobić, jest narzędzie przenoszenia komunikatów, które przenosi komunikaty z jednej kolejki do innej.

Wykonaj tę samą procedurę, co w przypadku przekazywania kontekstu tożsamości, z tą różnicą, że używana jest opcja MQOPEN MQOO_PASS_ALL_CONTEXT i opcja put-message MQPMO_PASS_ALL_CONTEXT.

## Ustawianie kontekstu tożsamości

Aby ustawić informacje o kontekście tożsamości dla komunikatu:

- Otwórz kolejkę za pomocą opcji MQOO_SET_IDENTITY_CONTEXT.
- Umieść komunikat w kolejce, określając opcję MQPMO_SET_IDENTITY_CONTEXT. W deskryptorze komunikatu określ wymagane informacje o kontekście tożsamości.

**Uwaga:** Podczas ustawiania niektórych (ale nie wszystkich) pól kontekstu tożsamości za pomocą opcji MQOO_SET_IDENTITY_CONTEXT i MQPMO_SET_IDENTITY_CONTEXT należy pamiętać, że menedżer kolejek nie ustawia żadnych innych pól.

Aby zmodyfikować dowolną z opcji kontekstu komunikatu, należy mieć odpowiednie autoryzacje do wywołania. Aby na przykład użyć komendy MQOO_SET_IDENTITY_CONTEXT lub MQPMO_SET_IDENTITY_CONTEXT, należy mieć uprawnienie `+setid` .

## Ustawianie kontekstu użytkownika

Aby ustawić właściwość w kontekście użytkownika, należy ustawić pole Context deskryptora właściwości komunikatu (MQPD) na wartość MQPD_USER_CONTEXT podczas wykonywania wywołania MQSETMP.

Do ustawienia właściwości w kontekście użytkownika nie są wymagane żadne uprawnienia specjalne. Kontekst użytkownika nie ma opcji kontekstu MQOO_SET_ * lub MQPMO_SET_ *.

## Ustawianie całego kontekstu

Aby ustawić zarówno informacje o tożsamości, jak i o kontekście źródłowym dla komunikatu:

1. Otwórz kolejkę za pomocą opcji MQOO_SET_ALL_CONTEXT.
2. Umieść komunikat w kolejce, podając opcję MQPMO_SET_ALL_CONTEXT. W deskryptorze komunikatu określ wymagane informacje o tożsamości i kontekście źródłowym.

Dla każdego typu ustawienia kontekstu wymagane są odpowiednie uprawnienia.

**Pojęcia pokrewne**

"Kontekst komunikatu" na stronie 48
Informacje *Kontekst komunikatu* umożliwiają aplikacji, która pobiera komunikat, uzyskanie informacji o twórcy komunikatu.

**Odsyłacze pokrewne**

"Opcje MQOPEN dotyczące kontekstu komunikatu" na stronie 774
Aby można było powiązać informacje o kontekście z komunikatem umieszczonym w kolejce, podczas otwierania kolejki należy użyć jednej z opcji kontekstu komunikatu.

## *Umieszczanie jednego komunikatu w kolejce za pomocą wywołania MQPUT1*

Wywołania MQPUT1 należy użyć, aby zamknąć kolejkę natychmiast po umieszczeniu w niej pojedynczego komunikatu. Na przykład aplikacja serwera może używać wywołania MQPUT1 , gdy wysyła odpowiedź do każdej z różnych kolejek.

MQPUT1 jest funkcjonalnie równoważne wywołaniu MQOPEN, po którym następuje MQPUT, a następnie MQCLOSE. Jedyna różnica w składni wywołań MQPUT i MQPUT1 polega na tym, że w przypadku MQPUT określany jest uchwyt obiektu, natomiast w przypadku MQPUT1 określana jest struktura deskryptora obiektu (MQOD) zdefiniowana w MQOPEN (patrz sekcja "Identyfikowanie obiektów (struktura MQOD)" na stronie 768 ). Jest to spowodowane koniecznością przekazywania do wywołania MQPUT1 informacji o kolejce, która ma zostać otwarta, podczas gdy w przypadku wywołania MQPUT kolejka musi być już otwarta.

Jako dane wejściowe wywołania MQPUT1 należy podać:

- Uchwyt połączenia.
- Opis obiektu, który ma zostać otwarty. Jest to struktura deskryptora obiektu (MQOD).
- Opis komunikatu, który ma zostać umieszczony w kolejce. Jest to struktura deskryptora komunikatu (MQMD).
- Informacje sterujące w postaci struktury opcji umieszczania komunikatu (MQPMO).
- Długość danych zawartych w komunikacie (MQLONG).
- Adres danych komunikatu.

Dane wyjściowe komendy MQPUT1 są następujące:

- Kod zakończenia
- Kod przyczyny

Jeśli wywołanie zakończy się pomyślnie, zwróci również strukturę opcji i strukturę deskryptora komunikatu. Wywołanie modyfikuje strukturę opcji, wyświetlając nazwę kolejki i menedżera kolejek, do którego został wysłany komunikat. W przypadku żądania wygenerowania przez menedżer kolejek unikalnej wartości dla identyfikatora umieszczanego komunikatu (przez określenie zera binarnego w polu *MsgId* struktury MQMD), wywołanie wstawia wartość w polu *MsgId* przed zwróceniem tej struktury do użytkownika.

**Uwaga:** Nie można używać opcji MQPUT1 z nazwą kolejki modelowej. Jednak po otwarciu kolejki modelowej można wprowadzić do kolejki dynamicznej komendę MQPUT1 .

Sześć parametrów wejściowych dla komendy MQPUT1 to:

***Hconn***
Jest to uchwyt połączenia. W przypadku aplikacji CICS można określić stałą MQHC_DEF_HCONN (która ma wartość zero) lub użyć uchwytu połączenia zwróconego przez wywołanie MQCONN lub MQCONNX. W przypadku innych programów należy zawsze używać uchwytu połączenia zwróconego przez wywołanie MQCONN lub MQCONNX.

***ObjDesc***
Jest to struktura deskryptora obiektu (MQOD).

W polach *ObjectName* i *ObjectQMgrName* należy podać nazwę kolejki, w której ma zostać umieszczony komunikat, oraz nazwę menedżera kolejek, który jest właścicielem tej kolejki.

Pole *DynamicQName* jest ignorowane w przypadku wywołania MQPUT1 , ponieważ nie może ono używać kolejek modelowych.

Pole *AlternateUserId* służy do wyznaczania alternatywnego identyfikatora użytkownika, który ma być używany do testowania uprawnień do otwierania kolejki.

***MsgDesc***
Jest to struktura deskryptora komunikatu (MQMD). Podobnie jak w przypadku wywołania MQPUT, ta struktura służy do definiowania komunikatu umieszczanego w kolejce.

***PutMsgOpts***
Jest to struktura opcji umieszczania komunikatów (MQPMO). Należy go używać w taki sam sposób, jak w przypadku wywołania MQPUT (patrz sekcja "Określanie opcji przy użyciu struktury MQPMO" na stronie 779 ).

Jeśli pole *Options* ma wartość zero, menedżer kolejek używa własnego identyfikatora użytkownika podczas wykonywania testów na potrzeby uprawnień dostępu do kolejki. Ponadto menedżer kolejek ignoruje każdy alternatywny identyfikator użytkownika podany w polu *AlternateUserId* struktury MQOD.

***BufferLength***
Jest to długość komunikatu.

***Buffer***
Jest to obszar buforu zawierający tekst komunikatu.

Jeśli używane są klastry, komenda MQPUT1 działa tak, jakby była używana opcja MQOO_BIND_NOT_FIXED. Aplikacje muszą używać rozstrzygniętych pól w strukturze MQPMO, a nie w strukturze MQOD, aby określić miejsce wysłania komunikatu. Więcej informacji na ten temat zawiera sekcja Konfigurowanie klastra menedżera kolejek .

Opis wywołania MQPUT1 znajduje się w sekcji MQPUT1.

## *Lista dystrybucyjna*

**Nieobsługiwane w systemie IBM MQ for z/OS.** Listy dystrybucyjne umożliwiają umieszczenie komunikatu w wielu miejscach docelowych w pojedynczym wywołaniu MQPUT lub MQPUT1 .Pojedyncze wywołanie MQOPEN może otwierać wiele kolejek, a pojedyncze wywołanie MQPUT może następnie umieścić komunikat w każdej z tych kolejek. Niektóre informacje ogólne ze struktur MQI używanych w tym procesie mogą zostać zastąpione konkretnymi informacjami dotyczącymi poszczególnych miejsc docelowych znajdujących się na liście dystrybucyjnej.

**Ostrzeżenie:** Listy dystrybucyjne nie obsługują korzystania z kolejek aliasowych, które wskazują na obiekty tematów. Jeśli kolejka aliasowa wskazuje na obiekt tematu na liście dystrybucyjnej, program IBM MQ zwraca MQRC_ALIAS_BASE_Q_TYPE_ERROR.

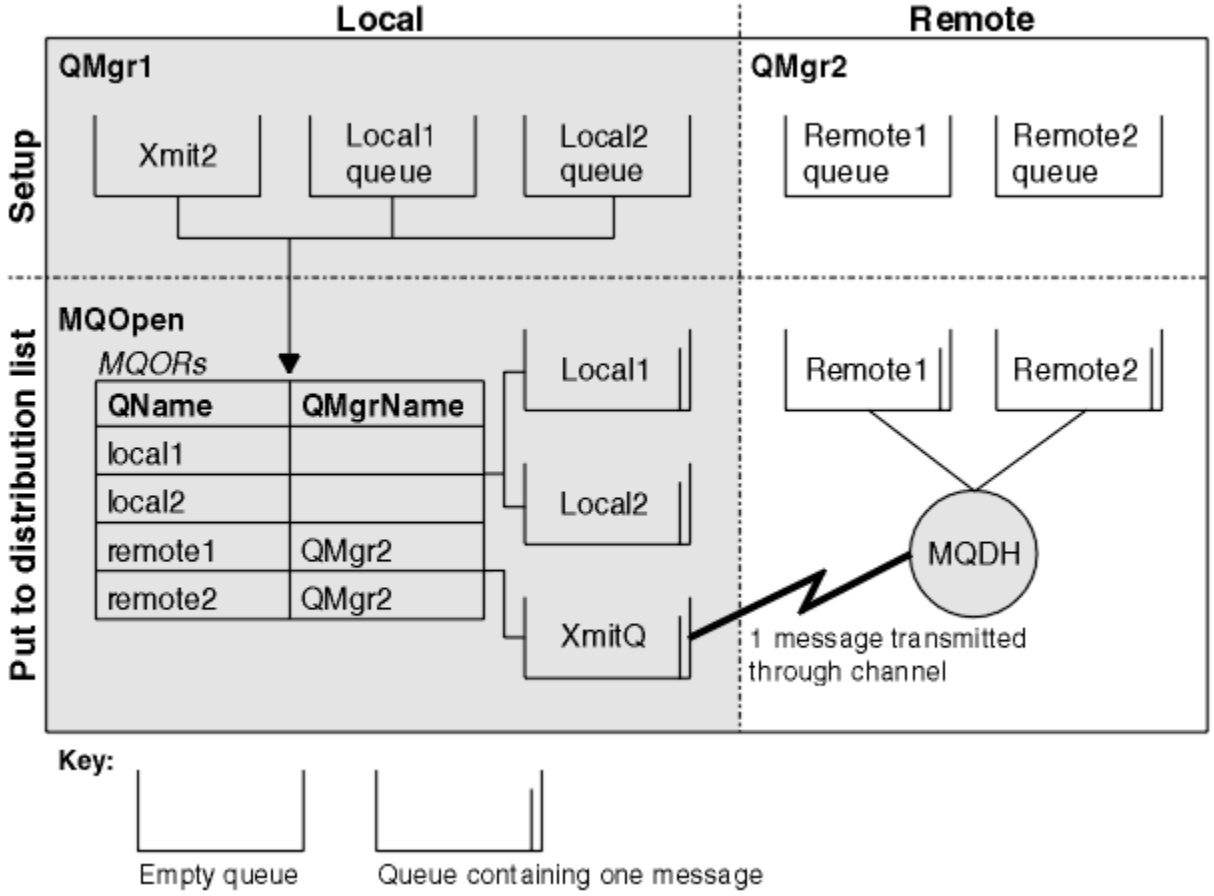
Po wywołaniu wywołania MQOPEN informacje ogólne są pobierane z deskryptora obiektu (MQOD). Jeśli w polu *Version* zostanie podana wartość MQOD_VERSION_2 i wartość większa niż zero w polu *RecsPresent* , *Hobj* może zostać zdefiniowany jako uchwyt listy (jednej lub większej liczby kolejek), a nie kolejki. W takim przypadku konkretne informacje są przekazywane za pośrednictwem rekordów obiektów (MQOR), które zawierają szczegóły miejsca docelowego (czyli *ObjectName* i *ObjectQMgrName*).

Uchwyt obiektu (*Hobj*) jest przekazywany do wywołania MQPUT, co umożliwia umieszczenie go na liście, a nie w pojedynczej kolejce.

Gdy komunikat jest umieszczany w kolejkach (MQPUT), informacje ogólne są pobierane ze struktury opcji umieszczania komunikatu (MQPMO) i deskryptora komunikatu (MQMD). Konkretne informacje są podawane w postaci rekordów umieszczania komunikatów (MQPMR).

Rekordy odpowiedzi (MQRR) mogą odbierać kod zakończenia i kod przyczyny specyficzne dla każdej kolejki docelowej.

Rysunek 56 na stronie 788 przedstawia sposób działania list dystrybucyjnych.




*Rysunek 56. Sposób działania list dystrybucyjnych*

*Otwieranie list dystrybucyjnych*
Użyj wywołania MQOPEN, aby otworzyć listę dystrybucyjną, a następnie użyj opcji wywołania, aby określić, co chcesz zrobić z listą.

Jako dane wejściowe dla MQOPEN należy podać:

- Uchwyt połączenia (opis znajduje się w sekcji "Umieszczanie komunikatów w kolejce" na stronie 777 )
- Informacje ogólne w strukturze deskryptora obiektu (MQOD)
- Nazwa każdej kolejki, która ma zostać otwarta, przy użyciu struktury rekordu obiektu (MQOR)

Dane wyjściowe komendy MQOPEN są następujące:

- Uchwyt obiektu reprezentujący dostęp użytkownika do listy dystrybucyjnej
- Ogólny kod zakończenia
- Ogólny kod przyczyny
- Rekordy odpowiedzi (opcjonalne), zawierające kod zakończenia i przyczynę dla każdego miejsca docelowego

## Korzystanie ze struktury MQOD

Struktura MQOD służy do identyfikowania kolejek, które mają zostać otwarte.

Aby zdefiniować listę dystrybucyjną, należy określić wartość MQOD_VERSION_2 w polu *Version* , wartość większą niż zero w polu *RecsPresent* i wartość MQOT_Q w polu *ObjectType* . Opis wszystkich pól struktury MQOD zawiera sekcja MQOD .

## Korzystanie ze struktury MQOR

Podaj strukturę MQOR dla każdego miejsca docelowego.

Struktura zawiera nazwy kolejki docelowej i menedżera kolejek. Pola *ObjectName* i *ObjectQMgrName* w programie MQOD nie są używane na potrzeby list dystrybucyjnych. Musi istnieć co najmniej jeden rekord obiektu. Jeśli pole *ObjectQMgrName* pozostanie puste, zostanie użyty lokalny menedżer kolejek. Więcej informacji na temat tych pól zawierają ObjectName i ObjectQMgrName .

Kolejki docelowe można określić na dwa sposoby:

- Używając zmiennej przesunięcia *ObjectRecOffset*.

  W takim przypadku aplikacja musi zadeklarować własną strukturę zawierającą strukturę MQOD, po której następuje tablica rekordów MQOR (z wymaganą liczbą elementów tablicy) i ustawić parametr *ObjectRecOffset* na przesunięcie pierwszego elementu w tablicy od początku MQOD. Upewnij się, że to przesunięcie jest poprawne.

  Zaleca się korzystanie z wbudowanych narzędzi udostępnianych przez język programowania, jeśli są one dostępne we wszystkich środowiskach, w których działa aplikacja. Poniższy kod ilustruje tę technikę w języku programowania COBOL:

```
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE LENGTH OF MY-MQOD TO MQOD-OBJECTRECOFFSET.
```

  Alternatywnie można użyć stałej MQOD_CURRENT_LENGTH, jeśli język programowania nie obsługuje niezbędnych wbudowanych narzędzi we wszystkich danych środowiskach. Technika ta została zilustrowana przez następujący kod:

```
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.
```

  Jednak działa to poprawnie tylko wtedy, gdy struktura MQOD i tablica rekordów MQOR są ciągłe. Jeśli kompilator wstawia pomijane bajty między MQOD i tablicą MQOR, należy je dodać do wartości zapisanej w pliku *ObjectRecOffset*.

  Użycie języka *ObjectRecOffset* jest zalecane w przypadku języków programowania, które nie obsługują typu danych wskaźnika lub implementują typ danych wskaźnika w sposób, który nie jest przenośny w różnych środowiskach (na przykład w języku programowania COBOL).

- Używając pola wskaźnika *ObjectRecPtr*.

  W takim przypadku aplikacja może zadeklarować tablicę struktur MQOR niezależnie od struktury MQOD i ustawić parametr *ObjectRecPtr* na adres tablicy. Poniższy kod ilustruje tę technikę w języku programowania C:

```
MQOD MyMqod;
```

```
MQOR MyMqor[100];
MyMqod.ObjectRecPtr = MyMqor;
```

Używanie języka *ObjectRecPtr* jest zalecane w przypadku języków programowania obsługujących typ danych wskaźnika w sposób przenośny do różnych środowisk (na przykład w języku programowania C).

Niezależnie od wybranej techniki należy użyć jednej z następujących metod: *ObjectRecOffset* i *ObjectRecPtr* ; Wywołanie nie powiodło się z kodem przyczyny MQRC_OBJECT_RECORDS_ERROR, jeśli obie wartości są równe zero lub obie są niezerowe.

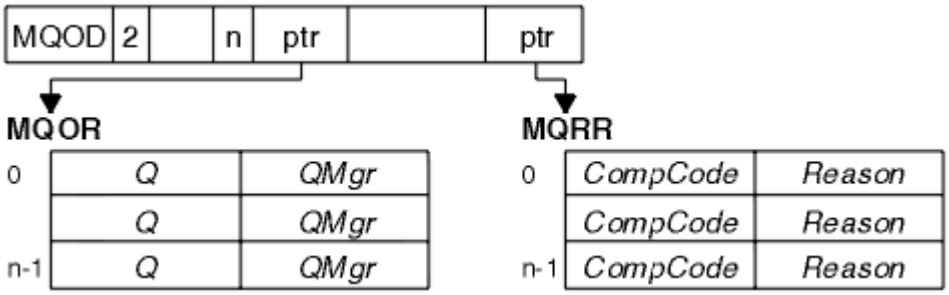
## Korzystanie ze struktury MQRR

Struktury te są specyficzne dla miejsca docelowego; każdy rekord odpowiedzi zawiera pole *CompCode* i *Reason* dla każdej kolejki listy dystrybucyjnej. Należy użyć tej struktury, aby umożliwić rozróżnienie miejsca występowania problemów.

Jeśli na przykład zostanie odebrany kod przyczyny MQRC_MULTIPLE_REASON, a lista dystrybucyjna zawiera pięć kolejek docelowych, nie będzie wiadomo, do których kolejek mają zastosowanie problemy, jeśli nie zostanie użyta ta struktura. Jeśli jednak dla każdego miejsca docelowego istnieje kod zakończenia i kod przyczyny, można łatwiej zlokalizować błędy.

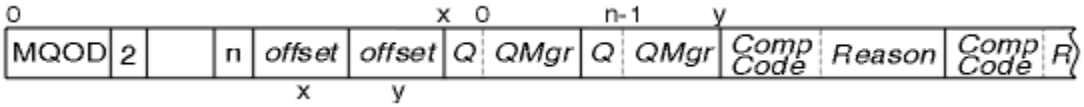
Więcej informacji na temat struktury MQRR zawiera sekcja MQRR .

Rysunek 57 na stronie 790 pokazuje, w jaki sposób można otworzyć listę dystrybucyjną w języku C.




*Rysunek 57. Otwieranie listy dystrybucyjnej w języku C*

Rysunek 58 na stronie 790 przedstawia sposób otwierania listy dystrybucyjnej w języku COBOL.




*Rysunek 58. Otwieranie listy dystrybucyjnej w języku COBOL*

## Korzystanie z opcji MQOPEN

Podczas otwierania listy dystrybucyjnej można określić następujące opcje:

- MQOO_OUTPUT
- MQOO_FAIL_IF_QUIESCING (opcjonalny)
- MQOO_ALTERNATE_USER_AUTHORITY (opcjonalne)
- MQOO_ * _CONTEXT (opcjonalny)

Opis tych opcji zawiera sekcja “Otwieranie i zamykanie obiektów” na stronie 766 .

*Umieszczanie komunikatów na liście dystrybucyjnej*
Aby umieścić komunikaty na liście dystrybucyjnej, można użyć komendy MQPUT lub MQPUT1.

Jako dane wejściowe należy podać:

- Uchwyt połączenia (opis zawiera sekcja “Umieszczanie komunikatów w kolejce” na stronie 777 ).

- Uchwyt obiektu. Jeśli lista dystrybucyjna jest otwierana za pomocą komendy MQOPEN, program *Hobj* umożliwia tylko umieszczenie jej na liście.
- Struktura deskryptora komunikatu (MQMD). Opis tej struktury zawiera sekcja MQMD .
- Informacje sterujące w postaci struktury opcji umieszczania komunikatu (MQPMO). Informacje na temat wypełniania pól struktury MQPMO zawiera sekcja “Określanie opcji przy użyciu struktury MQPMO” na stronie 779 .
- Informacje sterujące w postaci rekordów umieszczania komunikatów (MQPMR).
- Długość danych zawartych w komunikacie (MQLONG).
- Same dane komunikatu.

Dane wyjściowe:

- Kod zakończenia
- Kod przyczyny
- Rekordy odpowiedzi (opcjonalnie)

## Korzystanie ze struktury MQPMR

Ta struktura jest opcjonalna i udostępnia informacje specyficzne dla miejsca docelowego dla niektórych pól, które można zidentyfikować w inny sposób niż te, które zostały już określone w deskryptorze MQMD.

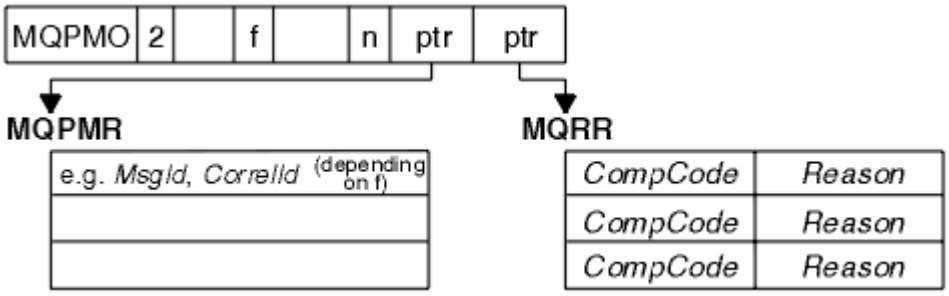
Opis tych pól zawiera sekcja MQPMR.

Zawartość każdego rekordu zależy od informacji podanych w polu *PutMsgRecFields* MQPMO. Na przykład w przykładowym programie AMQSPTL0.C (opis znajduje się w sekcji “Przykładowy program listy dystrybucyjnej” na stronie 1121 ) pokazującej użycie list dystrybucyjnych, przykład wybiera podanie wartości dla *MsgId* i *CorrelId* w MQPMR. Ta sekcja programu przykładowego wygląda następująco:

```
typedef struct
{
MQBYTE24 MsgId;
MQBYTE24 CorrelId;
} PutMsgRec;
...
/**********************
MQLONG PutMsgRecFields=MQPMRF_MSG_ID | MQPMRF_CORREL_ID;
```

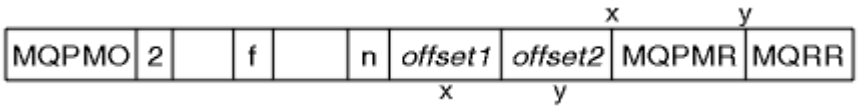
Oznacza to, że *MsgId* i *CorrelId* są udostępniane dla każdego miejsca docelowego listy dystrybucyjnej. Rekordy umieszczania komunikatów są udostępniane w postaci tablicy.

Rysunek 59 na stronie 791 pokazuje, w jaki sposób można umieścić komunikat na liście dystrybucyjnej w C.




*Rysunek 59. Umieszczanie komunikatu na liście dystrybucyjnej w języku C*

Rysunek 60 na stronie 792 pokazuje, w jaki sposób można umieścić komunikat na liście dystrybucyjnej w języku COBOL.




*Rysunek 60. Umieszczanie komunikatu na liście dystrybucyjnej w języku COBOL*

## Korzystanie z MQPUT1

Jeśli używany jest MQPUT1, należy wziąć pod uwagę następujące kwestie:

1. Wartości w polach *ResponseRec0ffset* i *ResponseRecPtr* muszą być równe null lub zeru.
2. Rekordy odpowiedzi, jeśli są wymagane, muszą być adresowane z MQOD.

### *Niektóre przypadki, w których wywołania put nie powiodły się*

Jeśli niektóre atrybuty kolejki zostaną zmienione przy użyciu opcji FORCE komendy w okresie między wywołaniem MQOPEN a wywołaniem MQPUT, wywołanie MQPUT nie powiedzie się i zwróci kod przyczyny MQRC_OBJECT_CHANGED.

Menedżer kolejek oznacza uchwyt obiektu jako niepoprawny. Dzieje się tak również wtedy, gdy zmiany są wprowadzane podczas przetwarzania wywołania MQPUT1 lub gdy zmiany dotyczą dowolnej kolejki, do której rozstrzygana jest nazwa kolejki. Atrybuty wpływające w ten sposób na uchwyt są wymienione w opisie wywołania MQOPEN w tabeli MQOPEN. Jeśli wywołanie zwraca kod przyczyny MQRC_OBJECT_CHANGED, zamknij kolejkę, otwórz ją ponownie, a następnie ponów próbę umieszczenia komunikatu.

Jeśli operacje umieszczania są zablokowane dla kolejki, w której podejmowana jest próba umieszczenia komunikatów (lub dowolnej kolejki, na którą tłumaczona jest nazwa kolejki), wywołanie MQPUT lub MQPUT1 kończy się niepowodzeniem i zwraca kod przyczyny MQRC_PUT_INHIBITED. Pomyślne umieszczenie komunikatu może być możliwe w przypadku próby wywołania w późniejszym czasie, jeśli projekt aplikacji jest taki, że inne programy regularnie zmieniają atrybuty kolejek.

Jeśli kolejka, w której ma zostać umieszczony komunikat, jest pełna, wywołanie MQPUT lub MQPUT1 kończy się niepowodzeniem i zwraca wartość MQRC_Q_FULL.

Jeśli kolejka dynamiczna (tymczasowa lub trwała) została usunięta, wywołania MQPUT używające poprzednio uzyskanego uchwytu obiektu nie powiodą się i zwrócą kod przyczyny MQRC_Q_DELETED. W takiej sytuacji zaleca się zamknięcie uchwytu obiektu, ponieważ nie jest on już używany.

W przypadku list dystrybucyjnych w pojedynczym żądaniu może wystąpić wiele kodów zakończenia i kodów przyczyny. Nie można ich obsłużyć tylko za pomocą pól wyjściowych *CompCode* i *Reason* w MQOPEN i MQPUT.

Jeśli listy dystrybucyjne są używane do umieszczania komunikatów w wielu miejscach docelowych, rekordy odpowiedzi zawierają konkretne *CompCode* i *Reason* dla każdego miejsca docelowego. Jeśli zostanie odebrany kod zakończenia MQCC_FAILED, żaden komunikat nie zostanie pomyślnie umieszczony w żadnej kolejce docelowej. Jeśli kod zakończenia ma wartość MQCC_WARNING, komunikat jest pomyślnie umieszczany w co najmniej jednej kolejce docelowej. Jeśli zostanie odebrany kod powrotu MQRC_MULTIPLE_REASON, kody przyczyny nie będą takie same dla każdego miejsca docelowego. Dlatego zaleca się użycie struktury MQRR w celu określenia kolejki lub kolejek, które spowodowały błąd, oraz przyczyn każdego z nich.

## Pobieranie komunikatów z kolejki

Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

Komunikaty z kolejki można pobrać na dwa sposoby:

1. Komunikat można usunąć z kolejki, aby inne programy nie mogły go już zobaczyć.
2. Można skopiować komunikat, pozostawiając oryginalny komunikat w kolejce. Jest to nazywane *przeglądaniem*. Po przeglądnięciu komunikatu można go usunąć.

W obu przypadkach należy użyć wywołania MQGET, ale najpierw aplikacja musi być połączona z menedżerem kolejek, a następnie należy użyć wywołania MQOPEN, aby otworzyć kolejkę (dla wejścia, przeglądania lub dla obu). Te operacje są opisane w sekcjach “Nawiązywanie i rozłączanie połączenia z menedżerem kolejek” na stronie 758 i “Otwieranie i zamykanie obiektów” na stronie 766.

Po otwarciu kolejki można za pomocą wywołania MQGET wielokrotnie przeglądać lub usuwać komunikaty w tej samej kolejce. Po zakończeniu pobierania wszystkich komunikatów z kolejki wywołaj komendę MQCLOSE.

Aby uzyskać więcej informacji na temat pobierania komunikatów z kolejki, należy użyć następujących odsyłaczy:

- “Pobieranie komunikatów z kolejki przy użyciu wywołania MQGET” na stronie 794
- “Kolejność, w jakiej komunikaty są pobierane z kolejki” na stronie 798
- “Uzyskiwanie konkretnego komunikatu” na stronie 810
- “Zwiększanie wydajności komunikatów nietrwałych” na stronie 811
- z/OS “Typ indeksu” na stronie 816
- “Obsługa komunikatów o długości większej niż 4 MB” na stronie 816
- “Oczekiwanie na komunikaty” na stronie 822
- z/OS “Sygnalizacja” na stronie 823
- “Pomijanie wycofywania” na stronie 824
- “Konwersja danych aplikacji” na stronie 827
- “Przeglądanie komunikatów w kolejce” na stronie 828
- “Niektóre przypadki, w których wywołanie MQGET kończy się niepowodzeniem” na stronie 834

**Pojęcia pokrewne**

“Przegląd interfejsu kolejki komunikatów” na stronie 745
Sekcja zawiera informacje na temat komponentów interfejsu kolejki komunikatów (Message Queue Interface-MQI).

“Nawiązywanie i rozłączanie połączenia z menedżerem kolejek” na stronie 758
Aby można było używać usług programistycznych IBM MQ , program musi mieć połączenie z menedżerem kolejek. Ten temat zawiera informacje o nawiązywaniu połączenia z menedżerem kolejek i rozłączaniu się z nim.

“Otwieranie i zamykanie obiektów” na stronie 766
Informacje te udostępniają wgląd w otwieranie i zamykanie obiektów IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 777
Ta sekcja zawiera informacje na temat umieszczania komunikatów w kolejce.

“Uzyskiwanie informacji o atrybutach obiektu i ustawianie ich” na stronie 876
Atrybuty są właściwościami definiującymi charakterystykę obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 879
W tej sekcji opisano sposób zatwierdzania i wycofywania wszelkich odtwarzalnych operacji pobierania i umieszczania, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy” na stronie 891
Informacje o wyzwalaczach i sposobie uruchamiania aplikacji IBM MQ za pomocą wyzwalaczy.

“Praca z funkcją MQI i klastrami” na stronie 911
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do łączenia w klastry.

“Używanie i pisanie aplikacji w systemie IBM MQ for z/OS” na stronie 916
Aplikacje IBM MQ for z/OS mogą być tworzone z programów działających w wielu różnych środowiskach. Oznacza to, że mogą korzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje mostu IMS i IMS w systemie IBM MQ for z/OS” na stronie 71
Informacje te są pomocne podczas pisania aplikacji IMS przy użyciu produktu IBM MQ.

### *Pobieranie komunikatów z kolejki przy użyciu wywołania MQGET*

Wywołanie MQGET pobiera komunikat z otwartej kolejki lokalnej. Nie może on pobrać komunikatu z kolejki w innym systemie.

Jako dane wejściowe wywołania MQGET należy podać:

- Uchwyt połączenia.
- Uchwyt kolejki.
- Opis komunikatu, który ma zostać odebrany z kolejki. Jest to struktura deskryptora komunikatu (MQMD).
- Informacje sterujące w postaci struktury Get Message Options (MQGMO).
- Wielkość buforu przypisanego do przechowywania komunikatu (MQLONG).
- Adres pamięci, w której ma zostać umieszczony komunikat.

Dane wyjściowe komendy MQGET to:

- Kod przyczyny
- Kod zakończenia
- Komunikat w podanym obszarze buforu, jeśli wywołanie zakończyło się pomyślnie
- Struktura opcji zmodyfikowana w celu wyświetlenia nazwy kolejki, z której komunikat został pobrany
- Struktura deskryptora komunikatu z zawartością pól zmodyfikowanych w celu opisania pobranego komunikatu.
- Długość komunikatu (MQLONG)

Opis wywołania MQGET znajduje się w sekcji MQGET.

W poniższych sekcjach opisano informacje, które należy podać jako dane wejściowe dla wywołania MQGET.

- “Określanie uchwytów połączeń” na stronie 794
- “Opisywanie komunikatów przy użyciu struktury MQMD i wywołania MQGET” na stronie 794
- “Określanie opcji MQGET przy użyciu struktury MQGMO” na stronie 795
- “Określanie wielkości obszaru buforu” na stronie 797

## Określanie uchwytów połączeń

z/OS W przypadku produktu CICS w aplikacjach z/OS można określić stałą MQHC_DEF_HCONN (która ma wartość zero) lub użyć uchwytu połączenia zwróconego przez wywołanie MQCONN lub MQCONNX. W przypadku innych aplikacji należy zawsze używać uchwytu połączenia zwróconego przez wywołanie MQCONN lub MQCONNX.

Należy użyć uchwytu kolejki (*Hobj*), który jest zwracany po wywołaniu MQOPEN.

## Opisywanie komunikatów przy użyciu struktury MQMD i wywołania MQGET

Aby zidentyfikować komunikat, który ma zostać odebrany z kolejki, należy użyć struktury deskryptora komunikatu (MQMD).

Jest to parametr wejścia/wyjścia wywołania MQGET. Istnieje wprowadzenie do właściwości komunikatu, które MQMD opisuje w “Komunikaty produktu IBM MQ” na stronie 18, oraz opis samej struktury w MQMD.

Jeśli wiadomo, który komunikat ma zostać odebrany z kolejki, należy zapoznać się z sekcją “Uzyskiwanie konkretnego komunikatu” na stronie 810.

Jeśli określony komunikat nie zostanie określony, komenda MQGET pobierze *pierwszy* komunikat w kolejce. W sekcji “Kolejność, w jakiej komunikaty są pobierane z kolejki” na stronie 798 opisano, w jaki sposób priorytet komunikatu, atrybut **MsgDeliverySequence** kolejki oraz opcja MQGMO_LOGICAL_ORDER określają kolejność komunikatów w kolejce.

**Uwaga:** Jeśli usługa MQGET ma być używana więcej niż jeden raz (na przykład w celu krokowego wykonywania komunikatów w kolejce), należy ustawić pola *MsgId* i *CorrelId* tej struktury na wartość NULL po każdym wywołaniu. Spowoduje to skasowanie tych pól z identyfikatorów pobranego komunikatu.

Jeśli jednak komunikaty mają być grupowane, wartość *GroupId* musi być taka sama dla komunikatów w tej samej grupie, aby wywołanie mogło znaleźć komunikat o takich samych identyfikatorach, jak poprzedni komunikat, aby utworzyć całą grupę.

## Określanie opcji MQGET przy użyciu struktury MQGMO

Struktura MQGMO jest zmienną wejściową/wyjściową do przekazywania opcji do wywołania MQGET. Poniższe sekcje ułatwiają wypełnienie niektórych pól tej struktury.

Istnieje opis struktury MQGMO w MQGMO.

***StrucId***
*StrucId* jest 4-znakowym polem używanym do identyfikowania struktury jako struktury opcji get-message. Zawsze podawaj wartość MQGMO_STRUC_ID.

***Version***
*Version*opisuje numer wersji struktury. MQGMO_VERSION_1 jest wartością domyślną. Aby użyć pól wersji 2 lub pobrać komunikaty w kolejności logicznej, należy określić wartość MQGMO_VERSION_2. Aby użyć pól wersji 3 lub pobrać komunikaty w kolejności logicznej, należy określić wartość MQGMO_VERSION_3. Opcja MQGMO_CURRENT_VERSION ustawia w aplikacji najnowszą wersję.

***Options***
W kodzie można wybrać opcje w dowolnej kolejności; każda opcja jest reprezentowana przez bit w polu *Options* .

Pole *Options* steruje:

- Określa, czy wywołanie MQGET oczekuje na przybycie komunikatu do kolejki przed jego zakończeniem (patrz sekcja “Oczekiwanie na komunikaty” na stronie 822 ).
- Określa, czy operacja pobierania jest uwzględniona w jednostce pracy.
- Określa, czy komunikat nietrwały jest pobierany poza punktem synchronizacji, co umożliwia szybkie przesyłanie komunikatów
- z/OS W systemie IBM MQ for z/OS, czy pobrany komunikat jest oznaczony jako pominięty (patrz sekcja “Pomijanie wycofywania” na stronie 824 )
- Określa, czy komunikat jest usuwany z kolejki, czy tylko przeglądany
- Wybór komunikatu za pomocą kursora przeglądania lub innych kryteriów wyboru
- Określa, czy wywołanie powiedzie się, nawet jeśli komunikat jest dłuższy niż bufor
- z/OS W systemie IBM MQ for z/OS: czy zezwolić na zakończenie wywołania. Ta opcja powoduje również ustawienie sygnału wskazującego, że użytkownik ma być powiadamiany o nadejściu komunikatu.
- Określa, czy wywołanie nie powiedzie się, jeśli menedżer kolejek jest w stanie wyciszania.
- z/OS W systemie IBM MQ for z/OS: określa, czy wywołanie nie powiedzie się, jeśli połączenie jest w stanie wyciszania.
- Określa, czy wymagana jest konwersja danych komunikatu aplikacji (patrz sekcja “Konwersja danych aplikacji” na stronie 827 ).
- Kolejność, w jakiej komunikaty i segmenty są pobierane z kolejki z/OS (z wyjątkiem IBM MQ for z/OS )
- Określa, czy kompletne komunikaty logiczne można pobierać tylko z/OS (z wyjątkiem systemu IBM MQ for z/OS )
- Określa, czy komunikaty w grupie mogą być pobierane tylko wtedy, gdy dostępne są *wszystkie* komunikaty w grupie.

- Określa, czy segmenty w komunikacie logicznym mogą zostać pobrane tylko wtedy, gdy *wszystkie* segmenty w komunikacie logicznym są dostępne z/OS (z wyjątkiem IBM MQ for z/OS ).

Jeśli pole *Options* pozostanie ustawione na wartość domyślną (MQGMO_NO_WAIT), wywołanie MQGET będzie działać w następujący sposób:

- Jeśli w kolejce nie ma komunikatu zgodnego z kryteriami wyboru, wywołanie nie czeka na nadejście komunikatu, ale kończy się natychmiast. z/OS Ponadto w programie IBM MQ for z/OSwywołanie nie ustawia sygnału żądającego powiadomienia w momencie nadejścia takiego komunikatu.
- Sposób działania wywołania z punktami synchronizacji jest określany przez platformę:

| **Platforma** | **Pod kontrolą punktu synchronizacji** |
|---|---|
| IBM i | Nie |
| Systemy AIX and Linux | Nie |
| z/OS z/OS z/OS | Tak |
| Systemy Windows | Nie |

- z/OS W systemie IBM MQ for z/OSpobrany komunikat nie jest oznaczany jako pominięty.
- Wybrany komunikat zostanie usunięty z kolejki (bez przeglądania).
- Konwersja danych komunikatu aplikacji nie jest wymagana.
- Wywołanie nie powiedzie się, jeśli komunikat jest dłuższy niż bufor.

***WaitInterval***
Pole *WaitInterval* określa maksymalny czas (w milisekundach), przez jaki wywołanie MQGET oczekuje na przybycie komunikatu do kolejki, gdy używana jest opcja MQGMO_WAIT. Jeśli w czasie określonym w parametrze *WaitInterval*nie zostanie odebrany żaden komunikat, wywołanie zostanie zakończone i zostanie zwrócony kod przyczyny wskazujący, że w kolejce nie ma komunikatu zgodnego z kryteriami wyboru.

z/OS W systemie IBM MQ for z/OS, jeśli używana jest opcja MQGMO_SET_SIGNAL, pole *WaitInterval* określa czas ustawienia sygnału.

Więcej informacji na temat tych opcji zawierają “Oczekiwanie na komunikaty” na stronie 822 z/OS i “Sygnalizacja” na stronie 823 .

z/OS ***Signal1***
Sygnał Signal1 jest obsługiwany tylko w systemie IBM MQ for z/OS.

Jeśli opcja MQGMO_SET_SIGNAL jest używana do żądania powiadomienia aplikacji o nadejściu odpowiedniego komunikatu, należy określić typ sygnału w polu *Signal1* . W systemie IBM MQ na wszystkich innych platformach pole *Signal1* jest zarezerwowane, a jego wartość nie jest istotna.

z/OS Więcej informacji na ten temat zawiera “Sygnalizacja” na stronie 823.

***Signal2***
Pole *Signal2* jest zarezerwowane na wszystkich platformach, a jego wartość nie jest istotna.

z/OS Więcej informacji na ten temat zawiera sekcja “Sygnalizacja” na stronie 823.

***ResolvedQName***
*ResolvedQName* to pole wyjściowe, w którym menedżer kolejek zwraca nazwę kolejki (po przetłumaczeniu dowolnego aliasu), z której został pobrany komunikat.

***MatchOptions***
Parametr *MatchOptions* steruje kryteriami wyboru dla wywołania MQGET.

***GroupStatus***
*GroupStatus* wskazuje, czy pobrany komunikat należy do grupy.

***SegmentStatus***
*SegmentStatus* wskazuje, czy pobrany element jest segmentem komunikatu logicznego.

***Segmentation***
*Segmentation* wskazuje, czy segmentacja jest dozwolona dla pobranego komunikatu.

***MsgToken***

*MsgToken* Jednoznacznie identyfikuje komunikat.

***ReturnedLength***
*ReturnedLength* to pole wyjściowe, w którym menedżer kolejek zwraca długość zwracanych danych komunikatu (w bajtach).

***MsgHandle***
Uchwyt komunikatu, który ma zostać zapełniony właściwościami komunikatu pobieranego z kolejki. Uchwyt został wcześniej utworzony przez wywołanie MQCRTMH. Wszystkie właściwości, które są już powiązane z uchwytem, są czyszczone przed pobraniem komunikatu.

## Określanie wielkości obszaru buforu

W parametrze **BufferLength** wywołania MQGET określ wielkość obszaru buforu, w którym mają być przechowywane pobierane dane komunikatu. Użytkownik decyduje o wielkości tego pliku na trzy sposoby:

1. Użytkownik może już wiedzieć, jakiej długości komunikatów oczekuje od tego programu. Jeśli tak, należy podać bufor o tej wielkości.

   Można jednak użyć opcji MQGMO_ACCEPT_TRUNCATED_MSG w strukturze MQGMO, jeśli wywołanie MQGET ma zostać zakończone, nawet jeśli komunikat jest zbyt duży dla buforu. W tym przypadku:

   - Bufor jest wypełniony tak dużą ilością komunikatu, jak może pomieścić
   - Wywołanie zwraca kod zakończenia ostrzeżenia
   - Komunikat jest usuwany z kolejki (usuwa pozostałą część komunikatu) lub kursor przeglądania jest zaawansowany (jeśli kolejka jest przeglądana)
   - Rzeczywista długość komunikatu jest zwracana w postaci *DataLength*

   Bez tej opcji wywołanie nadal kończy się ostrzeżeniem, ale nie usuwa komunikatu z kolejki (ani nie przesuwa kursora przeglądania).
2. Oszacuj wielkość buforu (lub nawet określ wielkość równą zero bajtów) i *nie* używaj opcji MQGMO_ACCEPT_TRUNCATED_MSG. Jeśli wywołanie MQGET nie powiedzie się (na przykład z powodu zbyt małego buforu), w parametrze **DataLength** wywołania zostanie zwrócona długość komunikatu. (Bufor jest nadal wypełniony jak najwięcej komunikatu, ale przetwarzanie wywołania nie zostało zakończone). Zapisz plik *MsgId* tego komunikatu, a następnie powtórz wywołanie MQGET, określając obszar buforu o poprawnej wielkości oraz wartość *MsgId* zanotowaną w pierwszym wywołaniu.

   Jeśli program obsługuje kolejkę, która jest również obsługiwana przez inne programy, jeden z tych programów może usunąć żądany komunikat, zanim program będzie mógł wysłać kolejne wywołanie MQGET. Program może tracić czas na wyszukiwanie komunikatu, który już nie istnieje. Aby tego uniknąć, należy najpierw przeglądać kolejkę aż do znalezienia żądanego komunikatu, określając wartość *BufferLength* równą zero i używając opcji MQGMO_ACCEPT_TRUNCATED_MSG. Spowoduje to umieszczenie kursora przeglądania pod wybranym komunikatem. Następnie można pobrać komunikat, ponownie wywołując komendę MQGET z opcją MQGMO_MSG_UNDER_CURSOR. Jeśli inny program usunie komunikat między wywołaniami przeglądania i usuwania, druga operacja MQGET zakończy się natychmiast niepowodzeniem (bez przeszukiwania całej kolejki), ponieważ pod kursorem przeglądania nie ma żadnego komunikatu.

3. Atrybut *MaxMsgLength kolejka* określa maksymalną długość komunikatów akceptowanych dla tej kolejki, a atrybut *MaxMsgLength menedżer kolejek* określa maksymalną długość komunikatów akceptowanych dla tego menedżera kolejek. Jeśli oczekiwana długość komunikatu nie jest znana, można zapytać o atrybut **MaxMsgLength** (przy użyciu wywołania MQINQ), a następnie określić bufor o tej wielkości.

   Aby uniknąć zmniejszenia wydajności, należy maksymalnie zbliżyć wielkość buforu do rzeczywistej wielkości komunikatu.

   Więcej informacji na temat atrybutu **MaxMsgLength** zawiera sekcja "Zwiększanie maksymalnej długości komunikatu" na stronie 817.

## *Kolejność, w jakiej komunikaty są pobierane z kolejki*

Można sterować kolejnością, w jakiej komunikaty są pobierane z kolejki. W tej sekcji przedstawiono dostępne opcje.

*Priorytet*

Program może przypisać priorytet do komunikatu podczas umieszczania komunikatu w kolejce (patrz sekcja "Priorytety komunikatów" na stronie 27 ). Komunikaty o jednakowych priorytetach są przechowywane w kolejce w kolejności ich nadejścia, a nie w kolejności, w jakiej zostały zatwierdzone.

Menedżer kolejek utrzymuje kolejki w ścisłej kolejności FIFO (pierwszy przyszedł, pierwszy wyszedł) lub w kolejności FIFO w kolejności priorytetów. Zależy to od ustawienia atrybutu **MsgDeliverySequence** kolejki. Po nadejściu komunikatu do kolejki jest on wstawiany bezpośrednio po ostatnim komunikacie, który ma ten sam priorytet.

Programy mogą albo pobrać pierwszy komunikat z kolejki, albo pobrać konkretny komunikat z kolejki, ignorując priorytet tych komunikatów. Na przykład program może chcieć przetworzyć odpowiedź na określony komunikat, który wysłał wcześniej. Więcej informacji na ten temat zawiera sekcja "Uzyskiwanie konkretnego komunikatu" na stronie 810.

Jeśli aplikacja umieszcza sekwencję komunikatów w kolejce, inna aplikacja może pobrać te komunikaty w tej samej kolejności, w jakiej zostały umieszczone, pod warunkiem, że:

- Wszystkie komunikaty mają ten sam priorytet
- Wszystkie komunikaty zostały umieszczone w tej samej jednostce pracy lub poza nią.
- Kolejka jest lokalna względem aplikacji umieszczającej

Jeśli te warunki nie są spełnione, a aplikacje zależą od komunikatów pobieranych w określonej kolejności, aplikacje muszą albo dołączyć informacje o sekwencji do danych komunikatu, albo ustanowić sposób potwierdzania odbioru komunikatu przed wysłaniem następnego komunikatu.

z/OS W systemie IBM MQ for z/OSmożna użyć atrybutu kolejki *IndexType*, aby zwiększyć szybkość operacji MQGET w kolejce. Więcej informacji na ten temat zawiera sekcja "Typ indeksu" na stronie 816.

*Porządkowanie logiczne i fizyczne*

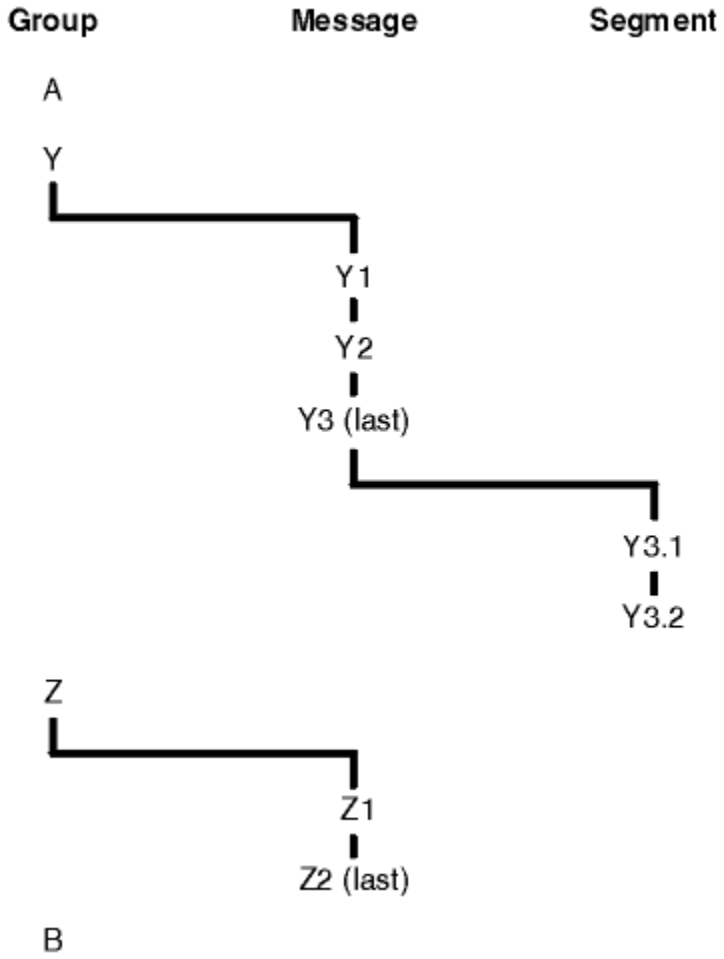
Komunikaty w kolejkach mogą występować (w obrębie każdego poziomu priorytetu) w kolejności *fizycznej* lub *logicznej* .

Kolejność fizyczna to kolejność, w jakiej komunikaty trafiają do kolejki. Porządek logiczny występuje wtedy, gdy wszystkie komunikaty i segmenty w grupie znajdują się w sekwencji logicznej obok siebie, w pozycji określonej przez pozycję fizyczną pierwszego elementu należącego do grupy.

Opis grup, komunikatów i segmentów zawiera sekcja "Grupy komunikatów" na stronie 45. Kolejność fizyczna i logiczna mogą się różnić, ponieważ:

- Grupy mogą dotrzeć do miejsca docelowego w podobnych momentach z różnych aplikacji, co oznacza utratę odrębnej kolejności fizycznej.
- Nawet w pojedynczej grupie komunikaty mogą być w nieporządku z powodu przekierowania lub opóźnienia niektórych komunikatów w grupie.

Na przykład kolejność logiczna może wyglądać następująco: Rysunek 61 na stronie 799:

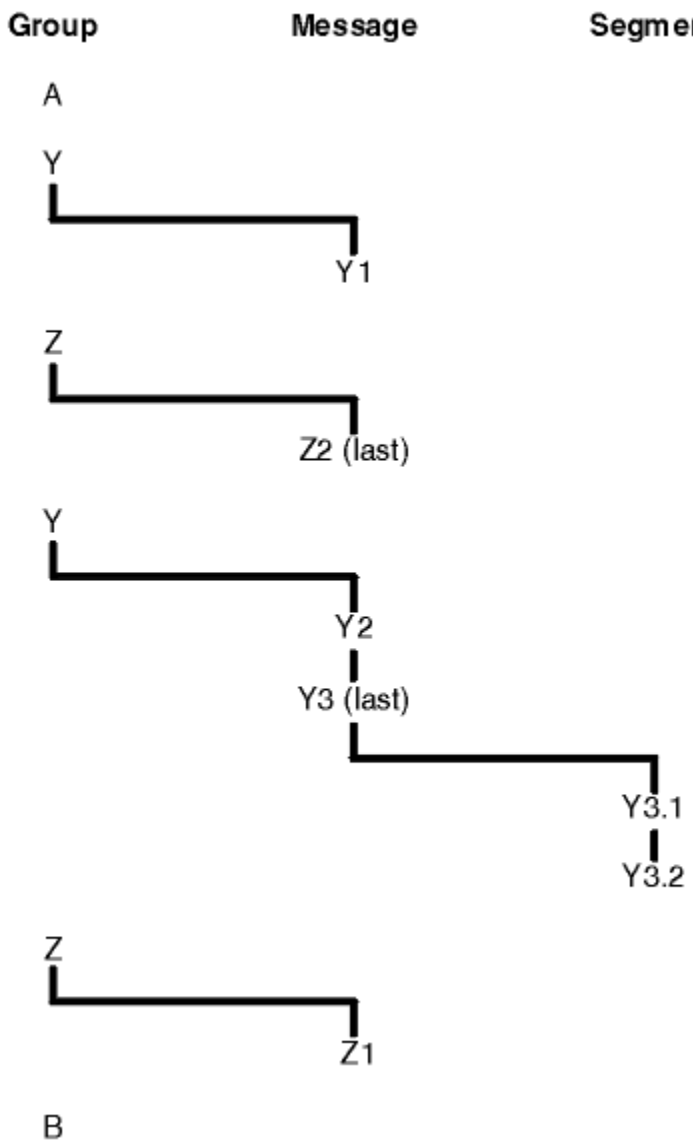



*Rysunek 61. Porządek logiczny w kolejce*

Te komunikaty mogą wystąpić w następującej kolejności logicznej w kolejce:

1. Komunikat A (nie w grupie)
2. Komunikat logiczny 1 grupy Y
3. Komunikat logiczny 2 z grupy Y
4. Segment 1 (ostatniego) komunikatu logicznego 3 grupy Y
5. (Ostatni) segment 2 (ostatniego) komunikatu logicznego 3 grupy Y
6. Komunikat logiczny 1 grupy Z
7. (Ostatni) komunikat logiczny 2 z grupy Z
8. Komunikat B (nie w grupie)

Jednak porządek fizyczny może być zupełnie inny. Pozycja fizyczna *pierwszego* elementu w każdej grupie określa pozycję logiczną całej grupy. Na przykład, jeśli grupy Y i Z pojawiły się w podobnym czasie, a komunikat 2 z grupy Z wyprzedzał komunikat 1 z tej samej grupy, kolejność fizyczna będzie wyglądać jak na rysunku Rysunek 62 na stronie 800:




*Rysunek 62. Kolejność fizyczna w kolejce*

Te komunikaty występują w następującej kolejności fizycznej w kolejce:

1. Komunikat A (nie w grupie)
2. Komunikat logiczny 1 grupy Y
3. Komunikat logiczny 2 grupy Z
4. Komunikat logiczny 2 z grupy Y
5. Segment 1 (ostatniego) komunikatu logicznego 3 grupy Y
6. (Ostatni) segment 2 (ostatniego) komunikatu logicznego 3 grupy Y
7. Komunikat logiczny 1 grupy Z
8. Komunikat B (nie w grupie)

**Uwaga:** W systemie IBM MQ for z/OSfizyczna kolejność komunikatów w kolejce nie jest gwarantowana, jeśli kolejka jest indeksowana według identyfikatora GROUPID.

Podczas pobierania komunikatów można określić parametr MQGMO_LOGICAL_ORDER, aby pobierać komunikaty w porządku logicznym, a nie w porządku fizycznym.

W przypadku wywołania MQGET z MQGMO_BROWSE_FIRST i MQGMO_LOGICAL_ORDER kolejne wywołania MQGET z MQGMO_BROWSE_NEXT muszą również określać parametr

MQGMO_LOGICAL_ORDER. I odwrotnie, jeśli komenda MQGET z opcją MQGMO_BROWSE_FIRST nie określa parametru MQGMO_LOGICAL_ORDER, nie muszą być używane następujące komendy MQGET z opcją MQGMO_BROWSE_NEXT.

Informacje o grupach i segmentach, które menedżer kolejek zachowuje dla wywołań MQGET, które przeglądają komunikaty w kolejce, są oddzielone od informacji o grupach i segmentach, które menedżer kolejek zachowuje dla wywołań MQGET, które usuwają komunikaty z kolejki. Po podaniu wartości MQGMO_BROWSE_FIRST menedżer kolejek ignoruje informacje o grupach i segmentach podczas przeglądania i skanuje kolejkę tak, jakby nie było bieżącej grupy ani bieżącego komunikatu logicznego.

**Uwaga:** Nie należy używać wywołania MQGET do przeglądania *poza koniec* grupy komunikatów (lub komunikatu logicznego, którego nie ma w grupie) bez określenia parametru MQGMO_LOGICAL_ORDER. Jeśli na przykład ostatni komunikat w grupie *poprzedza* pierwszy komunikat w grupie w kolejce, użycie opcji MQGMO_BROWSE_NEXT do przeglądania poza końcem grupy, podanie opcji MQMO_MATCH_MSG_SEQ_NUMBER z wartością *MsgSeqNumber* ustawioną na 1 (w celu znalezienia pierwszego komunikatu w następnej grupie) spowoduje ponowne zwrócenie pierwszego komunikatu w grupie, która była już przeglądana. Może to nastąpić natychmiast lub kilka wywołań MQGET później (jeśli istnieją grupy pośredniczące).

Aby uniknąć możliwości wystąpienia pętli nieskończonej, należy otworzyć kolejkę *dwukrotnie* w celu przeglądania:

- Użyj pierwszego uchwytu, aby przeglądać tylko pierwszy komunikat w każdej grupie.
- Drugi uchwyt służy do przeglądania tylko komunikatów w konkretnej grupie.
- Użyj opcji MQMO_ *, aby przenieść drugi kursor przeglądania do pozycji pierwszego kursora przeglądania przed przeglądaniem komunikatów w grupie.
- Nie należy używać opcji przeglądania MQGMO_BROWSE_NEXT poza końcem grupy.

Więcej informacji na ten temat zawierają artykuły MQGET, MQMDi Reguły sprawdzania poprawności opcji MQI.

W przypadku większości aplikacji podczas przeglądania prawdopodobnie zostanie wybrana kolejność logiczna lub fizyczna. Aby jednak przełączać się między tymi trybami, należy pamiętać, że przy pierwszym uruchomieniu przeglądania za pomocą wywołania MQGMO_LOGICAL_ORDER ustanawiana jest pozycja w sekwencji logicznej.

Jeśli pierwszy element w grupie nie jest w tym momencie obecny, grupa, w której znajduje się użytkownik, nie jest traktowana jako część sekwencji logicznej.

Gdy kursor przeglądania znajduje się w grupie, może być kontynuowany w tej samej grupie, nawet jeśli pierwszy komunikat zostanie usunięty. Początkowo jednak nie można przenieść się do grupy przy użyciu parametru MQGMO_LOGICAL_ORDER, gdy pierwszy element nie jest obecny.

**MQPMO_LOGICAL_ORDER,**

Opcja MQPMO informuje menedżer kolejek, w jaki sposób aplikacja umieszcza komunikaty w grupach i segmentach komunikatów logicznych. Można go określić tylko w wywołaniu MQPUT. Nie jest on poprawny w wywołaniu MQPUT1 .

Jeśli zostanie określona opcja MQPMO_LOGICAL_ORDER, oznacza to, że aplikacja używa kolejnych wywołań MQPUT w następujących celach:

1. Umieszczenie segmentów w każdym komunikacie logicznym w kolejności rosnącego przesunięcia segmentu, począwszy od 0, bez przerw.
2. Umieszczenie wszystkich segmentów w jednym komunikacie logicznym przed umieszczeniem segmentów w następnym komunikacie logicznym.
3. Umieszczenie komunikatów logicznych w każdej grupie komunikatów w kolejności rosnących numerów kolejnych komunikatów, począwszy od 1, bez przerw. Numer kolejny komunikatu jest zwiększany automatycznie w produkcie IBM MQ.
4. Umieszczenie wszystkich komunikatów logicznych w jednej grupie komunikatów przed umieszczeniem komunikatów logicznych w następnej grupie komunikatów.

Ponieważ aplikacja powiedziała menedżerowi kolejek, w jaki sposób umieszcza komunikaty w grupach i segmentach komunikatów logicznych, nie musi ona obsługiwać i aktualizować informacji o grupach i segmentach dotyczących każdego wywołania MQPUT, ponieważ menedżer kolejek obsługuje i aktualizuje te informacje. W szczególności oznacza to, że aplikacja nie musi ustawiać pól *GroupId*, *MsgSeqNumber*i *Offset* w strukturze MQMD, ponieważ menedżer kolejek ustawia te pola na odpowiednie wartości. Aplikacja musi tylko ustawić pole *MsgFlags* w strukturze MQMD, aby wskazać, że komunikaty należą do grup lub są segmentami komunikatów logicznych, oraz wskazać ostatni komunikat w grupie lub ostatni segment komunikatu logicznego.

Po uruchomieniu grupy komunikatów lub komunikatu logicznego kolejne wywołania MQPUT muszą określać odpowiednie flagi MQMF_ * w pliku *MsgFlags* w strukturze MQMD. Jeśli aplikacja próbuje umieścić komunikat, który nie jest w grupie, gdy istnieje niezakończona grupa komunikatów, lub umieścić komunikat, który nie jest segmentem, gdy istnieje niezakończony komunikat logiczny, wywołanie kończy się niepowodzeniem z kodem przyczyny MQRC_INCOMPLETE_GROUP lub MQRC_INCOMPLETE_MSG, w zależności od przypadku. Jednak menedżer kolejek zachowuje informacje o bieżącej grupie komunikatów lub bieżącym komunikacie logicznym, a aplikacja może je zakończyć, wysyłając komunikat (być może bez danych komunikatu aplikacji), określając odpowiednio MQMF_LAST_MSG_IN_GROUP lub MQMF_LAST_SEGMENT przed ponownym wywołaniem MQPUT w celu umieszczenia komunikatu, który nie znajduje się w grupie lub nie jest segmentem.

Rysunek 62 na stronie 800 przedstawia kombinacje poprawnych opcji i flag oraz wartości pól *GroupId*, *MsgSeqNumber*i *Offset* używane przez menedżer kolejek w każdym przypadku. Kombinacje opcji i flag, które nie są wyświetlane w tabeli, są niepoprawne. Kolumny w tabeli mają następujące znaczenie: Tak lub Nie:

**SŁOWO PROTOKOŁU**
Określa, czy opcja MQPMO_LOGICAL_ORDER jest określona w wywołaniu.

**MIG**
Określa, czy w wywołaniu podano opcję MQMF_MSG_IN_GROUP lub MQMF_LAST_MSG_IN_GROUP.

**SEG**
Określa, czy w wywołaniu podano opcję MQMF_SEGMENT lub MQMF_LAST_SEGMENT.

**SEG OK**
Określa, czy opcja MQMF_SEGMENTATION_ALLOWED jest określona w wywołaniu.

**Cur grp**
Określa, czy przed wywołaniem istnieje bieżąca grupa komunikatów.

**Komunikat dziennika Cur**
Określa, czy przed wywołaniem istnieje bieżący komunikat logiczny.

**Inne kolumny**
Pokaż wartości używane przez menedżer kolejek. Poprzedni oznacza wartość używaną dla pola w poprzednim komunikacie dla uchwytu kolejki.

*Tabela 116. Opcje MQPUT dotyczące komunikatów w grupach i segmentach komunikatów logicznych*

| **Określone opcje** | **Określone opcje** | **Określone opcje** | **Określone opcje** | **Status grupy i komunikatu dziennika przed wywołaniem** | **Status grupy i komunikatu dziennika przed wywołaniem** | **Wartości używane przez menedżer kolejek** | **Wartości używane przez menedżer kolejek** | **Wartości używane przez menedżer kolejek** |
|---|---|---|---|---|---|---|---|---|
| SŁOWO PROTOKOŁU | MIG | SEG | SEG OK | Cur grp | Komunikat dziennika Cur | **GroupId** | **MsgSeqNumber** | **Offset** |
| Tak | Nie | Nie | Nie | Nie | Nie | MQGI_NONE | 1 | 0 |
| Tak | Nie | Nie | Tak | Nie | Nie | Nowy identyfikator grupy | 1 | 0 |
| Tak | Nie | Tak | Albo | Nie | Nie | Nowy identyfikator grupy | 1 | 0 |
| Tak | Nie | Tak | Albo | Nie | Tak | Poprzedni identyfikator grupy | 1 | Poprzednie przesunięcie + poprzednia długość segmentu |
| Tak | Tak | Albo | Albo | Nie | Nie | Nowy identyfikator grupy | 1 | 0 |
| Tak | Tak | Albo | Albo | Tak | Nie | Poprzedni identyfikator grupy | Poprzedni numer kolejny + 1 | 0 |
| Tak | Tak | Tak | Albo | Tak | Tak | Poprzedni identyfikator grupy | Poprzedni numer kolejny | Poprzednie przesunięcie + poprzednia długość segmentu |
| Nie | Nie | Nie | Nie | Albo | Albo | MQGI_NONE | 1 | 0 |
| Nie | Nie | Nie | Tak | Albo | Albo | Nowy identyfikator grupy, jeśli MQGI_NONE, w przeciwnym razie wartość w polu | 1 | 0 |
| Nie | Nie | Tak | Albo | Albo | Albo | Nowy identyfikator grupy, jeśli MQGI_NONE, w przeciwnym razie wartość w polu | 1 | Wartość w polu |

*Tabela 116. Opcje MQPUT dotyczące komunikatów w grupach i segmentach komunikatów logicznych (kontynuacja)*

| Określone opcje | Określone opcje | Określone opcje | Określone opcje | Status grupy i komunikatu dziennika przed wywołaniem | Status grupy i komunikatu dziennika przed wywołaniem | Wartości używane przez menedżer kolejek | Wartości używane przez menedżer kolejek | Wartości używane przez menedżer kolejek |
|---|---|---|---|---|---|---|---|---|
| Nie | Tak | Nie | Albo | Albo | Albo | Nowy identyfikator grupy, jeśli MQGI_NONE, w przeciwnym razie wartość w polu | Wartość w polu | 0 |
| Nie | Tak | Tak | Albo | Albo | Albo | Nowy identyfikator grupy, jeśli MQGI_NONE, w przeciwnym razie wartość w polu | Wartość w polu | Wartość w polu |

**Uwaga:**

- Parametr MBQPMO_LOGICAL_ORDER jest niepoprawny w wywołaniu MQPUT1 .
- W przypadku pola *MsgId* menedżer kolejek generuje nowy identyfikator komunikatu, jeśli określono parametr MQPMO_NEW_MSG_ID lub MQMI_NONE, i w przeciwnym razie używa wartości z pola.
- W przypadku pola *CorrelId* menedżer kolejek generuje nowy identyfikator korelacji, jeśli określono parametr MQPMO_NEW_CORREL_ID i w przeciwnym razie używa wartości z tego pola.

Jeśli określono parametr MQPMO_LOGICAL_ORDER, menedżer kolejek wymaga, aby wszystkie komunikaty w grupie i segmenty w komunikacie logicznym były umieszczane z taką samą wartością w polu *Persistence* w strukturze MQMD, co oznacza, że wszystkie komunikaty muszą być trwałe lub wszystkie muszą być nietrwałe. Jeśli ten warunek nie jest spełniony, wywołanie MQPUT kończy się niepowodzeniem z kodem przyczyny MQRC_INCONSISTENT_PERSISTENCE.

Opcja MQPMO_LOGICAL_ORDER wpływa na jednostki pracy w następujący sposób:

- Jeśli pierwszy komunikat fizyczny w grupie lub komunikacie logicznym jest umieszczany w jednostce pracy, wszystkie pozostałe komunikaty fizyczne w grupie lub komunikacie logicznym muszą być umieszczane w jednostce pracy, jeśli używany jest ten sam uchwyt kolejki. Nie muszą one jednak być umieszczane w tej samej jednostce pracy, co umożliwia podzielenie grupy komunikatów lub komunikatu logicznego składającego się z wielu komunikatów fizycznych na dwie lub więcej kolejnych jednostek pracy dla uchwytu kolejki.
- Jeśli pierwszy komunikat fizyczny w grupie lub komunikacie logicznym nie zostanie umieszczony w jednostce pracy, żaden inny komunikat fizyczny w grupie lub komunikacie logicznym nie może zostać umieszczony w jednostce pracy, jeśli używany jest ten sam uchwyt kolejki.

Jeśli te warunki nie są spełnione, wywołanie MQPUT kończy się niepowodzeniem z kodem przyczyny MQRC_INCONSISTENT_UOW.

Jeśli określono parametr MQPMO_LOGICAL_ORDER, wartość MQMD podana w wywołaniu MQPUT nie może być mniejsza niż MQMD_VERSION_2. Jeśli ten warunek nie jest spełniony, wywołanie kończy się niepowodzeniem z kodem przyczyny MQRC_WRONG_MD_VERSION.

Jeśli parametr MQPMO_LOGICAL_ORDER nie jest określony, komunikaty w grupach i segmentach komunikatów logicznych mogą być umieszczane w dowolnej kolejności i nie jest konieczne umieszczanie kompletnych grup komunikatów ani kompletnych komunikatów logicznych. Odpowiedzialność za to, aby pola *GroupId*, *MsgSeqNumber*, *Offset*i *MsgFlags* miały odpowiednie wartości, spoczywa na aplikacji.

Ta technika służy do restartowania grupy komunikatów lub komunikatu logicznego w środku po wystąpieniu awarii systemu. Po zrestartowaniu systemu aplikacja może ustawić odpowiednie wartości w polach *GroupId*, *MsgSeqNumber*, *Offset*, *MsgFlags*si *Persistence* , a następnie wywołać wywołanie MQPUT z opcją MQPMO_SYNCPOINT lub MQPMO_NO_SYNCPOINT zgodnie z wymaganiami, ale bez określania parametru MQPMO_LOGICAL_ORDER. Jeśli wywołanie zakończy się pomyślnie, menedżer kolejek zachowa informacje o grupie i segmencie, a kolejne wywołania MQPUT używające tego uchwytu kolejki będą mogły normalnie określać parametr MQPMO_LOGICAL_ORDER.

Informacje o grupie i segmencie, które menedżer kolejek przechowuje dla wywołania MQPUT, są oddzielone od informacji o grupie i segmencie, które przechowuje dla wywołania MQGET.

Dla dowolnego uchwytu kolejki aplikacja może łączyć wywołania MQPUT, które określają MQPMO_LOGICAL_ORDER z wywołaniami MQPUT, które nie są, ale należy zwrócić uwagę na następujące punkty:

- Jeśli parametr MQPMO_LOGICAL_ORDER nie jest określony, każde pomyślne wywołanie MQPUT powoduje, że menedżer kolejek ustawia informacje o grupie i segmencie dla uchwytu kolejki na wartości określone przez aplikację, zastępując istniejące informacje o grupie i segmencie zachowane przez menedżer kolejek dla uchwytu kolejki.
- Jeśli parametr MQPMO_LOGICAL_ORDER nie jest określony, wywołanie nie kończy się niepowodzeniem, jeśli istnieje bieżąca grupa komunikatów lub komunikat logiczny; wywołanie może zakończyć się powodzeniem z kodem zakończenia MQCC_WARNING. Tabela 117 na stronie 805 przedstawia różne obserwacje, które mogą wystąpić. W takich przypadkach, jeśli kod zakończenia jest inny niż MQCC_OK, kod przyczyny jest jednym z następujących (w zależności od przypadku):
  - MQRC_INCOMPLETE_GROUP
  - MQRC_INCOMPLETE_MSG
  - MQRC_INCONSISTENT_PERSISTENCE
  - MQRC_INCONSISTENT_UOW,

  **Uwaga:** Menedżer kolejek nie sprawdza informacji o grupie i segmencie dla wywołania MQPUT1 .

*Tabela 117. Wynik, gdy wywołanie MQPUT lub MQCLOSE nie jest spójne z informacjami o grupie i segmencie*

| Bieżące połączenie to | Poprzednie wywołanie to MQPUT z MQPMO_LOGICAL_ORDER | Poprzednie wywołanie to MQPUT bez MQPMO_LOGICAL_ORDER |
|---|---|---|
| MQPUT z MQPMO_LOGICAL_ORDER | MQCC_FAILED (niepowodzenie MQC) | MQCC_FAILED (niepowodzenie MQC) |
| MQPUT bez MQPMO_LOGICAL_ORDER | Ostrzeżenie MQCC | MQCC_OK |
| MQCLOSE z niezakończoną grupą lub komunikatem logicznym | Ostrzeżenie MQCC | MQCC_OK |

W przypadku aplikacji, które umieszczają komunikaty i segmenty w porządku logicznym, należy określić parametr MQPMO_LOGICAL_ORDER, ponieważ jest to najprostsza opcja do użycia. Ta opcja zwalnia z konieczności zarządzania informacjami o grupach i segmentach, ponieważ menedżer kolejek zarządza tymi informacjami. Jednak wyspecjalizowane aplikacje mogą wymagać większej kontroli

niż ta, która jest dostępna w przypadku opcji MQPMO_LOGICAL_ORDER, co można osiągnąć, nie określając tej opcji. W takim przypadku należy upewnić się, że pola *GroupId*, *MsgSeqNumber*, *Offset*i *MsgFlags* w deskryptorze MQMD zostały ustawione poprawnie przed każdym wywołaniem MQPUT lub MQPUT1 .

Na przykład aplikacja, która chce przekazywać odebrane komunikaty fizyczne, bez względu na to, czy znajdują się one w grupach, czy w segmentach komunikatów logicznych, nie może określać parametru MQPMO_LOGICAL_ORDER z dwóch powodów:

- Jeśli komunikaty są pobierane i umieszczane w kolejności, określenie parametru MQPMO_LOGICAL_ORDER powoduje przypisanie do komunikatów nowego identyfikatora grupy, co może utrudnić lub uniemożliwić autorowi komunikatów korelację jakichkolwiek komunikatów odpowiedzi lub raportów, które wynikają z grupy komunikatów.
- W złożonej sieci z wieloma ścieżkami między wysyłającymi i odbierającymi menedżerami kolejek komunikaty fizyczne mogą pojawiać się w nieodpowiedniej kolejności. Jeśli w wywołaniu MQGET nie zostaną określone parametry MQPMO_LOGICAL_ORDER i MQGMO_LOGICAL_ORDER, aplikacja przekazująca może pobrać i przekazać każdy komunikat fizyczny natychmiast po nadejściu komunikatu, nie czekając na kolejny komunikat w kolejności logicznej.

Aplikacje generujące komunikaty raportów dla komunikatów w grupach lub segmentach komunikatów logicznych nie mogą również określać parametru MQPMO_LOGICAL_ORDER podczas umieszczania komunikatu raportu.

Parametr MQPMO_LOGICAL_ORDER można określić z dowolną inną opcją MQPMO_ *.

## Umieszczanie grup uporządkowanych logicznie w kolejce w klastrze (MQOO_BIND_ON_GROUP)

Opcja MQOO_BIND_ON_OPEN zapewnia, że wszystkie komunikaty z tej aplikacji, a więc wszystkie grupy, są kierowane do pojedynczej instancji. Jest to wadą, że ruch w aplikacji nie jest równoważony w wielu instancjach kolejki klastra. Aby włączyć równoważenie obciążenia, zachowując nienaruszone grupy komunikatów, należy ustawić następujące opcje:

- Wywołanie MQPUT musi określać parametr MQPMO_LOGICAL_ORDER
- Wywołanie MQOPEN musi określać jedną z następujących dwóch opcji:
  - MQOO_BIND_ON_GROUP
  - MQOO_BIND_AS_Q_DEF i definicja kolejki musi określać DEFBIND (GROUP)

Równoważenie obciążenia jest następnie sterowane *między grupami* komunikatów bez konieczności wykonywania operacji MQCLOSE i MQOPEN dla kolejki. *Między grupami* oznacza, że parametr MQMF_MSG_IN_GROUP jest ustawiony w deskryptorze MQMD (v2) lub MQMDE i nie ma częściowo kompletnej grupy w toku. Gdy grupa jest w toku, rozstrzygnięty menedżer kolejek i nazwa kolejki w uchwycie obiektu są ponownie wykorzystywane.

Jeśli poprzedni komunikat to MQPMO_LOGICAL_ORDER i/lub MQMF_MSG_IN_GROUP, ale bieżący komunikat nie jest częścią grupy, wywołanie PUT kończy się niepowodzeniem z komunikatem MQRC_INCOMPLETE_GROUP.

Jeśli w pojedynczej operacji MQPUT nie określono parametru MQPMO_LOGICAL_ORDER, a żadna bieżąca grupa nie jest aktywna, dla tego komunikatu sterowane jest równoważenie obciążenia (tak, jakby w wywołaniu MQOPEN określono parametr MQOO_BIND_NOT_FIXED).

Ponowne przydzielanie komunikatów powiązanych z miejscem docelowym przy użyciu opcji MQOO_BIND_ON_GROUP nie jest wykonywane. Więcej informacji na temat ponownego przydzielania zawiera sekcja “Grupy komunikatów” na stronie 45.

*Grupowanie komunikatów logicznych*

Istnieją dwie główne przyczyny używania komunikatów logicznych w grupie:

- Może być konieczne przetworzenie komunikatów w określonej kolejności.

- Konieczne może być przetworzenie każdego komunikatu w grupie w pokrewny sposób.

W obu przypadkach należy pobrać całą grupę przy użyciu tej samej instancji aplikacji.

Załóżmy na przykład, że grupa składa się z czterech komunikatów logicznych. Aplikacja zostanie wstawiona w następujący sposób:

```
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
```

Aplikacja pobierająca określa opcję MQGMO_ALL_MSGS_AVAILABLE dla pierwszego komunikatu w grupie. Dzięki temu przetwarzanie nie zostanie rozpoczęte do momentu pojawienia się wszystkich komunikatów w grupie. Opcja MQGMO_ALL_MSGS_AVAILABLE jest ignorowana dla kolejnych komunikatów w grupie.

Po pobraniu pierwszego komunikatu logicznego grupy można użyć parametru MQGMO_LOGICAL_ORDER, aby upewnić się, że pozostałe komunikaty logiczne grupy są pobierane w odpowiedniej kolejności.

Tak więc, aplikacja do pobierania wygląda następująco:

```
/* 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
```

Dalsze przykłady grupowania komunikatów znajdują się w sekcji “Segmentacja aplikacji dla komunikatów logicznych” na stronie 819 i “Tworzenie i tworzenie grupy, która obejmuje jednostki pracy” na stronie 807.

**Ostrzeżenie:** W przypadku używania publikowania/subskrypcji do wysyłania komunikatów do tematu (lub umieszczania komunikatów w aliasach tematów) grupowanie i segmentacja komunikatów nie jest dozwolona.

Ponieważ subskrypcje mogą być tworzone i usuwane niezależnie od działania publikowania, nie można zapewnić, że subskrybent otrzyma pełną grupę komunikatów lub wszystkie segmenty komunikatu. Patrz RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP.

Informacje na temat zezwalania aplikacji na żądanie przydzielenia grupy komunikatów do tej samej instancji docelowej dla kolejek klastra zawiera sekcja DefBind.

*Tworzenie i tworzenie grupy, która obejmuje jednostki pracy*

W poprzednim przypadku komunikaty lub segmenty nie mogą opuścić węzła (jeśli jego miejsce docelowe jest zdalne) ani nie mogą zostać pobrane, dopóki cała grupa nie zostanie wstawiona i jednostka pracy nie zostanie zatwierdzona. Może to być niepotrzebne, jeśli umieszczenie całej grupy zajmuje dużo czasu lub jeśli miejsce w kolejce jest ograniczone w węźle. Aby to przezwyciężyć, należy umieścić grupę w kilku jednostkach pracy.

Jeśli grupa jest umieszczana w wielu jednostkach pracy, możliwe jest, że część grupy zatwierdzi ją nawet wtedy, gdy aplikacja nie powiedzie się. Dlatego aplikacja musi zapisać informacje o statusie, zatwierdzone dla każdej jednostki pracy, których może użyć po restarcie, aby wznowić niekompletną grupę. Najprostszym miejscem zapisywania tych informacji jest kolejka STATUS. Jeśli pełna grupa została pomyślnie umieszczona, kolejka STATUS jest pusta.

Jeśli używana jest segmentacja, logika jest podobna. W tym przypadku **StatusInfo** musi zawierać *Offset*.

Oto przykład umieszczenia grupy w kilku jednostkach pracy:

```
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
```

Jeśli wszystkie jednostki pracy zostały zatwierdzone, cała grupa została pomyślnie umieszczona, a kolejka STATUS jest pusta. Jeśli nie, grupa musi zostać wznowiona w punkcie wskazanym przez informacje o statusie. Instrukcja MMQPMO_LOGICAL_ORDER nie może być używana dla pierwszej operacji umieszczania, ale może być używana później.

Przetwarzanie restartu wygląda następująco:

```
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
```

Z poziomu aplikacji pobierającej można rozpocząć przetwarzanie komunikatów w grupie przed nadejściem całej grupy. Poprawia to czasy odpowiedzi dla komunikatów w grupie, a także oznacza, że pamięć masowa nie jest wymagana dla całej grupy. W celu uzyskania korzyści należy użyć kilku jednostek pracy dla każdej grupy komunikatów. W celu odzyskiwania konieczne jest pobranie każdego komunikatu w ramach jednostki pracy.

Podobnie jak w przypadku odpowiedniej aplikacji umieszczającej, wymaga to, aby informacje o statusie były zapisywane w dowolnym miejscu automatycznie po zatwierdzeniu każdej jednostki pracy. Ponownie najprostszym miejscem, w którym można zapisać te informacje, jest kolejka STATUS. Jeśli kompletna grupa została pomyślnie przetworzona, kolejka STATUS jest pusta.

**Uwaga:** W przypadku pośrednich jednostek pracy można uniknąć wywołań MQGET z kolejki STATUS, określając, że każda operacja MQPUT do kolejki statusu jest segmentem komunikatu (czyli przez ustawienie flagi MQMF_SEGMENT), zamiast umieszczania kompletnego nowego komunikatu dla każdej jednostki pracy. W ostatniej jednostce pracy ostatni segment jest umieszczany w kolejce statusów określającej parametr MQMF_LAST_SEGMENT, a następnie informacje o statusie są czyszczone za pomocą komendy MQGET określającej parametr MQGMO_COMPLETE_MSG.

Podczas przetwarzania restartu, zamiast korzystać z pojedynczego wywołania MQGET w celu uzyskania możliwego komunikatu o statusie, przejrzyj kolejkę statusu za pomocą wywołania MQGMO_LOGICAL_ORDER, aż do osiągnięcia ostatniego segmentu (czyli do momentu, gdy nie zostaną zwrócone żadne dalsze segmenty). W pierwszej jednostce pracy po restarcie należy również jawnie określić przesunięcie podczas umieszczania segmentu statusu.

W poniższym przykładzie rozważane są tylko komunikaty w grupie, przy założeniu, że bufor aplikacji jest zawsze wystarczająco duży, aby pomieścić cały komunikat, niezależnie od tego, czy komunikat został podzielony na segmenty. Dlatego w każdej operacji MQGET określono parametr MQGMO_COMPLETE_MSG. Te same zasady mają zastosowanie w przypadku segmentacji (w tym przypadku element StatusInfo musi zawierać *Offset*).

Dla uproszczenia założono, że w pojedynczej jednostce pracy pobierane są maksymalnie 4 komunikaty:

```
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
```

Jeśli wszystkie jednostki pracy zostały zatwierdzone, cała grupa została pomyślnie pobrana, a kolejka STATUS jest pusta. Jeśli nie, grupa musi zostać wznowiona w punkcie wskazanym przez informacje o statusie. Parametr MQGMO_LOGICAL_ORDER nie może być używany podczas pierwszego pobierania, ale może być używany później.

Przetwarzanie restartu wygląda następująco:

```
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
```

### *Uzyskiwanie konkretnego komunikatu*

Istnieje wiele sposobów pobierania określonego komunikatu z kolejki. Są to: wybór w polu MsgId i CorrelId, wybór w polu GroupId, MsgSeqNumer i przesunięcie oraz wybór w polu MsgToken. Podczas otwierania kolejki można również użyć łańcucha wyboru.

Aby pobrać konkretny komunikat z kolejki, należy użyć pól `MsgId` i `CorrelId` struktury MQMD. Jednak aplikacje mogą jawnie ustawić te pola, dlatego podane wartości mogą nie identyfikować unikalnego komunikatu. Tabela 118 na stronie 810 pokazuje, który komunikat jest pobierany dla możliwych ustawień tych pól. Te pola są ignorowane w danych wejściowych, jeśli w parametrze **`GetMsgOpts`** wywołania MQGET określono wartość MQGMO_MSG_UNDER_CURSOR.

*Tabela 118. Korzystanie z identyfikatorów komunikatów i korelacji*

| **Aby pobrać ...** | **`MsgId`** | **`CorrelId`** |
|---|---|---|
| Pierwszy komunikat w kolejce | `MQMI_NONE` | `MQCI_NONE` |
| Pierwszy komunikat zgodny z *`MsgId`* | Niezerowy | `MQCI_NONE` |
| Pierwszy komunikat zgodny z *`CorrelId`* | `MQMI_NONE` | Niezerowy |
| Pierwszy komunikat, który jest zgodny zarówno z *`MsgId`* , jak i *`CorrelId`* | Niezerowy | Niezerowy |

W każdym przypadku *pierwszy* oznacza pierwszy komunikat, który spełnia kryteria wyboru (chyba że określono opcję MQGMO_BROWSE_NEXT, gdy oznacza to, że *następny* komunikat w sekwencji spełnia kryteria wyboru).

Podczas zwracania wywołania MQGET ustawia pola *`MsgId`* i *`CorrelId`* na identyfikatory komunikatu i korelacji zwróconego komunikatu (jeśli istnieją).

Jeśli pole *`Version`* struktury MQMD zostanie ustawione na wartość 2, można użyć pól *`GroupId`*, *`MsgSeqNumber`*i *`Offset`* . Tabela 119 na stronie 810 pokazuje, który komunikat jest pobierany dla możliwych ustawień tych pól.

*Tabela 119. Korzystanie z identyfikatora grupy*

| **Aby pobrać ...** | **opcje dopasowywania** |
|---|---|
| Pierwszy komunikat w kolejce | MQMO_BRAK |
| Pierwszy komunikat zgodny z `MsgId` | MQMO_MATCH_MSG_ID |

| *Tabela 119. Korzystanie z identyfikatora grupy (kontynuacja)* | |
|---|---|
| **Aby pobrać ...** | **opcje dopasowywania** |
| Pierwszy komunikat zgodny z `CorrelId` | MQMO_MATCH_CORREL_ID |
| Pierwszy komunikat zgodny z `GroupId` | MQMO_MATCH_GROUP_ID (identyfikator grupy zgodności MQ) |
| Pierwszy komunikat zgodny z `MsgSeqNumber` | MQMO_MATCH_MSG_SEQ_NUMBER |
| Pierwszy komunikat zgodny z `MsgToken` | MQMO_MATCH_MSG_TOKEN, |
| Pierwszy komunikat zgodny z `Offset` | MQMO_ZGODNE_PRZESUNIĘCIE |

**Uwagi:**

1. MQMO_MATCH_XXX oznacza, że pole *XXX* w strukturze MQMD jest ustawione na wartość, która ma być zgodna.
2. Flagi MQMO mogą być używane łącznie. Na przykład wartości MQMO_MATCH_GROUP_ID, MQMO_MATCH_MSG_SEQ_NUMBER i MQMO_MATCH_OFFSET mogą być używane razem w celu nadania segmentowi identyfikowanego przez pola `GroupId`, `MsgSeqNumber`i `Offset` .
3. Jeśli określono parametr MQGMO_LOGICAL_ORDER, wpływa to na komunikat, który ma zostać pobrany, ponieważ ta opcja zależy od informacji o stanie sterowanych dla uchwytu kolejki. Więcej informacji na ten temat zawierają "Porządkowanie logiczne i fizyczne" na stronie 798 i Opcje.

Wywołanie MQGET zwykle pobiera pierwszy komunikat z kolejki. Jeśli podczas używania wywołania MQGET zostanie określony konkretny komunikat, menedżer kolejek musi przeszukać kolejkę, dopóki nie znajdzie tego komunikatu. Może to mieć wpływ na wydajność aplikacji.

Jeśli używana jest struktura MQGMO w wersji 2 lub nowszej i nie zostaną podane flagi MQMO_MATCH_MSG_ID lub MQMO_MATCH_CORREL_ID, nie trzeba resetować pól `MsgId` lub `CorrelId` między operacjami MQGET.

z/OS W systemie IBM MQ for z/OSdo zwiększenia szybkości operacji MQGET w kolejce można użyć atrybutu kolejki `IndexType` . Więcej informacji na ten tematzawiera sekcja "Typ indeksu" na stronie 816.

Konkretny komunikat można pobrać z kolejki, określając jego MsgToken i MatchOption MQMO_MATCH_MSG_TOKEN w strukturze MQGMO. Element MsgToken jest zwracany przez wywołanie MQPUT, które pierwotnie umieściło ten komunikat w kolejce, lub przez poprzednie operacje MQGET i pozostaje stały, chyba że menedżer kolejek zostanie zrestartowany.

Jeśli użytkownik jest zainteresowany tylko podzbiorem komunikatów w kolejce, może określić, które komunikaty mają być przetwarzane, używając łańcucha wyboru z wywołaniem MQOPEN lub MQSUB. Następnie komenda MQGET pobiera następny komunikat, który spełnia warunki tego łańcucha wyboru. Więcej informacji na temat łańcuchów wyboru zawiera sekcja "Selektory" na stronie 31.

## *Zwiększanie wydajności komunikatów nietrwałych*

Gdy klient wymaga komunikatu od serwera, wysyła żądanie do serwera. Wysyła osobne żądanie dla każdego komunikatu, który konsumuje. Aby zwiększyć wydajność klienta korzystającego z nietrwałych komunikatów, unikając konieczności wysyłania tych komunikatów żądań, można skonfigurować klienta w taki sposób, aby używał *odczytu z wyprzedzeniem*. Odczyt z wyprzedzeniem umożliwia wysyłanie komunikatów do klienta bez konieczności żądania ich przez aplikację.

Gdy odczyt z wyprzedzeniem jest włączony, komunikaty są wysyłane do buforu pamięci na kliencie nazywanego *buforem odczytu z wyprzedzeniem*. Klient będzie miał bufor odczytu z wyprzedzeniem dla każdej otwartej kolejki z włączonym odczytem z wyprzedzeniem. Komunikaty w buforze odczytu z wyprzedzeniem nie są utrwalane. Klient okresowo aktualizuje serwer informacjami o ilości danych, które wykorzystał.

W przypadku wywołania MQOPEN z opcją MQOO_READ_AHEAD klient IBM MQ umożliwia odczyt z wyprzedzeniem, jeśli spełnione są pewne warunki. Są one następujące:

- Aplikacja kliencka musi zostać skompilowana i powiązana z wątkowymi bibliotekami klienta MQI IBM MQ.
- Kanał klienta musi używać protokołu TCP/IP.
- Ustawienie SharingConversations (SHARECNV) kanału musi mieć wartość niezerową w definicji kanału zarówno klienta, jak i serwera.

Użycie odczytu z wyprzedzeniem może zwiększyć wydajność podczas konsumowania nietrwałych komunikatów z aplikacji klienckiej. Ten wzrost wydajności jest dostępny zarówno dla aplikacji MQI, jak i JMS . Aplikacje klienckie korzystające z usługi MQGET lub z wykorzystania asynchronicznego będą korzystać ze zwiększonej wydajności podczas korzystania z nietrwałych komunikatów.

Nie wszystkie projekty aplikacji klienckich nadają się do korzystania z odczytu z wyprzedzeniem, ponieważ nie wszystkie opcje są obsługiwane w przypadku odczytu z wyprzedzeniem, a niektóre opcje muszą być spójne między wywołaniami MQGET, gdy włączony jest odczyt z wyprzedzeniem. Jeśli klient zmienia kryteria wyboru między wywołaniami MQGET, komunikaty przechowywane w buforze odczytu z wyprzedzeniem pozostaną porzucone w buforze odczytu z wyprzedzeniem klienta.

Jeśli dziennik komunikatów osieroconych z poprzednimi kryteriami wyboru nie jest już wymagany, można ustawić w kliencie konfigurowalny odstęp czasu między operacjami czyszczenia, aby te komunikaty były automatycznie usuwane z klienta. Odstęp czasu czyszczenia jest jedną z grup opcji strojenia odczytu z wyprzedzeniem określonych przez klienta. Opcje te można dostroić, aby spełnić wymagania użytkownika.

Jeśli aplikacja kliencka zostanie zrestartowana, komunikaty w buforze odczytu z wyprzedzeniem mogą zostać utracone. Z kolei komunikat, który został przeniesiony do buforu odczytu z wyprzedzeniem, może zostać usunięty z kolejki bazowej. Nie spowoduje to usunięcia go z buforu, więc wywołanie MQGET używające odczytu z wyprzedzeniem może zwrócić komunikat, który już nie istnieje.

Odczyt z wyprzedzeniem jest wykonywany tylko dla powiązań klienta. Atrybut jest ignorowany dla wszystkich innych powiązań.

Odczyt z wyprzedzeniem nie ma wpływu na wyzwalanie. Komunikat wyzwalacza nie jest generowany, gdy komunikat jest odczytywany z wyprzedzeniem przez klienta. Funkcja odczytu z wyprzedzeniem nie generuje informacji o rozliczeniach i statystykach, gdy jest włączona.

## Korzystanie z odczytu z wyprzedzeniem w przypadku przesyłania komunikatów w trybie publikowania i subskrybowania

Jeśli aplikacja subskrybująca określa kolejkę docelową, do której są wysyłane publikacje, wartość DEFREADA określonej kolejki jest używana jako domyślna wartość odczytu z wyprzedzeniem.

Gdy aplikacja subskrybująca żąda, aby program IBM MQ zarządzł miejscem docelowym, do którego są wysyłane publikacje, tworzona jest kolejka zarządzana jako kolejka dynamiczna oparta na predefiniowanej kolejce modelowej. Jest to wartość DEFREADA kolejki modelowej, która jest używana jako domyślna wartość odczytu z wyprzedzeniem. Domyślny model kolejki SYSTEM.DURABLE.PUBLICATIONS.MODEL lub SYSTEM.NONDURABLE.PUBLICATIONS.MODEL są używane, chyba że zdefiniowano kolejkę modelową dla tego tematu lub tematu nadrzędnego.

**Pojęcia pokrewne**

“Strojenie wydajności komunikatów nietrwałych w systemie AIX” na stronie 815
Jeśli używany jest system AIX V5.3 lub nowszej, należy rozważyć ustawienie parametru strojenia w celu użycia pełnej wydajności dla komunikatów nietrwałych.

**Zadania pokrewne**

“Włączanie i wyłączanie odczytu z wyprzedzeniem” na stronie 814
Domyślnie odczyt z wyprzedzeniem jest wyłączony. Odczyt z wyprzedzeniem można włączyć na poziomie kolejki lub aplikacji.

**Odsyłacze pokrewne**

"Opcje MQGET i odczyt z wyprzedzeniem" na stronie 813
Nie wszystkie opcje MQGET są obsługiwane, gdy włączona jest funkcja odczytu z wyprzedzeniem. Niektóre opcje muszą być spójne między wywołaniami MQGET.

*Opcje MQGET i odczyt z wyprzedzeniem*

Nie wszystkie opcje MQGET są obsługiwane, gdy włączona jest funkcja odczytu z wyprzedzeniem. Niektóre opcje muszą być spójne między wywołaniami MQGET.

W przypadku wywołania MQOPEN z opcją MQOO_READ_AHEAD klient IBM MQ umożliwia odczyt z wyprzedzeniem, jeśli spełnione są pewne warunki. Są one następujące:

- Aplikacja kliencka musi zostać skompilowana i powiązana z wątkowymi bibliotekami klienta MQI IBM MQ.
- Kanał klienta musi używać protokołu TCP/IP.
- Ustawienie SharingConversations (SHARECNV) kanału musi mieć wartość niezerową w definicji kanału zarówno klienta, jak i serwera.

Poniższa tabela zawiera informacje o obsługiwanych opcjach odczytu z wyprzedzeniem oraz o tym, czy można je zmieniać między wywołaniami MQGET.

*Tabela 120. Opcje MQGET i odczyt z wyprzedzeniem*

| **Wartości i opcje MQGET** | **Dozwolone, gdy odczyt z wyprzedzeniem jest włączony i można go zmieniać między wywołaniami MQGET** [5] | **Dozwolone, gdy odczyt z wyprzedzeniem jest włączony, ale nie można go zmienić między wywołaniami MQGET** [1] | **Opcje MQGET, które nie są dozwolone, gdy włączony jest odczyt z wyprzedzeniem** [2] |
|---|---|---|---|
| Wartości MQGET MQMD | MsgId [3]<br>CorrelId [3] | Kodowanie<br>CodedCharSetId | |
| Opcje MQGET MQGMO | • MQGMO_NO_WAIT<br>• MQGMO_BROWSE_MESSAGE _UNDER_CURSOR<br>• MQGMO_BROWSE_FIRST<br>• MQGMO_BROWSE_NEXT<br>• MQGMO_FAIL_IF_QUIESCING, | • MQGMO_SYNCPOINT_IF TRWAŁY<br>• MQGMO_NO_SYNCPOINT<br>• MQGMO_ACCEPT_OBCIĘTY KOMUNIKAT<br>• MQGMO_CONVERT | • MQGMO_SET_SIGNAL<br>• MQGMO_SYNCPOINT<br>• MQGMO_MARK_SKIP _BACKOUT<br>• MQGMO_MSG_UNDER _CURSOR [4]<br>• BLOKADA MQGMO_LOCK<br>• MQGMO_UNLOCK (odblokowanie MQGMO)<br>• MQGMO_LOGICAL_ORDER (KOLEJNA_KOLEJNA_STRUKTURA)<br>• MQGMO_COMPLETE_MSG<br>• MQGMO_ALL_MSGS_AVAILABLE<br>• MQGMO_ALL_SEGMENTS_ DOSTĘPNE |

**Uwagi:**

1. Jeśli te opcje zostaną zmienione między wywołaniami MQGET, zwracany jest kod przyczyny MQRC_OPTIONS_CHANGED.
2. Jeśli te opcje zostaną podane podczas pierwszego wywołania MQGET, odczyt z wyprzedzeniem zostanie wyłączony. Jeśli te opcje zostaną podane w kolejnym wywołaniu MQGET, zostanie zwrócony kod przyczyny MQRC_OPTIONS_ERROR.
3. Jeśli aplikacja kliencka modyfikuje wartości MsgId i CorrelId między wywołaniami MQGET, komunikaty z poprzednimi wartościami mogły już zostać wysłane do klienta i pozostaną w buforze odczytu z wyprzedzeniem klienta do momentu ich wykorzystania (lub automatycznego wyczyszczenia).
4. Opcja MQGMO_MSG_UNDER_CURSOR nie jest dostępna, jeśli włączony jest odczyt z wyprzedzeniem. Odczyt z wyprzedzeniem jest wyłączony, jeśli podczas otwierania kolejki określono zarówno opcję MQOO_BROWSE, jak i jedną z opcji MQOO_INPUT_SHARED lub MQOO_INPUT_EXCLUSIVE.

5. Jeśli funkcja odczytu z wyprzedzeniem jest włączona, pierwsza operacja MQGET określa, czy komunikaty mają być przeglądane, czy pobierane z kolejki. Jeśli aplikacja kliencka używa następnie wywołania MQGET ze zmienionymi opcjami, takimi jak próba przeglądania po operacji pobierania początkowego lub próba pobrania po operacji przeglądania początkowego, zwracany jest kod przyczyny MQRC_OPTIONS_CHANGED.

Jeśli klient zmienia swoje kryteria wyboru między wywołaniami MQGET, komunikaty przechowywane w buforze odczytu z wyprzedzeniem, które są zgodne z początkowymi kryteriami wyboru, nie są wykorzystywane przez aplikację kliencką i pozostają porzucone w buforze odczytu z wyprzedzeniem klienta. W sytuacjach, gdy bufor odczytu z wyprzedzeniem klienta zawiera wiele porzuconych komunikatów, korzyści związane z odczytem z wyprzedzeniem są tracone, a dla każdego wykorzystywanego komunikatu wymagane jest osobne żądanie skierowane do serwera. Aby określić, czy odczyt z wyprzedzeniem jest używany efektywnie, można użyć parametru statusu połączenia READA.

Odczyt z wyprzedzeniem może zostać zablokowany na żądanie aplikacji z powodu niezgodnych opcji określonych w pierwszym wywołaniu MQGET. W tej sytuacji status połączenia wskazuje, że odczyt z wyprzedzeniem jest zablokowany.

Jeśli z powodu tych ograniczeń w przypadku wywołania MQGET użytkownik zdecyduje, że projekt aplikacji klienckiej nie jest odpowiedni do odczytu z wyprzedzeniem, należy podać opcję MQOPEN MQOO_READ_AHEAD_NO. Alternatywnie można ustawić domyślną wartość odczytu z wyprzedzeniem dla otwieranej kolejki na wartość NO lub DISABLED.

*Włączanie i wyłączanie odczytu z wyprzedzeniem*
Domyślnie odczyt z wyprzedzeniem jest wyłączony. Odczyt z wyprzedzeniem można włączyć na poziomie kolejki lub aplikacji.

## O tym zadaniu

W przypadku wywołania MQOPEN z opcją MQOO_READ_AHEAD klient IBM MQ umożliwia odczyt z wyprzedzeniem, jeśli spełnione są pewne warunki. Są one następujące:

- Aplikacja kliencka musi zostać skompilowana i powiązana z wątkowymi bibliotekami klienta MQI IBM MQ.
- Kanał klienta musi używać protokołu TCP/IP.
- Ustawienie SharingConversations (SHARECNV) kanału musi mieć wartość niezerową w definicji kanału zarówno klienta, jak i serwera.

Aby włączyć odczyt z wyprzedzeniem:

- Aby skonfigurować odczyt z wyprzedzeniem na poziomie kolejki, należy ustawić atrybut kolejki DEFREADA na wartość YES.
- Aby skonfigurować odczyt z wyprzedzeniem na poziomie aplikacji:
  - aby użyć odczytu z wyprzedzeniem, jeśli to możliwe, należy użyć opcji MQOO_READ_AHEAD w wywołaniu funkcji MQOPEN. Aplikacja kliencka nie może korzystać z odczytu z wyprzedzeniem, jeśli atrybut kolejki DEFREADA został ustawiony na wartość DISABLED.
  - Aby użyć odczytu z wyprzedzeniem tylko wtedy, gdy odczyt z wyprzedzeniem jest włączony w kolejce, należy użyć opcji MQOO_READ_AHEAD_AS_Q_DEF w wywołaniu funkcji MQOPEN.

Jeśli projekt aplikacji klienckiej nie nadaje się do odczytu z wyprzedzeniem, można go wyłączyć:

- na poziomie kolejki przez ustawienie atrybutu kolejki DEFREADA na wartość NO, jeśli odczyt z wyprzedzeniem nie ma być używany, chyba że jest żądany przez aplikację kliencką, lub DISABLED, jeśli odczyt z wyprzedzeniem nie ma być używany niezależnie od tego, czy odczyt z wyprzedzeniem jest wymagany przez aplikację kliencką.
- na poziomie aplikacji za pomocą opcji MQOO_NO_READ_AHEAD w wywołaniu funkcji MQOPEN.

Dwie opcje MQCLOSE umożliwiają skonfigurowanie, co stanie się z komunikatami przechowywanymi w buforze odczytu z wyprzedzeniem, jeśli kolejka jest zamknięta.

- Użyj opcji MQCO_IMMEDIATE, aby usunąć komunikaty z buforu odczytu z wyprzedzeniem.

- Użyj komendy MQCO_QUIESCE, aby upewnić się, że komunikaty w buforze odczytu z wyprzedzeniem są używane przez aplikację przed zamknięciem kolejki. Jeśli zostanie wywołana komenda MQCLOSE z opcją MQCO_QUIESCE i w buforze odczytu z wyprzedzeniem pozostaną komunikaty, komenda MQRC_READ_AHEAD_MSGS zwróci komunikat MQCC_WARNING.

*Strojenie wydajności komunikatów nietrwałych w systemie AIX*
Jeśli używany jest system AIX V5.3 lub nowszej, należy rozważyć ustawienie parametru strojenia w celu użycia pełnej wydajności dla komunikatów nietrwałych.

Aby ustawić parametr strojenia w taki sposób, aby był on uwzględniany natychmiast, wydaj następującą komendę jako użytkownik root:

```
/usr/sbin/ioo -o j2_nPagesPerWriteBehindCluster=0
```

Aby ustawić parametr strojenia w taki sposób, aby był on uwzględniany natychmiast i zachowywał się po restarcie, wydaj następującą komendę jako użytkownik root:

```
/usr/sbin/ioo -p -o j2_nPagesPerWriteBehindCluster=0
```

Zwykle komunikaty nietrwałe są przechowywane tylko w pamięci, ale istnieją okoliczności, w których produkt AIX może zaplanować zapisywanie komunikatów nietrwałych na dysku. Komunikaty zaplanowane do zapisania na dysku są niedostępne dla operacji MQGET do czasu zakończenia zapisu na dysku. Sugerowana komenda strojenia zmienia ten próg; zamiast planować zapisywanie komunikatów na dysk, gdy w kolejce znajduje się 16 kilobajtów danych, zapis na dysk występuje tylko wtedy, gdy pamięć rzeczywista na komputerze jest bliska zapełnienia. Jest to zmiana globalna i może mieć wpływ na inne komponenty oprogramowania.

W systemie AIX, gdy używane są aplikacje wielowątkowe, a szczególnie w przypadku uruchamiania na komputerach z wieloma procesorami, zaleca się ustawienie parametru AIXTHREAD_SCOPE=S w identyfikatorze mqm `.profile` lub ustawienie parametru AIXTHREAD_SCOPE=S w środowisku przed uruchomieniem aplikacji, aby zapewnić lepszą wydajność i bardziej stabilne planowanie. Na przykład:

```
export AIXTHREAD_SCOPE=S
```

Ustawienie AIXTHREAD_SCOPE=S oznacza, że wątki użytkownika utworzone z atrybutami domyślnymi są umieszczane w zasięgu rywalizacji w całym systemie. Jeśli wątek użytkownika jest tworzony z zasięgiem rywalizacji w całym systemie, jest on powiązany z wątkiem jądra i jest zaplanowany przez jądro. Bazowy wątek jądra nie jest współużytkowany z żadnym innym wątkiem użytkownika.

## deskryptory plików.

Podczas uruchamiania procesu wielowątkowego, takiego jak proces agenta, można osiągnąć miękki limit dla deskryptorów plików. Ten limit powoduje wyświetlenie IBM MQ kodu przyczyny `MQRC_UNEXPECTED_ERROR (2195)` oraz, jeśli liczba deskryptorów plików jest wystarczająca, pliku™ produktu IBM MQ FFST.

Aby uniknąć tego problemu, można zwiększyć limit liczby deskryptorów plików. W tym celu należy zmienić atrybut `nofiles` w pliku `/etc/security/limits` na 10000 dla identyfikatora użytkownika mqm lub w sekcji default.

## Limity zasobów systemowych

Ustaw limit zasobów systemowych dla segmentu danych i segmentu stosu na nieograniczony, używając następujących komend w wierszu komend:

```
ulimit -d unlimited
ulimit -s unlimited
```

### *Typ indeksu*

Atrybut kolejki *IndexType*określa typ indeksu obsługiwanego przez menedżer kolejek w celu zwiększenia szybkości operacji MQGET w kolejce.

**Uwaga:** Obsługiwane tylko w systemie IBM MQ for z/OS.

Dostępnych jest pięć opcji:

| Wartość | Opis |
|---|---|
| Brak | Indeks nie jest obsługiwany. Tej opcji należy użyć podczas sekwencyjnego pobierania komunikatów (patrz sekcja “Priorytet” na stronie 798 ). |
| groupID | Obsługiwany jest indeks identyfikatorów grup. Tego typu indeksu należy użyć, jeśli wymagane jest logiczne porządkowanie grup komunikatów (patrz sekcja “Porządkowanie logiczne i fizyczne” na stronie 798 ). |
| ID komunikatu | Obsługiwany jest indeks identyfikatorów komunikatów. Tej opcji należy użyć podczas pobierania komunikatów przy użyciu pola *MsgId* jako kryterium wyboru w wywołaniu MQGET (patrz sekcja “Uzyskiwanie konkretnego komunikatu” na stronie 810 ). |
| MSGTOKEN | Obsługiwany jest indeks znaczników komunikatów. |
| CORRELID | Utrzymywany jest indeks identyfikatorów korelacji. Tej opcji należy użyć podczas pobierania komunikatów przy użyciu pola *CorrelId* jako kryterium wyboru w wywołaniu MQGET (patrz sekcja “Uzyskiwanie konkretnego komunikatu” na stronie 810 ). |

**Uwaga:**

1. W przypadku indeksowania przy użyciu opcji MSGID lub CORRELID należy ustawić względne parametry **MsgId** lub **CorrelId** w strukturze MQMD. Nie jest korzystne ustawienie obu tych wartości.
2. Funkcja przeglądania używa mechanizmu indeksu do znalezienia komunikatu, jeśli kolejka spełnia wszystkie następujące warunki:
   - Ma typ indeksu MSGID, CORRELID lub GROUPID
   - Jest przeglądana z tym samym typem identyfikatora
   - Zawiera on komunikaty tylko o jednym priorytecie
3. Należy unikać kolejek (indeksowanych przez *MsgId* lub *CorrelId*) zawierających tysiące komunikatów, ponieważ ma to wpływ na czas restartu. (Nie dotyczy to komunikatów nietrwałych, ponieważ są one usuwane podczas restartu).
4. Parametr MSGTOKEN jest używany do definiowania kolejek zarządzanych przez menedżera obciążenia z/OS .

Pełny opis atrybutu **IndexType** zawiera sekcja IndexType. Więcej informacji na temat atrybutu **IndexType** zawiera sekcja “Uwagi dotyczące projektowania i wydajności aplikacji z/OS” na stronie 67.

### *Obsługa komunikatów o długości większej niż 4 MB*

Komunikaty mogą być zbyt duże dla aplikacji, kolejki lub menedżera kolejek. W zależności od środowiska produkt IBM MQ udostępnia wiele sposobów obsługi komunikatów, które są dłuższe niż 4 MB.

Atrybut **MaxMsgLength** można zwiększyć do 100 MB we wszystkich systemach IBM MQ w wersji V6 lub nowszej. Ustaw tę wartość, aby odzwierciedlała wielkość komunikatów korzystających z kolejki. W systemach IBM MQ innych niż IBM MQ for z/OSmożna również:

1. Użyj komunikatów posegmentowanych. Komunikaty mogą być segmentowane przez aplikację lub menedżera kolejek.
2. Użyj komunikatów referencyjnych.

Każde z tych podejść zostało opisane w dalszej części tej sekcji.

## Zwiększanie maksymalnej długości komunikatu

Atrybut menedżera kolejek systemu **MaxMsgLength** definiuje maksymalną długość komunikatu, który może być obsługiwany przez menedżer kolejek. Podobnie, atrybut kolejki **MaxMsgLength** jest maksymalną długością komunikatu, który może być obsługiwany przez kolejkę. Domyślna maksymalna obsługiwana długość komunikatu zależy od środowiska, w którym pracuje użytkownik.

W przypadku obsługi dużych komunikatów można zmienić te atrybuty niezależnie na platformach innych niż z/OS. Można ustawić wartość atrybutu menedżera kolejek z zakresu od 32768 bajtów do 100 MB.

**Ostrzeżenie:** W systemie IBM MQ for z/OS atrybut **MaxMsgLength** menedżera kolejek jest zakodowany na stałe z wartością 100 MB.

Na wszystkich platformach można ustawić wartość atrybutu kolejki z zakresu od 0 do 100 MB.

Po zmianie jednego lub obu atrybutów **MaxMsgLength** należy zrestartować aplikacje i kanały, aby zmiany odniosły skutek.

Po wprowadzeniu tych zmian długość komunikatu musi być mniejsza lub równa zarówno atrybutom kolejki, jak i atrybutu **MaxMsgLength** menedżera kolejek. Jednak istniejące komunikaty mogą być dłuższe niż jeden z tych atrybutów.

Jeśli komunikat jest zbyt duży dla kolejki, zwracany jest komunikat MQRC_MSG_TOO_BIG_FOR_Q. Podobnie, jeśli komunikat jest zbyt duży dla menedżera kolejek, zwracana jest wartość MQRC_MSG_TOO_BIG_FOR_Q_MGR.

Ta metoda obsługi dużych wiadomości jest łatwa i wygodna. Jednak przed użyciem należy wziąć pod uwagę następujące czynniki:

- Zmniejsza się jednolitość między menedżerami kolejek. Maksymalna wielkość danych komunikatu jest określana przez parametr *MaxMsgLength* dla każdej kolejki (w tym kolejek transmisji), w której zostanie umieszczony komunikat. Ta wartość często przyjmuje wartość domyślną *MaxMsgLength*menedżera kolejek, szczególnie dla kolejek transmisji. Utrudnia to przewidywanie, czy komunikat jest zbyt duży, jeśli ma być przesyłany do zdalnego menedżera kolejek.
- Użycie zasobów systemowych zostało zwiększone. Na przykład aplikacje wymagają większych buforów, a na niektórych platformach może wystąpić zwiększone użycie współużytkowanej pamięci masowej. Pamięć kolejki powinna być używana tylko wtedy, gdy jest wymagana dla większych komunikatów.
- Ma to wpływ na przetwarzanie wsadowe kanału. Duży komunikat jest nadal liczony jako jeden komunikat w stosunku do liczby zadań wsadowych, ale wymaga więcej czasu na przesłanie, co zwiększa czasy odpowiedzi dla innych komunikatów.

Multi *Segmentacja komunikatów*

Ten temat zawiera informacje na temat segmentacji komunikatów. Ta funkcja nie jest obsługiwana w systemie IBM MQ for z/OS ani przez aplikacje korzystające z produktu IBM MQ classes for JMS.

Zwiększenie maksymalnej długości komunikatu zgodnie z opisem w temacie "Zwiększanie maksymalnej długości komunikatu" na stronie 817 ma pewne negatywne konsekwencje. Może to również spowodować, że komunikat będzie zbyt duży dla kolejki lub menedżera kolejek. W takich przypadkach można segmentować komunikat. Więcej informacji na temat segmentów zawiera sekcja "Grupy komunikatów" na stronie 45.

Następne sekcje zawierają informacje na temat powszechnych zastosowań segmentacji komunikatów. W przypadku umieszczania i niszczycielskiego pobierania zakłada się, że wywołania MQPUT lub MQGET *zawsze* działają w obrębie jednostki pracy. Należy zawsze rozważyć użycie tej techniki w celu zmniejszenia możliwości obecności niekompletnych grup w sieci. Zakłada się, że menedżer kolejek zatwierdził jednofazowo, ale inne techniki koordynacji są równie poprawne.

Ponadto w przypadku pobierania aplikacji zakłada się, że jeśli wiele serwerów przetwarza tę samą kolejkę, każdy serwer wykonuje podobny kod, dzięki czemu jeden serwer nigdy nie znajdzie oczekiwanego komunikatu lub segmentu (ponieważ wcześniej określał MQGMO_ALL_MSGS_AVAILABLE lub MQGMO_ALL_SEGMENTS_AVAILABLE).

**Ostrzeżenie:** W przypadku używania publikowania/subskrypcji do wysyłania komunikatów do tematu (lub umieszczania komunikatów w aliasach tematów) grupowanie i segmentacja komunikatów nie jest dozwolona.

Ponieważ subskrypcje mogą być tworzone i usuwane niezależnie od działania publikowania, nie można zapewnić, że subskrybent otrzyma pełną grupę komunikatów lub wszystkie segmenty komunikatu. Patrz RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP.

## Umieszczanie i pobieranie segmentowanego komunikatu, który obejmuje jednostki pracy

Istnieje możliwość umieszczenia i pobrania komunikatu segmentowanego, który obejmuje jednostkę pracy w sposób podobny do następującego: "Tworzenie i tworzenie grupy, która obejmuje jednostki pracy" na stronie 807.

Nie można jednak umieścić lub uzyskać segmentowanych komunikatów w globalnej jednostce pracy.

Multi *Segmentacja i reasemblacja według menedżera kolejek*

Jest to najprostszy scenariusz, w którym jedna aplikacja umieszcza komunikat do pobrania przez inną aplikację. Komunikat może być duży: nie jest zbyt duży dla aplikacji umieszczającej lub pobierającej do obsługi w pojedynczym buforze, ale zbyt duży dla menedżera kolejek lub kolejki, w której komunikat ma zostać umieszczony.

Jedynymi zmianami wymaganymi przez te aplikacje są uprawnienia aplikacji do wykonywania segmentacji przez menedżera kolejek, jeśli jest to konieczne:

```
   PMO.Options = (existing options)
   MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED
   MD.Version = MQMD_VERSION_2
   memcpy(MD.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH)
MQPUT
```

i aby aplikacja zwracała się do menedżera kolejek z prośbą o ponowne złożenie komunikatu, jeśli został on podzielony na segmenty:

```
   GMO.Options = MQGMO_COMPLETE_MSG | (existing options)
   MQGET
```

W tym najprostszym scenariuszu aplikacja musi zresetować pole GroupId na wartość MQGI_NONE przed wywołaniem MQPUT, aby menedżer kolejek mógł wygenerować unikalny identyfikator grupy dla każdego komunikatu. Jeśli ta czynność nie zostanie wykonana, niepowiązane komunikaty mogą mieć ten sam identyfikator grupy, co może prowadzić do niepoprawnego przetwarzania.

Bufor aplikacji musi być wystarczająco duży, aby mógł zawierać ponownie złożony komunikat (chyba że zostanie użyta opcja MQGMO_ACCEPT_TRUNCATED_MSG).

Jeśli atrybut MAXMSGLEN kolejki ma zostać zmodyfikowany w celu uwzględnienia segmentacji komunikatów, należy rozważyć następujące kwestie:

- Minimalny segment komunikatu obsługiwany w kolejce lokalnej to 16 bajtów.
- W przypadku kolejki transmisji parametr MAXMSGLEN musi również zawierać miejsce wymagane dla nagłówków. Należy rozważyć użycie wartości o długości co najmniej 4000 bajtów większej niż maksymalna oczekiwana długość danych użytkownika w dowolnym segmencie komunikatu, który może zostać umieszczony w kolejce transmisji.

Jeśli konwersja danych jest konieczna, aplikacja pobierająca może ją wykonać, określając opcję MQGMO_CONVERT. Powinno to być proste, ponieważ wyjście konwersji danych jest wyświetlane z kompletnym komunikatem. Nie należy podejmować próby konwersji danych w kanale nadawczym, jeśli komunikat jest segmentowany, a format danych jest taki, że wyjście konwersji danych nie może przeprowadzić konwersji na niekompletnych danych.

Multi *Segmentacja aplikacji*

Segmentacja aplikacji jest używana, gdy segmentacja menedżera kolejek nie jest odpowiednia lub gdy aplikacje wymagają konwersji danych o określonych granicach segmentów.

Segmentacja aplikacji jest używana z dwóch głównych powodów:

1. Sama segmentacja menedżera kolejek nie jest odpowiednia, ponieważ komunikat jest zbyt duży, aby mógł być obsługiwany w pojedynczym buforze przez aplikacje.
2. Konwersja danych musi być wykonywana przez kanały nadawcze, a format jest taki, że aplikacja składająca musi określać, gdzie mają być granice segmentu, aby możliwa była konwersja pojedynczego segmentu.

Jeśli jednak konwersja danych nie jest problemem lub jeśli aplikacja pobierająca zawsze używa opcji MQGMO_COMPLETE_MSG, segmentacja menedżera kolejek może być również dozwolona przez określenie opcji MQMF_SEGMENTATION_ALLOWED. W tym przykładzie aplikacja dzieli komunikat na cztery segmenty:

```
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
```

Jeśli parametr MQPMO_LOGICAL_ORDER nie jest używany, aplikacja musi ustawić parametr *Offset* i długość każdego segmentu. W takim przypadku stan logiczny nie jest zachowywany automatycznie.

Aplikacja pobierująca nie może zagwarantować, że bufor będzie wystarczająco duży, aby pomieścić ponownie złożony komunikat. Dlatego musi być przygotowana do indywidualnego przetwarzania segmentów.

W przypadku komunikatów segmentowanych ta aplikacja nie chce rozpoczynać przetwarzania jednego segmentu, dopóki nie zostaną wyświetlone wszystkie segmenty, które tworzą komunikat logiczny. Dlatego MQGMO_ALL_SEGMENTS_AVAILABLE jest określony dla pierwszego segmentu. Jeśli określono parametr MQGMO_LOGICAL_ORDER i istnieje bieżący komunikat logiczny, parametr MQGMO_ALL_SEGMENTS_AVAILABLE jest ignorowany.

Po pobraniu pierwszego segmentu komunikatu logicznego należy użyć parametru MQGMO_LOGICAL_ORDER, aby upewnić się, że pozostałe segmenty komunikatu logicznego są pobierane w odpowiedniej kolejności.

Komunikaty w różnych grupach nie są brane pod uwagę. W przypadku wystąpienia takich komunikatów są one przetwarzane w kolejności, w jakiej występuje pierwszy segment każdego komunikatu w kolejce.

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_SEGMENTS_AVAILABLE | MQGMO_WAIT
do while ( SegmentStatus == MQSS_SEGMENT )
   MQGET
   /* Process each remaining segment of the logical message */
   ...

MQCMIT
```

Multi *Segmentacja aplikacji dla komunikatów logicznych*

Komunikaty muszą być obsługiwane w porządku logicznym w grupie, a niektóre lub wszystkie z nich mogą być tak duże, że wymagają segmentacji aplikacji.

W naszym przykładzie należy umieścić grupę czterech logicznych komunikatów. Wszystkie komunikaty oprócz trzeciego są duże i wymagają segmentacji, która jest wykonywana przez aplikację umieszczającą:

```
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
```

W aplikacji pobierającej MQGMO_ALL_MSGS_AVAILABLE jest określony w pierwszej operacji MQGET. Oznacza to, że nie są pobierane żadne komunikaty ani segmenty grupy, dopóki nie będzie dostępna cała grupa. Po pobraniu pierwszego komunikatu fizycznego grupy parametr MQGMO_LOGICAL_ORDER jest używany w celu zapewnienia, że segmenty i komunikaty grupy są pobierane w kolejności:

```
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
```

**Uwaga:** Jeśli określono parametr MQGMO_LOGICAL_ORDER i istnieje bieżąca grupa, parametr MQGMO_ALL_MSGS_AVAILABLE jest ignorowany.

*Komunikaty referencyjne*
Ten temat zawiera więcej informacji na temat komunikatów referencyjnych.

**Uwaga:** Nieobsługiwany w systemie IBM MQ for z/OS.

Ta metoda umożliwia przesłanie dużego obiektu z jednego węzła do innego bez zapisywania obiektu w kolejkach IBM MQ w węźle źródłowym lub docelowym. Jest to szczególnie korzystne, gdy dane istnieją w innej formie, na przykład dla aplikacji poczty.

W tym celu należy określić wyjście komunikatu na obu końcach kanału. Więcej informacji na ten temat zawiera sekcja “Programy obsługi wyjścia komunikatów kanału” na stronie 1008.

IBM MQ definiuje format nagłówka komunikatu referencyjnego (MQRMH). Opis tej sytuacji można znaleźć w sekcji MQRMH . Jest ona rozpoznawana przy użyciu zdefiniowanej nazwy formatu i może następować po niej rzeczywiste dane.

Aby zainicjować przesyłanie dużego obiektu, aplikacja może umieścić komunikat składający się z nagłówka komunikatu odwołania bez żadnych danych po nim. Ponieważ ten komunikat opuszcza węzeł, wyjście komunikatu pobiera obiekt w odpowiedni sposób i dodaje go do komunikatu odwołania. Następnie zwraca komunikat (teraz większy niż wcześniej) do wysyłającego agenta kanału komunikatów w celu transmisji do odbierającego agenta MCA.

W odbierającym MCA skonfigurowano inne wyjście komunikatu. Gdy to wyjście komunikatu odbierze jeden z tych komunikatów, tworzy obiekt przy użyciu danych obiektu, które zostały dodane, i przekazuje komunikat odniesienia *bez* tego komunikatu. Komunikat odniesienia może zostać odebrany przez aplikację i aplikacja ta wie, że obiekt (lub przynajmniej jego część reprezentowana przez ten komunikat odniesienia) został utworzony w tym węźle.

Maksymalna ilość danych obiektu, którą wyjście komunikatu wysyłającego może dołączyć do komunikatu odniesienia, jest ograniczona przez wynegocjowaną maksymalną długość komunikatu dla kanału. Wyjście może zwrócić tylko jeden komunikat do agenta MCA dla każdego przekazywanego komunikatu, dzięki czemu aplikacja umieszczająca może umieścić kilka komunikatów w celu przesłania jednego obiektu.

Każdy komunikat musi identyfikować *logiczną* długość i przesunięcie obiektu, który ma zostać do niego dodany. Jednak w przypadkach, gdy nie jest możliwe poznanie całkowitej wielkości obiektu lub maksymalnej wielkości dozwolonej przez kanał, należy zaprojektować wyjście komunikatu wysyłającego, tak aby aplikacja umieszczająca po prostu umieszczała pojedynczy komunikat, a wyjście umieszczało następny komunikat w kolejce transmisji, gdy do przekazanego komunikatu dołączono tyle danych, ile jest możliwe.

Przed użyciem tej metody obsługi dużych komunikatów należy rozważyć następujące kwestie:

- Agent MCA i wyjście komunikatu są uruchamiane przy użyciu identyfikatora użytkownika IBM MQ . Wyjście komunikatu (i w związku z tym ID użytkownika) musi uzyskać dostęp do obiektu, aby pobrać go na końcu wysyłającym lub utworzyć go na końcu odbierającym; może to być wykonalne tylko w przypadkach, gdy obiekt jest powszechnie dostępny. Powoduje to problem z bezpieczeństwem.
- Jeśli komunikat odwołania z dołączonymi do niego danymi masowymi musi przejść przez kilka menedżerów kolejek przed osiągnięciem miejsca docelowego, dane masowe są obecne w kolejkach produktu IBM MQ w węzłach pośrednich. Jednak w takich przypadkach nie ma potrzeby zapewnienia specjalnego wsparcia lub wyjścia.
- Projektowanie wyjścia komunikatu jest utrudnione, jeśli dozwolone jest przekierowanie lub kolejkowanie niedostarczonych komunikatów. W takich przypadkach części obiektu mogą pojawić się w nieodpowiedniej kolejności.
- Gdy komunikat odniesienia dociera do miejsca docelowego, wyjście komunikatu odbierającego tworzy obiekt. Jednak nie jest to zsynchronizowane z jednostką pracy agenta MCA, więc jeśli zadanie wsadowe zostanie wycofane, inny komunikat odniesienia zawierający tę samą część obiektu zostanie odebrany w późniejszym zadaniu wsadowym, a wyjście komunikatu może podjąć próbę ponownego utworzenia tej samej części obiektu. Jeśli obiekt jest na przykład serią aktualizacji bazy danych, może to być nie do przyjęcia. Jeśli tak, wyjście komunikatu musi przechowywać dziennik, którego aktualizacje zostały zastosowane. Może to wymagać użycia kolejki IBM MQ .
- W zależności od charakterystyki typu obiektu, wyjścia komunikatów i aplikacje mogą wymagać współpracy przy utrzymywaniu liczników użycia, aby obiekt mógł zostać usunięty, gdy nie jest już potrzebny. Identyfikator instancji może być również wymagany. W tym celu należy podać pole w nagłówku komunikatu odniesienia (patrz MQRMH ).
- Jeśli komunikat odniesienia jest umieszczany jako lista dystrybucyjna, obiekt musi być dostępny do pobrania dla każdej wynikowej listy dystrybucyjnej lub pojedynczego miejsca docelowego w tym węźle. Może być konieczne zachowanie liczby użyć. Należy również rozważyć możliwość, że węzeł może być węzłem końcowym dla niektórych miejsc docelowych na liście, ale węzłem pośrednim dla innych.
- Dane masowe zwykle nie są konwertowane. Jest to spowodowane tym, że konwersja odbywa się *przed* wywołaniem wyjścia komunikatu. Z tego powodu konwersja nie może być żądana w źródłowym kanale nadawczym. Jeśli komunikat odwołania przechodzi przez węzeł pośredni, dane masowe są przekształcane po wysłaniu z węzła pośredniego (jeśli jest to wymagane).
- Komunikaty odniesienia nie mogą być segmentowane.

## Korzystanie ze struktur MQRMH i MQMD

Sekcja MQRMH i MQMD zawiera opis pól w nagłówku komunikatu odniesienia i deskryptorze komunikatu.

W strukturze MQMD ustaw pole *Format* na wartość MQFMT_REF_MSG_HEADER. Format MQHREF na żądanie w MQGET jest automatycznie przekształcany przez program IBM MQ wraz z danymi masowymi, które następują po nim.

Poniżej przedstawiono przykład użycia pól *DataLogicalOffset* i *DataLogicalLength* w programie MQRMH:

Aplikacja umieszczającą może umieścić komunikat odwołania z następującymi danymi:

- Brak danych fizycznych
- *DataLogicalLength* = 0 (ten komunikat reprezentuje cały obiekt)
- *DataLogicalOffset* = 0.

Przyjmując, że obiekt ma 70 000 bajtów długości, wyjście komunikatu wysyłającego wysyła pierwsze 40 000 bajtów wzdłuż kanału w komunikacie odniesienia zawierającym:

- 40 000 bajtów danych fizycznych po MQRMH
- *DataLogicalLength* = 40000
- *DataLogicalOffset* = 0 (od początku obiektu).

Następnie umieszcza inny komunikat w kolejce transmisji zawierający:

- Brak danych fizycznych
- *DataLogicalLength* = 0 (do końca obiektu). W tym miejscu można określić wartość 30 000.
- *DataLogicalOffset* = 40000 (począwszy od tego punktu).

Gdy wyjście komunikatu jest widoczne dla wyjścia komunikatu wysyłającego, pozostałe 30 000 bajtów danych jest dołączanych, a pola są ustawione na:

- 30 000 bajtów danych fizycznych po MQRMH
- *DataLogicalLength* = 30000
- *DataLogicalOffset* = 40000 (począwszy od tego punktu).

Flaga MQRMHF_LAST jest również ustawiona.

Opis przykładowych programów, w których można używać komunikatów informacyjnych, zawiera sekcja “Korzystanie z przykładowych programów w systemie Multiplatforms” na stronie 1090.

### *Oczekiwanie na komunikaty*

Aby program oczekiwał na przybycie komunikatu do kolejki, należy określić opcję MQGMO_WAIT w polu *Options* struktury MQGMO.

Użyj pola *WaitInterval* w strukturze MQGMO, aby określić Maksymalny czas (w milisekundach), przez który wywołanie MQGET ma oczekiwać na przybycie komunikatu do kolejki.

Jeśli komunikat nie zostanie odebrany w tym czasie, wywołanie MQGET zostanie zakończone z kodem przyczyny MQRC_NO_MSG_AVAILABLE.

Nieograniczony przedział czasu oczekiwania można określić za pomocą stałej MQWI_UNLIMITED w polu *WaitInterval* . Jednak zdarzenia poza kontrolą użytkownika mogą spowodować, że program będzie czekał przez długi czas, dlatego należy używać tej stałej z zachowaniem ostrożności. Aplikacje IMS nie mogą określać nieograniczonego odstępu czasu oczekiwania, ponieważ mogłoby to uniemożliwić zakończenie działania systemu IMS . (Po zakończeniu działania programu IMS wymagane jest zakończenie wszystkich regionów zależnych). Zamiast tego aplikacje IMS mogą określić skończony przedział czasu oczekiwania. Jeśli wywołanie zostanie zakończone bez pobierania komunikatu po upływie tego okresu, należy wywołać inną operację MQGET z opcją oczekiwania.

**Uwaga:** Jeśli więcej niż jeden program oczekuje w tej samej kolejce współużytkowanej na *usunięcie* komunikatu, tylko jeden program jest aktywowany przez nadchodzący komunikat. Jeśli jednak więcej niż jeden program oczekuje na przejrzanie komunikatu, wszystkie programy mogą zostać aktywowane. Więcej informacji na ten temat zawiera opis pola *Options* struktury MQGMO w sekcji MQGMO.

Jeśli stan kolejki lub menedżera kolejek zmieni się przed upływem okresu oczekiwania, wykonywane są następujące działania:

- Jeśli menedżer kolejek przejdzie w stan wyciszania i użyto opcji MQGMO_FAIL_IF_QUIESCING, oczekiwanie zostanie anulowane, a wywołanie MQGET zostanie zakończone z kodem przyczyny MQRC_Q_MGR_QUIESCING. Bez tej opcji wywołanie pozostaje w stanie oczekiwania.
- z/OS Jeśli w systemie z/OSpołączenie (dla aplikacji CICS lub IMS ) przejdzie w stan wyciszania i zostanie użyta opcja MQGMO_FAIL_IF_QUIESCING, oczekiwanie zostanie anulowane, a wywołanie MQGET zakończy się z kodem przyczyny MQRC_CONN_QUIESCING. Bez tej opcji wywołanie pozostaje w stanie oczekiwania.

- Jeśli zostanie wymuszone zatrzymanie menedżera kolejek lub zostanie on anulowany, wywołanie MQGET zostanie zakończone z kodem przyczyny MQRC_Q_MGR_STOP lub MQRC_CONNECTION_BROKEN.
- Jeśli atrybuty kolejki (lub kolejki, na którą tłumaczona jest nazwa kolejki) zostaną zmienione w taki sposób, że żądania pobierania są teraz zablokowane, oczekiwanie zostanie anulowane, a wywołanie MQGET zostanie zakończone z kodem przyczyny MQRC_GET_INHIBITED.
- Jeśli atrybuty kolejki (lub kolejki, na którą tłumaczona jest nazwa kolejki) zostały zmienione w taki sposób, że opcja FORCE jest wymagana, oczekiwanie jest anulowane, a wywołanie MQGET kończy się z kodem przyczyny MQRC_OBJECT_CHANGED.

z/OS Jeśli aplikacja ma oczekiwać na więcej niż jedną kolejkę, należy użyć narzędzia sygnału IBM MQ for z/OS (patrz sekcja "Sygnalizacja" na stronie 823 ). Więcej informacji na temat okoliczności, w których występują te działania, zawiera sekcja MQGMO.

### *Sygnalizacja*

Sygnalizacja jest obsługiwana tylko w systemie IBM MQ for z/OS.

Sygnalizowanie jest opcją wywołania MQGET umożliwiającą systemowi operacyjnemu powiadomienie (lub *sygnał* ) Program, gdy w kolejce pojawi się oczekiwany komunikat. Jest to podobne do funkcji *get with wait* opisanej w temacie "Oczekiwanie na komunikaty" na stronie 822 , ponieważ umożliwia ona programowi kontynuowanie pracy podczas oczekiwania na sygnał. Jeśli jednak używane jest sygnalizowanie, można zwolnić wątek aplikacji i polegać na systemie operacyjnym, który powiadamia program o nadejściu komunikatu.

## Aby ustawić sygnał

Aby ustawić sygnał, wykonaj następujące czynności w strukturze MQGMO używanej w wywołaniu MQGET:

1. Ustaw opcję MQGMO_SET_SIGNAL w polu *Options* .
2. Ustaw maksymalny czas życia sygnału w polu *WaitInterval* . Określa czas (w milisekundach), przez który program IBM MQ ma monitorować kolejkę. Aby określić nieograniczony czas życia, należy użyć wartości MQWI_UNLIMITED.

   **Uwaga:** Aplikacje IMS nie mogą określać nieograniczonego odstępu czasu oczekiwania, ponieważ mogłoby to uniemożliwić zakończenie działania systemu IMS . (Po zakończeniu działania programu IMS wymagane jest zakończenie wszystkich regionów zależnych). Zamiast tego aplikacje IMS mogą sprawdzać stan EBC w regularnych odstępach czasu (patrz krok 3). Program może mieć jednocześnie ustawione sygnały w kilku uchwytach kolejki:
3. W polu *Signal1* podaj adres *Event Control Block* (EBC). Spowoduje to powiadomienie o wyniku sygnału. Pamięć masowa EBC musi pozostać dostępna do czasu zamknięcia kolejki.

**Uwaga:** Opcji MQGMO_SET_SIGNAL nie można używać z opcją MQGMO_WAIT.

## Po nadejściu komunikatu

Po nadejściu odpowiedniego komunikatu do EBC zwracany jest kod zakończenia.

Kod zakończenia opisuje jedną z następujących czynności:

- Komunikat, dla którego ustawiono sygnał, dotarł do kolejki. Komunikat nie jest zarezerwowany dla programu, który zażądał sygnału, dlatego program musi ponownie wywołać komendę MQGET, aby uzyskać komunikat.

  **Uwaga:** Inna aplikacja może pobrać komunikat w czasie między odebraniem sygnału a wysłaniem kolejnego wywołania MQGET.
- Ustawiony odstęp czasu oczekiwania wygasł, a komunikat, dla którego ustawiono sygnał, nie dotarł do kolejki. Program IBM MQ anulował sygnał.
- Sygnał został anulowany. Dzieje się tak na przykład w przypadku zatrzymania menedżera kolejek lub zmiany atrybutu kolejki, co powoduje, że wywołania MQGET nie są już dozwolone.

Jeśli odpowiedni komunikat znajduje się już w kolejce, wywołanie MQGET kończy się w taki sam sposób, jak wywołanie MQGET bez sygnalizowania. Jeśli błąd zostanie wykryty natychmiast, wywołanie zostanie zakończone i zostaną ustawione kody powrotu.

Gdy wywołanie jest akceptowane i nie jest natychmiast dostępny żaden komunikat, sterowanie jest zwracane do programu, aby mógł on kontynuować pracę. Nie ustawiono żadnego pola wyjściowego w deskryptorze komunikatu, ale parametr **CompCode** jest ustawiony na wartość MQCC_WARNING, a parametr **Reason** na wartość MQRC_SIGNAL_REQUEST_ACCEPTED.

Informacje na temat tego, co produkt IBM MQ może zwrócić do aplikacji podczas wykonywania wywołania MQGET przy użyciu sygnalizacji, zawiera sekcja MQGET.

Jeśli program nie ma innego zadania do wykonania w czasie, gdy oczekuje na opublikowanie przez EBC, może czekać na EBC przy użyciu:

- W przypadku programu CICS Transaction Server for z/OS komenda EXEC CICS WAIT EXTERNAL
- W przypadku programów wsadowych i IMS makro z/OS WAIT

Jeśli stan kolejki lub menedżera kolejek zmienia się podczas ustawiania sygnału (to znaczy, że EBC nie został jeszcze wysłany), wykonywane są następujące działania:

- Jeśli menedżer kolejek przejdzie w stan wyciszania i zostanie użyta opcja MQGMO_FAIL_IF_QUIESCING, sygnał zostanie anulowany. EBC jest księgowany z kodem zakończenia MQEC_Q_MGR_QUIESCING. Bez tej opcji sygnał pozostaje ustawiony.
- Jeśli zostanie wymuszone zatrzymanie menedżera kolejek lub zostanie on anulowany, sygnał zostanie anulowany. Sygnał jest dostarczany z kodem zakończenia MQEC_WAIT_ANULOWANA.
- Jeśli atrybuty kolejki (lub kolejki, na którą tłumaczona jest nazwa kolejki) zostaną zmienione w taki sposób, że żądania pobierania są teraz zablokowane, sygnał jest anulowany. Sygnał jest dostarczany z kodem zakończenia MQEC_WAIT_ANULOWANA.

**Uwaga:**

1. Jeśli więcej niż jeden program ustawił sygnał w tej samej kolejce współużytkowanej w celu usunięcia komunikatu, tylko jeden program jest aktywowany przez nadchodzący komunikat. Jeśli jednak więcej niż jeden program oczekuje na przejrzanie komunikatu, wszystkie programy mogą zostać aktywowane. Reguły stosowane przez menedżer kolejek przy podejmowaniu decyzji o tym, które aplikacje mają zostać aktywowane, są takie same jak w przypadku oczekujących aplikacji: więcej informacji na ten temat zawiera opis pola *Options* struktury MQGMO w sekcji MQGMO-Get-message options(Opcje pobierania komunikatów).
2. Jeśli istnieje więcej niż jedno wywołanie MQGET oczekujące na ten sam komunikat, z kombinacją opcji oczekiwania i sygnału, każde wywołanie oczekujące jest traktowane jednakowo. Więcej informacji na ten temat zawiera opis pola *Options* struktury MQGMO w sekcji Opcje MQGMO-Get-message.
3. W pewnych warunkach możliwe jest zarówno pobranie komunikatu przez wywołanie MQGET, jak i dostarczenie sygnału (będącego wynikiem nadejścia tego samego komunikatu). Oznacza to, że gdy program wysyła kolejne wywołanie MQGET (ponieważ sygnał został dostarczony), komunikat może nie być dostępny. Zaprojektuj program do testowania tej sytuacji.

Więcej informacji na temat ustawiania sygnału zawiera opis opcji MQGMO_SET_SIGNAL i pola *Signal1* w publikacji Signal1.

### *Pomijanie wycofywania*

Podając opcję **MQGMO_MARK_SKIP_BACKOUT** w wywołaniu MQGET, można uniemożliwić aplikacji wprowadzenie pętli *MQGET-error-backout* .

**Uwaga:** Obsługiwane tylko w systemie IBM MQ for z/OS.

W ramach jednostki pracy aplikacja może wywołać co najmniej jedno wywołanie MQGET w celu pobrania komunikatów z kolejki. Jeśli aplikacja wykryje błąd, może wycofać jednostkę pracy. Spowoduje to odtworzenie wszystkich zasobów zaktualizowanych podczas tej jednostki pracy do stanu, w którym znajdowały się one przed uruchomieniem jednostki pracy, i przywrócenie komunikatów pobranych przez wywołania MQGET.

Po przywróceniu te komunikaty są dostępne dla kolejnych wywołań MQGET wysyłanych przez aplikację. W wielu przypadkach nie powoduje to problemu z programem użytkowym. Jednak w przypadkach, gdy nie można obejść błędu prowadzącego do wycofania, przywrócenie komunikatu do kolejki może spowodować, że aplikacja wejdzie w pętlę *MQGET-error-backout* .

Aby uniknąć tego problemu, należy podać opcję MQGMO_MARK_SKIP_BACKOUT w wywołaniu MQGET. Oznacza to, że żądanie MQGET nie jest związane z wycofaniem inicjowanym przez aplikację, co oznacza, że nie może zostać wycofane. Użycie tej opcji oznacza, że w przypadku wycofania aktualizacji innych zasobów są one wycofywane zgodnie z wymaganiami, ale oznaczony komunikat jest traktowany tak, jakby został pobrany w ramach nowej jednostki pracy.

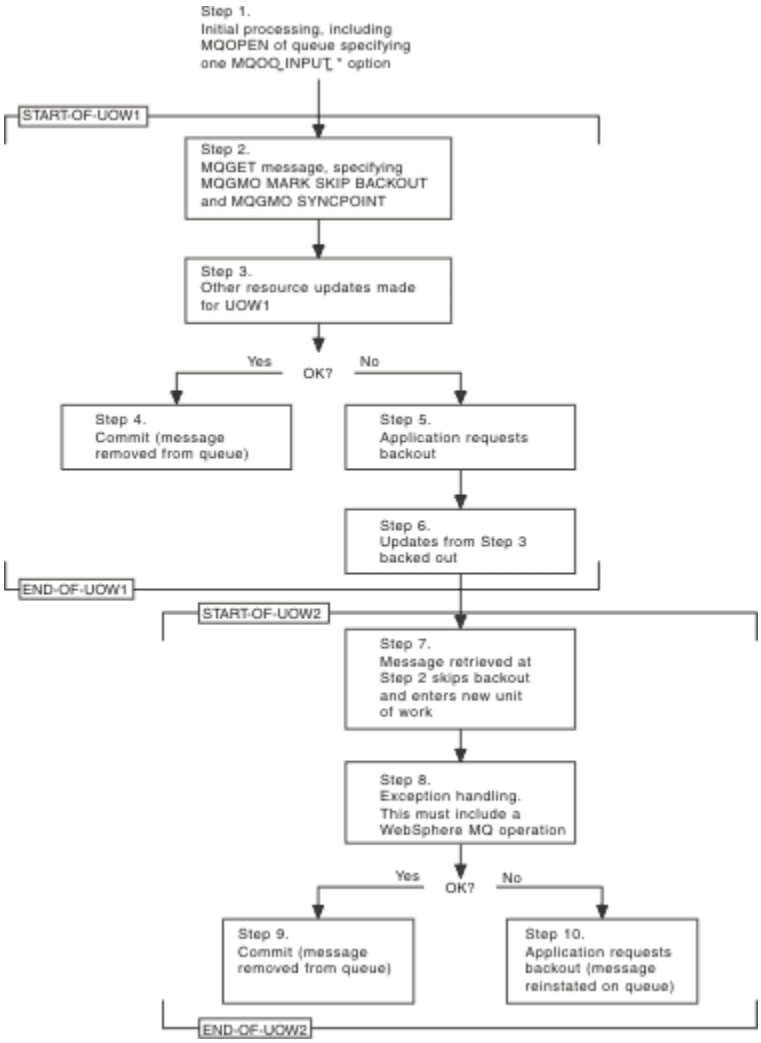
Aplikacja musi wywołać funkcję IBM MQ , aby zatwierdzić nową jednostkę pracy lub wycofać nową jednostkę pracy. Na przykład program może wykonać obsługę wyjątków, na przykład poinformować nadawcę, że komunikat został odrzucony, i zatwierdzić jednostkę pracy, tak aby usunąć komunikat z kolejki. Jeśli nowa jednostka pracy zostanie wycofana (z dowolnej przyczyny), komunikat zostanie przywrócony do kolejki.

W obrębie jednostki pracy może istnieć tylko jedno żądanie MQGET oznaczone jako pomijane wycofywanie; może jednak istnieć kilka innych komunikatów, które nie są oznaczane jako pomijane wycofywanie. Po oznaczeniu komunikatu jako pominiętego wycofania wszystkie kolejne wywołania MQGET w jednostce pracy, które określają opcję MQGMO_MARK_SKIP_BACKOUT, kończą się niepowodzeniem z kodem przyczyny MQRC_SECOND_MARK_NOT_ALLOWED.

**Uwaga:**

1. Oznaczony komunikat jest wycofywany tylko wtedy, gdy jednostka pracy, która go zawiera, została zakończona przez żądanie aplikacji dotyczące wycofania. Jeśli jednostka pracy zostanie wycofana z innego powodu, komunikat zostanie wycofany do kolejki w taki sam sposób, jak gdyby nie został oznaczony do pominięcia wycofania.
2. Funkcja pominięcia wycofania nie jest obsługiwana w procedurach składowanych Db2 uczestniczących w jednostkach pracy kontrolowanych przez usługi RRS. Na przykład wywołanie MQGET z opcją MQGMO_MARK_SKIP_BACKOUT zakończy się niepowodzeniem z kodem przyczyny MQRC_OPTION_ENVIRONMENT_ERROR.

Rysunek 63 na stronie 826 przedstawia typową sekwencję kroków, które aplikacja może zawierać, gdy do pominięcia wycofania wymagane jest żądanie MQGET.




*Rysunek 63. Pomijanie wycofywania przy użyciu opcji MQGMO_MARK_SKIP_BACKOUT*

Rysunek 63 na stronie 826 zawiera następujące kroki:

**Krok 1**
Przetwarzanie początkowe odbywa się w ramach transakcji, w tym wywołanie MQOPEN w celu otwarcia kolejki (określenie jednej z opcji MQOO_INPUT_ * w celu pobrania komunikatów z kolejki w kroku 2).

**Krok 2**
Wywoływana jest komenda MQGET z MQGMO_SYNCPOINT i MQGMO_MARK_SKIP_BACKOUT. Parametr MQGMO_SYNCPOINT jest wymagany, ponieważ komenda MQGET musi znajdować się w jednostce pracy, aby opcja MQGMO_MARK_SKIP_BACKOUT była aktywna. W Rysunek 63 na stronie 826 ta jednostka pracy jest nazywana UOW1.

**Krok 3.**
Inne aktualizacje zasobów są wykonywane w ramach UOW1. Mogą to być kolejne wywołania MQGET (wydane bez opcji MQGMO_MARK_SKIP_BACKOUT).

**Krok 4**
Wszystkie aktualizacje z kroków 2 i 3 są wykonywane zgodnie z wymaganiami. Aplikacja zatwierdza aktualizacje i kończy działanie UOW1 . Komunikat pobrany w kroku 2 zostanie usunięty z kolejki.

**Krok 5**
Niektóre aktualizacje z kroków 2 i 3 nie są wykonywane zgodnie z wymaganiami. Aplikacja żąda, aby aktualizacje wykonane podczas tych kroków zostały wycofane.

**Krok 6**
Aktualizacje wprowadzone w kroku 3 zostały wycofane.

**Krok 7**
Żądanie MQGET wykonane w kroku 2 pomija wycofanie i staje się częścią nowej jednostki pracy UOW2.

**Krok 8**
UOW2 wykonuje obsługę wyjątków w odpowiedzi na wycofaną operację UOW1 . (Na przykład wywołanie MQPUT do innej kolejki wskazujące, że wystąpił problem, który spowodował wycofanie UOW1 ).

**Krok 9**
Krok 8 kończy się zgodnie z wymaganiami, program użytkowy zatwierdza działanie i kończy działanie UOW2 . Ponieważ żądanie MQGET jest częścią UOW2 (patrz krok 7), to zatwierdzenie powoduje usunięcie komunikatu z kolejki.

**Krok 10**
Krok 8 nie zostanie zakończony zgodnie z wymaganiami, a aplikacja wycofa UOW2. Ponieważ żądanie pobrania komunikatu jest częścią UOW2 (patrz krok 7), również zostało wycofane i przywrócone do kolejki. Jest on teraz dostępny dla kolejnych wywołań MQGET wysyłanych przez tę lub inną aplikację (w taki sam sposób, jak każdy inny komunikat w kolejce).

## *Konwersja danych aplikacji*

Jeśli jest to konieczne, adaptery MCA przekształcają dane deskryptora komunikatu i nagłówka w wymagany zestaw znaków i kodowanie. Konwersję może wykonać dowolny koniec łącza (czyli lokalny agent MCA lub zdalny agent MCA).

Gdy aplikacja umieszcza komunikaty w kolejce, menedżer kolejek lokalnych dodaje informacje sterujące do deskryptorów komunikatów, aby ułatwić sterowanie komunikatami podczas ich przetwarzania przez menedżery kolejek i adaptery MCA. W zależności od środowiska pola danych nagłówka komunikatu są tworzone w zestawie znaków i kodowaniu systemu lokalnego.

Podczas przenoszenia komunikatów między systemami czasami konieczne jest przekształcenie danych aplikacji na zestaw znaków i kodowanie wymagane przez system odbierający. Można to zrobić z poziomu aplikacji w systemie odbierającym lub przez adaptery MCA w systemie wysyłającym. Jeśli konwersja danych jest obsługiwana w systemie odbierającym, należy użyć aplikacji do konwersji danych aplikacji, a nie w zależności od konwersji, która już wystąpiła w systemie wysyłającym.

Dane aplikacji są przekształcane w programie użytkowym po podaniu opcji MQGMO_CONVERT w polu *Options* struktury MQGMO przekazywanej do wywołania MQGET i po *wszystkich* spełnieniu następujących warunków:

- Pola *CodedCharSetId* lub *Encoding* ustawione w strukturze MQMD powiązanej z komunikatem w kolejce różnią się od pól *CodedCharSetId* lub *Encoding* ustawionych w strukturze MQMD określonej w wywołaniu MQGET.
- Pole *Format* w strukturze MQMD powiązanej z komunikatem nie ma wartości MQFMT_NONE.
- Wartość *BufferLength* określona w wywołaniu MQGET nie jest równa zero.
- Długość danych komunikatu jest różna od zera.
- Menedżer kolejek obsługuje konwersję między polami *CodedCharSetId* i *Encoding* określonymi w strukturach MQMD powiązanych z komunikatem i wywołaniem MQGET. Szczegółowe informacje na temat obsługiwanych identyfikatorów kodowanych zestawów znaków i kodowań maszynowych można znaleźć w sekcji CodedCharSetId i Encoding .
- Menedżer kolejek obsługuje konwersję formatu komunikatu. Jeśli pole *Format* struktury MQMD powiązanej z komunikatem jest jednym z wbudowanych formatów, menedżer kolejek może przekształcić komunikat. Jeśli *Format* nie jest jednym z wbudowanych formatów, należy napisać wyjście konwersji danych, aby przekształcić komunikat.

Jeśli wysyłający agent MCA ma przekształcić dane, należy określić słowo kluczowe CONVERT (YES) w definicji każdego kanału wysyłającego lub kanału serwera, dla którego konwersja jest wymagana. Jeśli konwersja danych nie powiedzie się, komunikat jest wysyłany do kolejki DLQ w nadawczym menedżerze kolejek, a pole *Feedback* w strukturze MQDLH wskazuje przyczynę. Jeśli nie można umieścić komunikatu w kolejce DLQ, kanał zostanie zamknięty, a nieprzekształcony komunikat pozostanie w kolejce transmisji.

Taka sytuacja pozwala uniknąć konwersji danych w obrębie aplikacji, a nie podczas wysyłania agentów MCA.

Z reguły dane w komunikacie opisane jako dane *znakowe* przez wbudowany format lub wyjście konwersji danych są konwertowane z kodowanego zestawu znaków używanego przez komunikat na żądany, a pola *liczbowe* są konwertowane na żądane kodowanie.

Więcej informacji na temat konwencji przetwarzania konwersji używanych podczas konwersji wbudowanych formatów oraz informacje o zapisywaniu własnych wyjść konwersji danych zawiera sekcja "Zapisywanie wyjść konwersji danych" na stronie 1013. Informacje na temat tabel obsługi języków i obsługiwanych kodowań znajdują się w sekcji Języki narodowe i Kodowania maszynowego .

## Konwersja znaków nowego wiersza EBCDIC

Aby upewnić się, że dane wysyłane z platformy EBCDIC na platformę ASCII są identyczne z danymi, które są ponownie odbierane, należy kontrolować konwersję znaków nowego wiersza EBCDIC.

Można to zrobić za pomocą zależnego od platformy przełącznika, który wymusza użycie niezmodyfikowanych tabel konwersji przez produkt IBM MQ , ale należy pamiętać o niespójnym zachowaniu, które może spowodować.

Problem pojawia się, ponieważ znak nowego wiersza EBCDIC nie jest poddawany spójnie konwersji między platformami lub tabelami konwersji. W rezultacie, jeśli dane są wyświetlane na platformie ASCII, formatowanie może być niepoprawne. Utrudniłoby to na przykład zdalne administrowanie systemem IBM i z poziomu platformy ASCII przy użyciu komendy RUNMQSC.

Więcej informacji na temat konwersji danych w formacie EBCDIC na format ASCII zawiera sekcja Konwersja danych .

### *Przeglądanie komunikatów w kolejce*

Ten temat zawiera informacje o przeglądaniu komunikatów w kolejce przy użyciu wywołania MQGET.

Aby użyć wywołania MQGET do przeglądania komunikatów w kolejce:

1. Wywołaj funkcję MQOPEN, aby otworzyć kolejkę do przeglądania, podając opcję MQOO_BROWSE.
2. Aby przejrzeć pierwszy komunikat w kolejce, wywołaj komendę MQGET z opcją MQGMO_BROWSE_FIRST. Aby znaleźć żądany komunikat, należy wielokrotnie wywołać komendę MQGET z opcją MQGMO_BROWSE_NEXT, aby przejść przez wiele komunikatów.

   Należy ustawić pola *MsgId* i *CorrelId* struktury MQMD na wartość NULL po każdym wywołaniu MQGET, aby wyświetlić wszystkie komunikaty.
3. Wywołaj komendę MQCLOSE, aby zamknąć kolejkę.

*Kursor przeglądania*
Po otwarciu (MQOPEN) kolejki do przeglądania wywołanie ustanawia kursor przeglądania do użycia z wywołaniami MQGET, które korzystają z jednej z opcji przeglądania. Kursor przeglądania można traktować jako wskaźnik logiczny, który znajduje się przed pierwszym komunikatem w kolejce.

Można mieć więcej niż jeden aktywny kursor przeglądania (z jednego programu), wysyłając kilka żądań MQOPEN dla tej samej kolejki.

Podczas wywoływania komendy MQGET w celu przeglądania należy użyć jednej z następujących opcji w strukturze MQGMO:

**MQGMO_BROWSE_FIRST**
Pobiera kopię pierwszego komunikatu, który spełnia warunki określone w strukturze MQMD.

**MQGMO_BROWSE_NEXT**
Pobiera kopię następnego komunikatu, który spełnia warunki określone w strukturze MQMD.

**MQGMO_BROWSE_MSG_UNDER_CURSOR**
Pobiera kopię komunikatu aktualnie wskazywanego przez kursor, czyli tego, który został ostatnio pobrany przy użyciu opcji MQGMO_BROWSE_FIRST lub MQGMO_BROWSE_NEXT.

We wszystkich przypadkach komunikat pozostaje w kolejce.

Po otwarciu kolejki kursor przeglądania jest umieszczany logicznie tuż przed pierwszym komunikatem w kolejce. Oznacza to, że jeśli wywołanie MQGET zostanie wykonane natychmiast po wywołaniu MQOPEN, można użyć opcji MQGMO_BROWSE_NEXT do przeglądania pierwszego komunikatu; nie ma potrzeby używania opcji MQGMO_BROWSE_FIRST.

Kolejność, w jakiej komunikaty są kopiowane z kolejki, jest określana przez atrybut **MsgDeliverySequence** kolejki. Więcej informacji na ten temat zawiera sekcja "Kolejność, w jakiej komunikaty są pobierane z kolejki" na stronie 798.


- "Kolejki w kolejności FIFO (pierwszy przyszedł, pierwszy wyszedł)" na stronie 829
- "Kolejki w kolejności priorytetów" na stronie 829
- "Niezatwierdzone komunikaty" na stronie 829
- "Zmień na sekwencję kolejki" na stronie 829
- "Korzystanie z indeksu kolejki" na stronie 830


## Kolejki w kolejności FIFO (pierwszy przyszedł, pierwszy wyszedł)

Pierwszym komunikatem w kolejce w tej kolejności jest komunikat, który był w kolejce najdłużej.

Użyj komendy MQGMO_BROWSE_NEXT, aby odczytać komunikaty sekwencyjnie w kolejce. Podczas przeglądania zostaną wyświetlone wszystkie komunikaty umieszczone w kolejce, ponieważ na końcu kolejki w tej kolejności znajdują się komunikaty. Gdy kursor rozpozna, że osiągnął koniec kolejki, kursor przeglądania pozostaje w miejscu, w którym się znajduje i zwraca wartość MQRC_NO_MSG_AVAILABLE. Następnie można pozostawić go tam, oczekując na kolejne komunikaty, lub zresetować go do początku kolejki za pomocą wywołania MQGMO_BROWSE_FIRST.

## Kolejki w kolejności priorytetów

Pierwszy komunikat w kolejce w tej kolejności to komunikat, który był w kolejce najdłużej i miał najwyższy priorytet w momencie wywołania MQOPEN.

Użyj komendy MQGMO_BROWSE_NEXT, aby odczytać komunikaty w kolejce.

Kursor przeglądania wskazuje na następny komunikat, od priorytetu pierwszego komunikatu do zakończenia z komunikatem o najniższym priorytecie. Przegląda wszystkie komunikaty umieszczone w kolejce w tym czasie, o ile mają one priorytet równy lub niższy niż komunikat identyfikowany przez bieżący kursor przeglądania.

Wszystkie komunikaty umieszczone w kolejce o wyższym priorytecie mogą być przeglądane tylko przez:

- Ponowne otwieranie kolejki do przeglądania, po czym tworzony jest nowy kursor przeglądania
- Korzystanie z opcji MQGMO_BROWSE_FIRST

## Niezatwierdzone komunikaty

Niezatwierdzony komunikat nigdy nie jest widoczny dla przeglądania; kursor przeglądania pomija go.

Komunikaty w jednostce pracy nie mogą być przeglądane, dopóki jednostka pracy nie zostanie zatwierdzona. Komunikaty nie zmieniają swojej pozycji w kolejce po zatwierdzeniu, dlatego pominięte niezatwierdzone komunikaty nie będą widoczne, nawet jeśli *są* zatwierdzone, chyba że zostanie użyta opcja MQGMO_BROWSE_FIRST i kolejka zostanie ponownie użyta.

## Zmień na sekwencję kolejki

Jeśli kolejność dostarczania komunikatów zostanie zmieniona z priorytetu na FIFO, gdy w kolejce znajdują się komunikaty, kolejność komunikatów, które są już w kolejce, nie zostanie zmieniona. Komunikaty dodane później do kolejki, przyjmują domyślny priorytet kolejki.

## Korzystanie z indeksu kolejki

Podczas przeglądania poindeksowanej kolejki, która zawiera tylko komunikaty o pojedynczym priorytecie (trwałe, nietrwałe lub oba), menedżer kolejek używa indeksu do przeglądania, gdy używane są określone formy przeglądania.

**Uwaga:** Obsługiwane tylko w systemie IBM MQ for z/OS.

Jeśli indeksowana kolejka zawiera tylko komunikaty o pojedynczym priorytecie, używana jest dowolna z następujących form przeglądania:

1. Jeśli kolejka jest indeksowana według identyfikatora MSGID, żądania przeglądania, które przekazują identyfikator MSGID w strukturze MQMD, są przetwarzane przy użyciu indeksu w celu znalezienia komunikatu docelowego.
2. Jeśli kolejka jest indeksowana według identyfikatora CORRELID, żądania przeglądania, które przekazały identyfikator CORRELID w strukturze MQMD, są przetwarzane przy użyciu indeksu w celu znalezienia komunikatu docelowego.
3. Jeśli kolejka jest indeksowana według identyfikatora GROUPID, żądania przeglądania, które przekazują identyfikator GROUPID w strukturze MQMD, są przetwarzane przy użyciu indeksu w celu znalezienia komunikatu docelowego.

Jeśli żądanie przeglądania nie przekaże identyfikatora MSGID, CORRELID lub GROUPID w strukturze MQMD, kolejka zostanie poindeksowana i zostanie zwrócony komunikat, pozycja indeksu dla komunikatu musi zostać znaleziona, a informacje w niej użyte do aktualizacji kursora przeglądania. Jeśli używany jest szeroki wybór wartości indeksu, nie powoduje to znaczącego dodatkowego przetwarzania żądania przeglądania.

*Przeglądanie komunikatów, gdy długość komunikatu jest nieznana*

Aby przeglądać komunikat, gdy wielkość komunikatu nie jest znana, a do znalezienia komunikatu nie mają być używane pola *MsgId*, *CorrelId*lub *GroupId* , można użyć opcji MQGMO_BROWSE_MSG_UNDER_CURSOR:

1. Wykonaj komendę MQGET z następującymi danymi:
   - Opcja MQGMO_BROWSE_FIRST lub MQGMO_BROWSE_NEXT
   - Opcja MQGMO_ACCEPT_TRUNCATED_MSG
   - Zerowa długość buforu

   **Uwaga:** Jeśli istnieje prawdopodobieństwo, że inny program będzie otrzymywał ten sam komunikat, należy rozważyć również użycie opcji MQGMO_LOCK. Należy zwrócić wartość MQRC_TRUNCATED_MSG_ACCEPTED.
2. Użyj zwróconej wartości *DataLength* , aby przydzielić potrzebną pamięć.
3. Uruchom komendę MQGET z kursorem MQGMO_BROWSE_MSG_UNDER_CURSOR.

Wskazywany komunikat jest ostatnim, który został pobrany; kursor przeglądania nie zostanie przeniesiony. Można zablokować komunikat przy użyciu opcji MQGMO_LOCK lub odblokować zablokowany komunikat przy użyciu opcji MQGMO_UNLOCK.

Wywołanie nie powiedzie się, jeśli żadne wywołanie MQGET z opcjami MQGMO_BROWSE_FIRST lub MQGMO_BROWSE_NEXT nie zostało pomyślnie wykonane od momentu otwarcia kolejki.

*Usuwanie przeglądniętej wiadomości*
Istnieje możliwość usunięcia z kolejki komunikatu, który został już przejrzany, pod warunkiem, że kolejka została otwarta w celu usunięcia komunikatów, a także w celu przeglądania. (W wywołaniu MQOPEN należy określić jedną z opcji MQOO_INPUT_ *, a także opcję MQOO_BROWSE).

Aby usunąć komunikat, wywołaj ponownie komendę MQGET, ale w polu *Options* struktury MQGMO podaj wartość MQGMO_MSG_UNDER_CURSOR.W tym przypadku wywołanie MQGET ignoruje pola *MsgId*, *CorrelId*i *GroupId* struktury MQMD.

W czasie między przeglądaniem i usuwaniem inny program mógł usunąć komunikaty z kolejki, w tym komunikat znajdujący się pod kursorem przeglądania. W takim przypadku wywołanie MQGET zwraca kod przyczyny informujący o tym, że komunikat jest niedostępny.

*Przeglądanie komunikatów w porządku logicznym*

"Porządkowanie logiczne i fizyczne" na stronie 798 wyjaśnia różnicę między logiczną i fizyczną kolejnością komunikatów w kolejce. To rozróżnienie jest szczególnie ważne podczas przeglądania kolejki, ponieważ komunikaty nie są usuwane, a operacje przeglądania nie muszą być uruchamiane na początku kolejki.

Jeśli aplikacja przegląda różne komunikaty z jednej grupy (używając kolejności logicznej), ważne jest, aby kolejność logiczna była stosowana w celu osiągnięcia początku następnej grupy, ponieważ ostatni komunikat z jednej grupy może wystąpić fizycznie *po* pierwszym komunikacie z następnej grupy. Opcja MQGMO_LOGICAL_ORDER zapewnia zachowanie porządku logicznego podczas skanowania kolejki.

Użyj MQGMO_ALL_MSGS_AVAILABLE (lub MQGMO_ALL_SEGMENTS_AVAILABLE) z ostrożnością dla operacji przeglądania. Rozważmy przypadek komunikatów logicznych z MQGMO_ALL_MSGS_AVAILABLE. W rezultacie komunikat logiczny jest dostępny tylko wtedy, gdy wszystkie pozostałe komunikaty w grupie również istnieją. Jeśli nie są, komunikat jest przekazywany. Może to oznaczać, że po odebraniu brakujących komunikatów nie są one zauważane przez operację przeglądania następnego komunikatu.

Na przykład, jeśli występują następujące komunikaty logiczne:

```
Logical message 1 (not last) of group 123
Logical message 1 (not last) of group 456
Logical message 2 (last)     of group 456
```

i wywołano funkcję przeglądania z MQGMO_ALL_MSGS_AVAILABLE, zwracany jest pierwszy komunikat logiczny grupy 456, pozostawiając kursor przeglądania w tym komunikacie logicznym. Jeśli pojawi się drugi (ostatni) komunikat grupy 123:

```
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
```

i ta sama funkcja przeglądania następnego, nie jest zauważona, że grupa 123 jest teraz kompletna, ponieważ pierwszy komunikat tej grupy jest *przed* kursorem przeglądania.

W niektórych przypadkach (na przykład jeśli komunikaty są pobierane ze zniszczeniem, gdy grupa jest obecna w całości) można użyć opcji MQGMO_ALL_MSGS_AVAILABLE razem z opcją MQGMO_BROWSE_FIRST. W przeciwnym razie należy powtórzyć skanowanie przeglądania, aby zanotować nowo odebrane komunikaty, które zostały pominięte. Po prostu wywołanie MQGMO_WAIT razem z MQGMO_BROWSE_NEXT i MQGMO_ALL_MSGS_AVAILABLE nie uwzględnia ich. (Dzieje się tak również w przypadku komunikatów o wyższym priorytecie, które mogą nadejść po zakończeniu skanowania wiadomości).

W następnych sekcjach przedstawiono przykłady przeglądania, które dotyczą komunikatów niesegmentowanych. Segmentowane komunikaty są zgodne z podobnymi zasadami.

*Przeglądanie komunikatów w grupach*
W tym przykładzie aplikacja przegląda każdy komunikat w kolejce w kolejności logicznej.

Komunikaty w kolejce mogą być zgrupowane. W przypadku zgrupowanych komunikatów aplikacja nie chce rozpocząć przetwarzania żadnej grupy, dopóki nie zostaną nadeszły wszystkie komunikaty w niej zawarte. Dlatego opcja MQGMO_ALL_MSGS_AVAILABLE jest określana dla pierwszego komunikatu w grupie; w przypadku kolejnych komunikatów w grupie ta opcja jest zbędna.

W tym przykładzie użyto komendy MQGMO_WAIT. Jednak oczekiwanie może być spełnione, jeśli pojawi się nowa grupa, z powodów określonych w sekcji "Przeglądanie komunikatów w porządku logicznym" na stronie 831, nie jest spełnione, jeśli kursor przeglądania przekazał już pierwszy komunikat logiczny w grupie, a pozostałe komunikaty zostały odebrane. Jednak oczekiwanie na odpowiedni odstęp czasu

zapewnia, że aplikacja nie będzie stale zapętlać się podczas oczekiwania na nowe komunikaty lub segmenty.

Parametr MQGMO_LOGICAL_ORDER jest używany w celu zapewnienia, że skanowanie odbywa się w porządku logicznym. Jest to przeciwieństwo destrukcyjnego przykładu MQGET, w którym, ponieważ każda grupa jest usuwana, parametr MQGMO_LOGICAL_ORDER nie jest używany podczas wyszukiwania pierwszego (lub jedynego) komunikatu w grupie.

Zakłada się, że bufor aplikacji jest zawsze wystarczająco duży, aby pomieścić cały komunikat, niezależnie od tego, czy komunikat został podzielony na segmenty. Dlatego w każdej operacji MQGET określono parametr MQGMO_COMPLETE_MSG.

Poniżej przedstawiono przykład przeglądania komunikatów logicznych w grupie:

```
/* 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 */
   ...
```

Grupa jest powtarzana do momentu zwrócenia wartości MQRC_NO_MSG_AVAILABLE.

*Przeglądanie i pobieranie ze zniszczeniem*
W tym przykładzie aplikacja przegląda każdy komunikat logiczny w grupie przed podjęciem decyzji o tym, czy grupa ma zostać pobrana ze zniszczeniem.

Pierwsza część tego przykładu jest podobna do poprzedniej. Jednak w tym przypadku, po przeglądnięciu całej grupy, zdecydowaliśmy się wrócić i odzyskać ją destrukcyjnie.

Ponieważ każda grupa jest usuwana w tym przykładzie, parametr MQGMO_LOGICAL_ORDER nie jest używany podczas wyszukiwania pierwszego lub jedynego komunikatu w grupie.

Poniżej przedstawiono przykład przeglądania, a następnie pobierania ze zniszczeniem:

```
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MESSAGES_AVAILABE | MQGMO_WAIT
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group (or as many as
      necessary to decide whether to get it destructively) */
   ...

if ( we want to retrieve the group destructively )

   if ( GroupStatus == ' ' )
      /* We retrieved an ungrouped message */
      GMO.Options = MQGMO_MSG_UNDER_CURSOR | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = 0
      /* Process the message */
      ...

   else
      /* We retrieved one or more messages in a group.  The browse cursor */
      /* will not normally be still on the first in the group, so we have */
      /* to match on the GroupId and MsgSeqNumber = 1.                    */
      /* Another way, which works for both grouped and ungrouped messages,*/
      /* would be to remember the MsgId of the first message when it was  */
      /* browsed, and match on that.                                      */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID
                             | MQMO_MATCH_MSG_SEQ_NUMBER,
           (MQMD.GroupId      = value already in the MD)
            MQMD.MsgSeqNumber = 1
      /* Process first or only message */
      ...
```

```
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
                    | MQGMO_LOGICAL_ORDER
      do while ( GroupStatus == MQGS_MSG_IN_GROUP )
         MQGET
         /* Process each remaining message in the group */
...
```

*Unikanie wielokrotnego dostarczania przeglądanych komunikatów*
Korzystając z niektórych opcji otwierania i pobierania komunikatów, można oznaczyć komunikaty jako przejrzane, aby nie były ponownie pobierane przez bieżącą lub inną współpracującą aplikację. Wiadomości mogą być oznaczane jawnie lub automatycznie, aby były ponownie dostępne do przeglądania.

Przeglądając komunikaty w kolejce, można je pobrać w innej kolejności niż ta, w której zostały pobrane w przypadku ich destrukcji. W szczególności ten sam komunikat można przeglądać wielokrotnie, co nie jest możliwe, jeśli zostanie usunięty z kolejki. Aby tego uniknąć, można *oznaczyć* komunikaty podczas przeglądania i uniknąć pobierania oznaczonych komunikatów. Jest to czasami nazywane *przeglądaniem ze znacznikiem*. Aby oznaczyć przejrzane komunikaty, należy użyć opcji pobierania komunikatów MQGMO_MARK_BROWSE_HANDLE, a aby pobrać tylko nieoznaczone komunikaty, należy użyć opcji MQGMO_UNMARKED_BROWSE_MSG. Jeśli zostanie użyta kombinacja opcji MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG i MQGMO_MARK_BROWSE_HANDLE, a następnie zostaną wydane powtarzające się komendy MQGET, każdy komunikat będzie pobierany z kolejki po kolei. Zapobiega to powtarzaniu dostarczania komunikatów, nawet jeśli komenda MQGMO_BROWSE_FIRST jest używana do zapewnienia, że komunikaty nie będą pomijane. Ta kombinacja opcji może być reprezentowana przez pojedynczą stałą MQGMO_BROWSE_HANDLE. Jeśli w kolejce nie ma komunikatów, które nie zostały przejrzane, zwracana jest wartość MQRC_NO_MSG_AVAILABLE.

Jeśli wiele aplikacji przegląda tę samą kolejkę, mogą otworzyć kolejkę z opcjami MQOO_CO_OP i MQOO_BROWSE. Uchwyt obiektu zwracany przez każde wywołanie MQOPEN jest traktowany jako część grupy współpracującej. Każdy komunikat zwrócony przez wywołanie MQGET z opcją MQGMO_MARK_BROWSE_CO_OP jest traktowany jako oznaczony dla tego współpracującego zestawu uchwytów.

Jeśli komunikat został oznaczony przez pewien czas, może zostać automatycznie anulowany przez menedżer kolejek i ponownie udostępniony do przeglądania. Atrybut menedżera kolejek MsgMarkBrowseInterval podaje czas w milisekundach, przez który komunikat ma pozostać oznaczony dla współpracującego zestawu uchwytów. Wartość MsgMarkBrowseInterval równa -1 oznacza, że komunikaty nigdy nie są automatycznie oznaczane.

Gdy pojedynczy proces lub zestaw procesów kooperatywnych oznaczających komunikaty zatrzymują się, wszystkie oznaczone komunikaty stają się nieoznaczone.

### Przykłady przeglądania kooperatywnego

Można uruchomić wiele kopii aplikacji przekaźnika w celu przeglądania komunikatów w kolejce i zainicjowania konsumenta na podstawie treści każdego komunikatu. W każdym programie rozsyłającym otwórz kolejkę za pomocą komendy MQOO_CO_OP. Oznacza to, że przekaźniki współpracują ze sobą i będą wiedzieć o sobie nawzajem oznaczonych komunikatach. Następnie każdy program rozsyłający wykonuje powtarzające się wywołania MQGET, określając opcje MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG i MQGMO_MARK_BROWSE_CO_OP (do reprezentowania tej kombinacji opcji można użyć pojedynczej stałej MQGMO_BROWSE_CO_OP). Następnie każda aplikacja programu rozsyłającego pobiera tylko te komunikaty, które nie zostały jeszcze oznaczone przez inne współpracujące programy rozsyłające. Program rozsyłający inicjuje konsument i przekazuje element MsgToken zwrócony przez komendę MQGET do konsumenta, który destrukcyjnie pobiera komunikat z kolejki. Jeśli konsument wycofa żądanie MQGET komunikatu, komunikat będzie dostępny dla jednej z przeglądarek w celu ponownego rozesłania, ponieważ nie jest już oznaczony. Jeśli konsument nie wykona operacji MQGET dla komunikatu, po przejściu komunikatu MsgMarkBrowseInterval , menedżer kolejek usuwa oznaczenie komunikatu dla współpracującego zestawu uchwytów i może zostać ponownie rozesłany.

Zamiast wielu kopii tej samej aplikacji programu rozsyłającego, może istnieć wiele różnych aplikacji programu rozsyłającego przeglądających kolejkę, z których każda jest odpowiednia do przetwarzania podzbioru komunikatów w kolejce. W każdym programie rozsyłającym otwórz kolejkę za pomocą komendy MQOO_CO_OP. Oznacza to, że przekaźniki współpracują ze sobą i będą wiedzieć o sobie nawzajem oznaczonych komunikatach.

- Jeśli kolejność przetwarzania komunikatów dla pojedynczego programu rozsyłającego jest istotna, każdy program rozsyłający wykonuje powtarzające się wywołania MQGET, określając opcje MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG i MQGMO_MARK_BROWSE_HANDLE (lub MQGMO_BROWSE_HANDLE). Jeśli przeglądany komunikat jest odpowiedni do przetworzenia przez ten program rozsyłający, wykonuje on wywołanie MQGET, określając opcje MQMO_MATCH_MSG_TOKEN, MQGMO_MARK_BROWSE_CO_OP i MsgToken zwrócone przez poprzednie wywołanie MQGET. Jeśli wywołanie powiedzie się, program rozsyłający inicjuje konsumenta, przekazując do niego MsgToken .
- Jeśli kolejność przetwarzania komunikatów nie jest istotna, a program rozsyłający ma przetwarzać większość napotkanych komunikatów, należy użyć opcji MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG i MQGMO_MARK_BROWSE_CO_OP (lub MQGMO_BROWSE_CO_OP). Jeśli program rozsyłający przegląda komunikat, którego nie może przetworzyć, usuwa zaznaczenie komunikatu, wywołując komendę MQGET z opcją MQMO_MATCH_MSG_TOKEN, MQGMO_UNMARK_BROWSE_CO_OP i zwrócił poprzednio wartość MsgToken .

### *Niektóre przypadki, w których wywołanie MQGET kończy się niepowodzeniem*

Jeśli niektóre atrybuty kolejki zostaną zmienione za pomocą opcji FORCE w komendzie między wywołaniem MQOPEN a wywołaniem MQGET, wywołanie MQGET zakończy się niepowodzeniem i zwróci kod przyczyny MQRC_OBJECT_CHANGED.

Menedżer kolejek oznacza uchwyt obiektu jako niepoprawny. Dzieje się tak również wtedy, gdy zmiany dotyczą dowolnej kolejki, na którą tłumaczona jest nazwa kolejki. Atrybuty wpływające w ten sposób na uchwyt są wymienione w opisie wywołania MQOPEN w tabeli MQOPEN. Jeśli wywołanie zwraca kod przyczyny MQRC_OBJECT_CHANGED, zamknij kolejkę, otwórz ją ponownie, a następnie spróbuj ponownie uzyskać komunikat.

Jeśli operacje pobierania są zablokowane dla kolejki, z której podejmowana jest próba pobrania komunikatów (lub dowolnej kolejki, na którą tłumaczona jest nazwa kolejki), wywołanie MQGET kończy się niepowodzeniem i zwraca kod przyczyny MQRC_GET_INHIBITED. Dzieje się tak nawet wtedy, gdy do przeglądania używane jest wywołanie MQGET. Jeśli w późniejszym czasie zostanie podjęta próba wywołania MQGET, komunikat może zostać wyświetlony pomyślnie, a projekt aplikacji będzie taki, że inne programy będą regularnie zmieniać atrybuty kolejek.

Jeśli kolejka dynamiczna (tymczasowa lub trwała) została usunięta, wywołania MQGET używające poprzednio uzyskanego uchwytu obiektu nie powiodą się i zwrócą kod przyczyny MQRC_Q_DELETED.

## Pisanie aplikacji publikowania/subskrybowania

Rozpocznij pisanie aplikacji IBM MQ publikowania/subskrypcji.

Przegląd pojęć związanych z publikowaniem/subskrybowaniem można znaleźć w sekcji Przesyłanie komunikatów w trybie publikowania/subskrybowania.

Informacje o pisaniu różnych typów aplikacji publikowania/subskrybowania można znaleźć w następujących tematach:

- “Pisanie aplikacji publikatora” na stronie 835
- “Pisanie aplikacji subskrybenta” na stronie 842
- “Cykle życia publikowania/subskrypcji” na stronie 859
- “Właściwości komunikatu publikowania/subskrypcji” na stronie 864
- “Porządkowanie komunikatów” na stronie 865
- “Przechwytywanie publikacji” na stronie 866

- “Opcje publikowania” na stronie 874
- “Opcje subskrypcji” na stronie 874

**Pojęcia pokrewne**

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć języka proceduralnego lub obiektowego. Przed rozpoczęciem projektowania i pisania aplikacji IBM MQ należy zapoznać się z podstawowymi pojęciami związanymi z produktem IBM MQ .

“Tworzenie aplikacji dla składnika IBM MQ” na stronie 5
Użytkownik może tworzyć aplikacje do wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i zasobami pokrewnymi. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i środowiskach.

“Uwagi dotyczące projektowania dla aplikacji IBM MQ” na stronie 51
Po podjęciu decyzji, w jaki sposób aplikacje mogą korzystać z dostępnych platform i środowisk, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

“Pisanie aplikacji proceduralnej dotyczącej kolejkowania” na stronie 744
Ten temat zawiera informacje o pisaniu aplikacji kolejkowania, nawiązywaniu i rozłączaniu połączeń z menedżerem kolejek, publikowaniu i subskrybowaniu oraz otwieraniu i zamykaniu obiektów.

“Pisanie aplikacji proceduralnych klienta” na stronie 941
Informacje potrzebne do pisania aplikacji klienckich w systemie IBM MQ przy użyciu języka proceduralnego.

“Budowanie aplikacji proceduralnej” na stronie 1029
Użytkownik może napisać aplikację IBM MQ w jednym z kilku języków proceduralnych i uruchomić ją na kilku różnych platformach.

“Obsługa błędów proceduralnych programu” na stronie 1069
Te informacje wyjaśniają błędy powiązane z wywołaniami MQI aplikacji podczas wykonywania wywołania lub po dostarczeniu komunikatu do miejsca docelowego.

**Zadania pokrewne**

“Korzystanie z przykładowych programów proceduralnych IBM MQ” na stronie 1089
Te przykładowe programy zostały napisane w językach proceduralnych i przedstawiają typowe zastosowania interfejsu kolejek komunikatów (Message Queue Interface-MQI). Programy IBM MQ na różnych platformach.

## *Pisanie aplikacji publikatora*

Zacznij pisać aplikacje wydawcy, analizując dwa przykłady. Pierwszy z nich jest modelowany tak blisko, jak to możliwe, w aplikacji umieszczającej komunikaty w kolejce, a drugi demonstruje dynamiczne tworzenie tematów-bardziej powszechny wzorzec dla aplikacji publikujących.

Pisanie prostej aplikacji publikatora IBM MQ jest podobne do pisania aplikacji typu IBM MQ punkt z punktem, która umieszcza komunikaty w kolejce (Tabela 121 na stronie 835). Różnica polega na tym, że komunikaty MQPUT są kierowane do tematu, a nie do kolejki.

*Tabela 121. Wzorzec programu IBM MQ typu punkt z punktem a publikowanie/subskrypcja.*

| Krok | Punkt z punktem MQ Call | Publikowanie wywołania produktu MQ |
|---|---|---|
| **Nawiązywanie połączenia z menedżerem kolejek** | ZMQCONN | ZMQCONN |
| **Otwórz kolejkę** | MQOPEN | |
| **Otwórz temat** | | MQOPEN |
| **Umieść komunikaty** | MQPUT | MQPUT |
| **Zamknij temat** | | MQCLOSE |

| *Tabela 121. Wzorzec programu IBM MQ typu punkt z punktem a publikowanie/subskrypcja. (kontynuacja)* | | |
|---|---|---|
| **Krok** | **Punkt z punktem MQ Call** | **Publikowanie wywołania produktu MQ** |
| **Zamknij kolejkę** | MQCLOSE | |
| **Rozłącz z menedżerem kolejek** | MQDISC | MQDISC |

Aby było to konkretne, istnieją dwa przykłady wniosków o publikację cen akcji. W pierwszym przykładzie (“Przykład 1: publikator do tematu stałego” na stronie 836), który jest ściśle modelowany podczas umieszczania komunikatów w kolejce, administrator tworzy definicję tematu w sposób podobny do tworzenia kolejki. Programista koduje MQPUT, aby zapisywać komunikaty w temacie zamiast zapisywać je w kolejce. W drugim przykładzie (“Przykład 2: publikator tematu zmiennej” na stronie 839) wzorzec interakcji programu z programem IBM MQ jest podobny. Różnica polega na tym, że programista udostępnia temat, w którym komunikat jest zapisywany, a nie administrator. W praktyce oznacza to zwykle, że łańcuch tematu jest zdefiniowany w treści lub jest udostępniany przez inne źródło, takie jak dane wejściowe wprowadzane przez użytkownika za pośrednictwem przeglądarki.

**Pojęcia pokrewne**

“Pisanie aplikacji subskrybenta” na stronie 842
Aby rozpocząć pisanie aplikacji subskrybentów, należy zapoznać się z trzema przykładami: aplikacją IBM MQ korzystającą z komunikatów z kolejki, aplikacją tworzącą subskrypcję, która nie wymaga wiedzy na temat kolejkowania, a także przykładem używającym zarówno kolejkowania, jak i subskrypcji.

**Odsyłacze pokrewne**

ZDEFINIUJ TEMAT
WYŚWIETL_TEMAT
STATUS DYSKU

*Przykład 1: publikator do tematu stałego*
Program IBM MQ ilustrujący publikowanie w temacie zdefiniowanym administracyjnie.

**Uwaga:** Zwarty styl kodowania jest przeznaczony do czytelności, a nie do użytku produkcyjnego.

Patrz dane wyjściowe w sekcji Rysunek 65 na stronie 837

```
#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);
}
```

*Rysunek 64. Prosty publikator IBM MQ do stałego tematu.*

```
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
```

*Rysunek 65. Przykładowe dane wyjściowe z pierwszego przykładu publikatora*

Poniższe wybrane wiersze kodu ilustrują aspekty pisania aplikacji publikatora dla produktu IBM MQ.

**char topicNameDefault[] = "IBMSTOCKPRICE";**
Domyślna nazwa tematu jest zdefiniowana w programie. Można go nadpisać, podając nazwę innego obiektu tematu jako pierwszy argument programu.

**MQCHAR resTopicStr[151];**
Element `resTopicStr` jest wskazywany przez element `td.ResObjectString.VSPtr` i jest używany przez element `MQOPEN` do zwracania rozstrzygniętego łańcucha tematu. Aby zwolnić miejsce na zakończenie zerowe, należy zwiększyć długość zmiennej `resTopicStr` o jeden większą niż długość przekazana w zmiennej `td.ResObjectString.VSBufSize` .

**memset (resTopicStr, 0, sizeof(resTopicStr));**
Zainicjuj łańcuch `resTopicStr` wartością NULL, aby upewnić się, że rozstrzygnięty łańcuch tematu zwrócony w elemencie `MQCHARV` jest zakończony wartością NULL.

**td.ObjectType = MQOT_TOPIC**
Istnieje nowy typ obiektu dla publikowania/subskrypcji: *obiekt tematu*.

**td.Version = MQOD_VERSION_4;**
Aby użyć nowego typu obiektu, należy użyć co najmniej *wersji 4* deskryptora obiektu.

**strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);**
`topicName` jest nazwą obiektu tematu, czasem nazywanego obiektem tematu administracyjnego. W tym przykładzie obiekt tematu musi zostać wcześniej utworzony przy użyciu Eksploratora IBM MQ lub tej komendy MQSC.

```
DEFINE TOPIC(IBMSTOCKPRICE) TOPICSTR(NYSE/IBM/PRICE) REPLACE;
```

**td.ResObjectString.VSPtr = resTopicStr;**
Rozstrzygnięty łańcuch tematu jest echowany w finalnym `printf` w programie. Skonfiguruj strukturę `MQCHARV ResObjectString` dla IBM MQ , aby zwrócić przetłumaczony łańcuch z powrotem do programu.

**MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);**
Otwórz temat dla danych wyjściowych, podobnie jak kolejkę dla danych wyjściowych.

**pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;**
Nowi subskrybenci mają mieć możliwość otrzymywania publikacji, a przez określenie parametru `MQPMO_RETAIN` w publikatorze, po uruchomieniu subskrybenta otrzymuje on najnowszą publikację opublikowaną przed uruchomieniem subskrybenta jako jego pierwszą zgodną publikację. Alternatywą jest udostępnienie subskrybentom publikacji opublikowanych dopiero po uruchomieniu subskrybenta. Dodatkowo subskrybent może odmówić otrzymania zachowanej publikacji, podając w swojej subskrypcji parametr `MQSO_NEW_PUBLICATIONS_ONLY` .

**MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);**
Dodaj 1 do długości łańcucha przekazanego do `MQPUT` w celu przekazania znaku zakończenia o kodzie zero do IBM MQ jako części buforu komunikatów.

Co pokazuje pierwszy przykład? Przykład ten naśladuje jak najbardziej wypróbowany i przetestowany tradycyjny wzorzec do pisania programów typu punkt z punktem IBM MQ . Ważną cechą wzorca programowania IBM MQ jest to, że programista nie jest zainteresowany wysyłaniem komunikatów. Zadaniem programisty jest nawiązanie połączenia z menedżerem kolejek i przekazanie mu komunikatów, które mają być dystrybuowane do odbiorców. W paradygmatu punkt z punktem programista otwiera kolejkę (prawdopodobnie kolejkę aliasową) skonfigurowaną przez administratora. Kolejka aliasowa kieruje komunikaty do kolejki docelowej w lokalnym menedżerze kolejek lub do zdalnego menedżera kolejek. Komunikaty oczekujące na dostarczenie są przechowywane w kolejkach między źródłem i miejscem docelowym.

We wzorcu publikowania/subskrypcji, zamiast otwierać kolejkę, programista otwiera temat. W tym przykładzie temat jest powiązany z łańcuchem tematu przez administratora. Menedżer kolejek przekazuje publikację przy użyciu kolejek do lokalnych lub zdalnych subskrybentów, którzy mają subskrypcje zgodne z łańcuchem tematu publikacji. Jeśli publikacje są zachowywane, menedżer kolejek przechowuje najnowszą kopię publikacji, nawet jeśli nie ma teraz subskrybentów. Zachowana publikacja jest dostępna

do przekazania przyszłym subskrybentom. Aplikacja publikatora nie odgrywa żadnej roli w wybieraniu lub kierowaniu publikacji do miejsca docelowego. Jej zadaniem jest tworzenie i umieszczanie publikacji w tematach zdefiniowanych przez administratora.

Ten stały przykład tematu jest nietypowy dla wielu aplikacji publikowania/subskrybowania: jest to temat statyczny. Wymaga, aby administrator zdefiniował łańcuchy tematów i zmienił tematy, w których są publikowane. Aplikacje publikowania/subskrypcji muszą znać niektóre lub wszystkie drzewa tematów. Być może tematy zmieniają się często lub chociaż tematy nie zmieniają się zbyt często, liczba kombinacji tematów jest duża i zdefiniowanie węzła tematu dla każdego łańcucha tematu, który może wymagać opublikowania, jest zbyt uciążliwe dla administratora. Być może łańcuchy tematów nie są znane przed publikacją. Aplikacja publikatora może używać informacji z treści publikacji do określenia łańcucha tematu lub może mieć informacje o łańcuchach tematów do opublikowania z innego źródła, takiego jak dane wejściowe użytkownika z przeglądarki. Aby uzyskać więcej dynamicznych stylów publikowania, w następnym przykładzie przedstawiono sposób dynamicznego tworzenia tematów w ramach aplikacji publikatora.

Tematy para wydawców i subskrybentów razem. Projektowanie reguł lub architektury na potrzeby nadawania nazw tematom i organizowania ich w drzewach tematów jest ważnym krokiem w tworzeniu rozwiązania publikowania/subskrybowania. Należy dokładnie przyjrzeć się, w jakim stopniu organizacja drzewa tematów wiąże razem programy publikatora i subskrybenta, a następnie wiąże je z treścią drzewa tematów. Zadaj sobie pytanie, czy zmiany w drzewie tematów mają wpływ na aplikacje publikatora i subskrybenta oraz w jaki sposób można zminimalizować ten efekt. Wbudowana w architekturę modelu publikowania/subskrypcji produktu IBM MQ jest pojęciem obiektu tematu administracyjnego, który udostępnia część główną (poddrzewo główne) tematu. Obiekt tematu udostępnia opcję administracyjnego definiowania głównej części drzewa tematów, która upraszcza programowanie i operacje aplikacji, a tym samym zwiększa możliwości konserwacji. Jeśli na przykład wdrażanych jest wiele aplikacji publikowania/subskrypcji, które mają izolowane drzewa tematów, to definiując administracyjnie główną część drzewa tematów, można zagwarantować izolację drzew tematów, nawet jeśli konwencje nazewnictwa tematów przyjęte przez różne aplikacje nie są spójne.

W praktyce, aplikacje wydawcy obejmują spektrum od wyłącznie przy użyciu stałych tematów, jak w tym przykładzie, i zmiennych tematów, jak w następnym. “Przykład 2: publikator tematu zmiennej” na stronie 839 demonstruje również łączenie użycia tematów i łańcuchów tematów.

**Pojęcia pokrewne**

“Przykład 2: publikator tematu zmiennej” na stronie 839
Program WebSphere MQ ilustrujący publikowanie w temacie zdefiniowanym programowo.

“Pisanie aplikacji subskrybenta” na stronie 842
Aby rozpocząć pisanie aplikacji subskrybentów, należy zapoznać się z trzema przykładami: aplikacją IBM MQ korzystającą z komunikatów z kolejki, aplikacją tworzącą subskrypcję, która nie wymaga wiedzy na temat kolejkowania, a także przykładem używającym zarówno kolejkowania, jak i subskrypcji.

*Przykład 2: publikator tematu zmiennej*
Program WebSphere MQ ilustrujący publikowanie w temacie zdefiniowanym programowo.

**Uwaga:** Zwarty styl kodowania jest przeznaczony do czytelności, a nie do użytku produkcyjnego.

Patrz dane wyjściowe w sekcji Rysunek 67 na stronie 840.

```
#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);
}
```

*Rysunek 66. Prosty publikator IBM MQ dla tematu zmiennej.*

```
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
```

*Rysunek 67. Przykładowe dane wyjściowe z przykładu drugiego publikatora*

Istnieje kilka punktów, które należy zwrócić uwagę na ten przykład.

**char topicNameDefault[] = "STOCKS";**
Domyślna nazwa tematu STOCKS definiuje część łańcucha tematu. Można nadpisać tę nazwę tematu, podając ją jako pierwszy argument programu, lub wyeliminować użycie nazwy tematu, podając / jako pierwszy parametr.

**char topicString[101] = "IBM/PRICE";**
IBM/PRICE jest domyślnym łańcuchem tematu. Ten łańcuch tematu można nadpisać, podając go jako drugi argument programu.

Menedżer kolejek łączy łańcuch tematu udostępniony przez STOCKS obiekt tematu "NYSE"z łańcuchem tematu udostępnionym przez program "IBM/PRICE" i wstawia łańcuch "/" między dwoma łańcuchami tematu. Wynikiem jest rozstrzygnięty łańcuch tematu "NYSE/IBM/PRICE". Wynikowy łańcuch tematu jest taki sam jak zdefiniowany w obiekcie tematu IBMSTOCKPRICE i ma dokładnie taki sam efekt.

Obiekt tematu administracyjnego powiązany z rozstrzygniętym łańcuchem tematu nie musi być tym samym obiektem tematu, który został przekazany do programu MQOPEN przez publikator. Produkt IBM MQ używa drzewa niejawnego w rozstrzygniętym łańcuchu tematu do określania, który obiekt tematu administracyjnego definiuje atrybuty powiązane z publikacją.

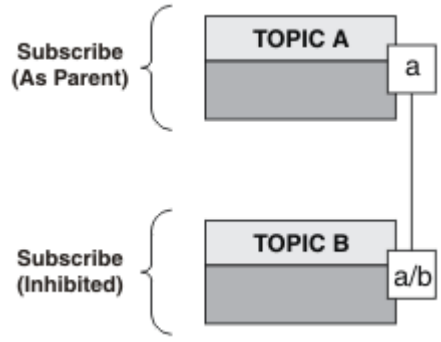
Załóżmy, że istnieją dwa obiekty tematu A i B, a A definiuje temat "a", a B definiuje temat "a/b" ( Rysunek 68 na stronie 841 ). Jeśli program publikujący odwołuje się do obiektu tematu A i udostępnia łańcuch tematu "b", rozstrzygając temat na łańcuch tematu "a/b", wówczas publikacja dziedziczy właściwości z obiektu tematu B , ponieważ temat jest zgodny z łańcuchem tematu "a/b" zdefiniowanym dla produktu B.

**if (strcmp(argv[1],"/"))**
argv[1] jest opcjonalnie podaną nazwą topicName. "/" jest niepoprawna jako nazwa tematu; oznacza to, że nie ma nazwy tematu, a łańcuch tematu jest udostępniany w całości przez program. Dane wyjściowe w sekcji Rysunek 67 na stronie 840 zawierają cały łańcuch tematu dostarczany dynamicznie przez program.

**strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);**
W domyślnym przypadku należy wcześniej utworzyć opcjonalny plik topicName przy użyciu Eksploratora IBM MQ lub następującej komendy MQSC:

```
DEFINE TOPIC(STOCKS) TOPICSTR(NYSE) REPLACE;
```

**td.ObjectString.VSPtr = topicString;**
Łańcuch tematu jest polem MQCHARV w deskryptorze tematu.




*Rysunek 68. Powiązania obiektów tematu*

Co pokazuje drugi przykład? Chociaż kod jest bardzo podobny do pierwszego przykładu-w rzeczywistości są tylko dwie linie różnicy-wynik jest znacznie inny program niż pierwszy. Programista steruje miejscami docelowymi, do których są wysyłane publikacje. W połączeniu z minimalnymi danymi wejściowymi

administratora używanymi do projektowania aplikacji subskrybenta nie ma potrzeby predefiniowania tematów ani kolejek w celu kierowania publikacji z publikatorów do subskrybentów.

W przypadku paradygmatu przesyłania komunikatów w trybie punkt z punktem kolejki muszą zostać zdefiniowane, zanim komunikaty będą mogły przepływać. W przypadku publikowania/subskrypcji nie są one wykonywane, chociaż produkt IBM MQ implementuje publikowanie/subskrypcję przy użyciu bazowego systemu kolejkowania. Korzyści wynikające z gwarantowanego dostarczania, transakcyjności i luźnego powiązania związanego z przesyłaniem komunikatów i kolejkowaniem są dziedziczone przez aplikacje publikowania/subskrypcji.

Projektant musi zdecydować, czy wydawca i subskrybent mają być świadomi bazowego drzewa tematów, czy też nie, a także czy programy subskrybentów mają być poinformowane o kolejkowaniu. Następnie należy przeanalizować przykładowe aplikacje subskrybenta. Są one przeznaczone do użytku z przykładami publikatorów, zwykle publikującymi i subskrybującymi produkt `NYSE/IBM/PRICE`.

**Pojęcia pokrewne**

“Przykład 1: publikator do tematu stałego” na stronie 836
Program IBM MQ ilustrujący publikowanie w temacie zdefiniowanym administracyjnie.

“Pisanie aplikacji subskrybenta” na stronie 842
Aby rozpocząć pisanie aplikacji subskrybentów, należy zapoznać się z trzema przykładami: aplikacją IBM MQ korzystającą z komunikatów z kolejki, aplikacją tworzącą subskrypcję, która nie wymaga wiedzy na temat kolejkowania, a także przykładem używającym zarówno kolejkowania, jak i subskrypcji.

## *Pisanie aplikacji subskrybenta*

Aby rozpocząć pisanie aplikacji subskrybentów, należy zapoznać się z trzema przykładami: aplikacją IBM MQ korzystającą z komunikatów z kolejki, aplikacją tworzącą subskrypcję, która nie wymaga wiedzy na temat kolejkowania, a także przykładem używającym zarówno kolejkowania, jak i subskrypcji.

W produkcie Tabela 122 na stronie 842 wymienione są trzy style konsumenta lub subskrybenta wraz z sekwencjami wywołań funkcji IBM MQ , które je charakteryzują.

1. Pierwszy styl, MQ Publication Consumer, jest identyczny z punktem w którym znajduje się program MQ , który wykonuje tylko MQGET. Aplikacja nie wie, że zajmuje się publikacjami-po prostu odczytuje komunikaty z kolejki. Subskrypcja, która powoduje, że publikacje są kierowane do kolejki, jest tworzona administracyjnie przy użyciu programu IBM MQ Explorer lub komendy.
2. Drugi styl jest preferowanym wzorcem dla większości aplikacji subskrybenta. Aplikacja subskrybenta tworzy subskrypcję, a następnie pobiera publikacje. Wszystkie operacje zarządzania kolejkami są wykonywane przez menedżer kolejek. Jest to tzw. *subskrybent zarządzany*.
3. W trzecim stylu aplikacja subskrybenta jest odpowiedzialna za określenie kolejki, która będzie używana do przechowywania publikacji, otwierania i zamykania tej kolejki oraz wydawania subskrypcji w celu wypełnienia kolejki publikacjami. Jest to tzw. *niezarządzany subskrybent*.

Jednym ze sposobów zrozumienia tych stylów jest zapoznanie się z przykładowymi programami C wymienionymi w sekcji Tabela 122 na stronie 842 dla każdego ze stylów. Przykłady są przeznaczone do uruchamiania w połączeniu z przykładowym publikatorem, który znajduje się w katalogu “Pisanie aplikacji publikatora” na stronie 835.

*Tabela 122. Wskaż i zasubskrybuj wzorce programu IBM MQ .*

| **Krok** | **Konsument komunikatów produktu MQ** | **“Przykład 1: konsument publikacji produktu MQ” na stronie 843** | **“Przykład 2: Zarządzany subskrybent produktu MQ” na stronie 845** | **“Przykład 3: Niezarządzany subskrybent produktu MQ” na stronie 850** |
|---|---|---|---|---|
| **Nawiązywanie połączenia z menedżerem kolejek** | ZMQCONN | ZMQCONN | ZMQCONN | ZMQCONN |

| *Tabela 122. Wskaż i zasubskrybuj wzorce programu IBM MQ . (kontynuacja)* | | | | |
|---|---|---|---|---|
| **Krok** | **Konsument komunikatów produktu MQ** | **“Przykład 1: konsument publikacji produktu MQ” na stronie 843** | **“Przykład 2: Zarządzany subskrybent produktu MQ” na stronie 845** | **“Przykład 3: Niezarządzany subskrybent produktu MQ” na stronie 850** |
| **Otwórz kolejkę** | MQOPEN | MQOPEN | | MQOPEN |
| **Subskrybowanie** | | | MQSUB | MQSUB |
| **Pobierz komunikaty** | MQGET | MQGET | MQGET | MQGET |
| **Zamknij kolejkę** | MQCLOSE | MQCLOSE | (MQCLOSE) | MQCLOSE |
| **Zamknij subskrypcję** | | | MQCLOSE | MQCLOSE |
| **Rozłącz z menedżerem kolejek** | MQDISC | MQDISC | MQDISC | MQDISC |

Użycie opcji MQCLOSE jest zawsze opcjonalne, zarówno w celu zwolnienia zasobów, przekazania opcji MQCLOSE, jak i w celu uzyskania symetrii z opcją MQOPEN. Ponieważ mało prawdopodobne jest określenie opcji MQCLOSE podczas zamykania kolejki subskrypcji w przypadku zarządzanego subskrybenta produktu MQ , a argument symetrii nie jest istotny, kolejka subskrypcji nie jest jawnie zamykana w Przykład 2: Zarządzany subskrybent produktu MQ.

Innym sposobem zrozumienia wzorców aplikacji publikowania/subskrypcji jest zbyt spojrzenie na interakcje między różnymi zaangażowanymi jednostkami. Linie życia lub diagramy sekwencji UML są dobrym sposobem na badanie interakcji. Trzy przykłady linii życia zostały opisane w sekcji “Cykle życia publikowania/subskrypcji” na stronie 859.

*Przykład 1: konsument publikacji produktu MQ*
Konsument publikacji MQ jest konsumentem komunikatów IBM MQ , który nie subskrybuje samych tematów.

Aby utworzyć kolejkę subskrypcji i publikacji dla tego przykładu, uruchom następujące komendy lub zdefiniuj obiekty za pomocą programu IBM MQ Explorer.

```
DEFINE QLOCAL(STOCKTICKER) REPLACE;
DEFINE SUB(IBMSTOCKPRICESUB) DEST(STOCKTICKER) TOPICOBJ(IBMSTOCKPRICE) REPLACE;
```

Subskrypcja programu `IBMSTOCKPRICESUB` odwołuje się do obiektu tematu `IBMSTOCK` utworzonego dla przykładu publikatora i kolejki lokalnej `STOCKTICKER`. Obiekt tematu `IBMSTOCK` definiuje łańcuch tematu, który jest używany w subskrypcji `NYSE/IBM/PRICE`. Należy zauważyć, że obiekt tematu i kolejka używana do odbierania publikacji muszą zostać zdefiniowane przed utworzeniem subskrypcji.

Wzorzec konsumenta publikacji produktu MQ zawiera wiele cennych aspektów:

1. Multiprocessing: udostępnianie poza pracą odczytu publikacji. Wszystkie publikacje są umieszczane w pojedynczej kolejce powiązanej z tematem subskrypcji. Kolejka może zostać otwarta przez wielu konsumentów za pomocą programu `MQOO_INPUT_SHARED`.
2. Subskrypcje zarządzane centralnie. Aplikacje nie tworzą własnych tematów subskrypcji ani subskrypcji. Administrator jest odpowiedzialny za miejsce, do którego są wysyłane publikacje.
3. Koncentracja subskrypcji: do pojedynczej kolejki można wysłać wiele różnych subskrypcji.
4. Trwałość subskrypcji: kolejka odbiera wszystkie publikacje, niezależnie od tego, czy konsumenci są aktywni.
5. Migracja i współistnienie: kod konsumenta działa równie dobrze w przypadku scenariusza typu punkt z punktem i publikowania/subskrypcji.

Subskrypcja tworzy relację między łańcuchem tematu NYSE/IBM/PRICE a kolejką STOCKTICKER. Publikacje, w tym wszystkie obecnie przechowywane publikacje, są przekazywane do programu STOCKTICKER od momentu utworzenia subskrypcji.

Subskrypcja utworzona administracyjnie może być zarządzana lub niezarządzana. Subskrypcja zarządzana zaczyna obowiązywać natychmiast po jej utworzeniu, podobnie jak subskrypcja niezarządzana. Nie wszystkie aspekty wzorca są dostępne dla subskrypcji zarządzanej. Więcej informacji znajduje się w sekcji “Przykład 3: Niezarządzany subskrybent produktu MQ” na stronie 850

**Uwaga:** Zwarty styl kodowania jest przeznaczony do czytelności, a nie do użytku produkcyjnego.

---

Wyniki są wyświetlane w sekcji Rysunek 70 na stronie 845.

```
#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);
}
```

*Rysunek 69. Konsument publikacji MQ .*

---

```
X:\Subscribe1\Debug>Subscribe1
Optional parameter: subscriptionQueue
Waiting 10 seconds for publications from STOCKTICKER
Received publication "129"
Completion code 0 and Return code 0
```

*Rysunek 70. Dane wyjściowe konsumenta publikacji produktu MQ*

Istnieje kilka standardowych wskazówek dotyczących programowania w języku IBM MQ *C* , które należy wziąć pod uwagę:

**memset(publication, 0, sizeof(publicationBuffer));**
Upewnij się, że na końcu komunikatu znajduje się wartość NULL, aby ułatwić formatowanie przy użyciu elementu `printf`. Przykład publikatora zawiera końcową wartość NULL w buforze komunikatów przekazanym do programu MQPUT przez dodanie 1 do pliku `strlen(publication)`. Ustawienie wartości NULL dla buforów MQCHAR jest dobrym stylem programowania dla programów w języku C systemu IBM MQ , które używają buforów do przechowywania łańcuchów, dzięki czemu wartość NULL występuje po tablicy znaków, które nie zapełniają całkowicie buforu.

**MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen, &CompCode, &Reason);**
Należy zarezerwować jedną wartość NULL na końcu buforu komunikatów, aby upewnić się, że zwrócony komunikat ma wartość NULL na końcu, jeśli parametr `if (messlen == strlen(publication));` ma wartość true. Ta wskazówka uzupełnia poprzednią i zapewnia, że w pliku `publicationBuffer` istnieje co najmniej jedna wartość NULL, która nie jest nadpisywana przez treść pliku `publication`.

**Pojęcia pokrewne**

“Przykład 2: Zarządzany subskrybent produktu MQ” na stronie 845
Zarządzany subskrybent produktu MQ jest preferowanym wzorcem dla większości aplikacji subskrybenta. Subskrypcja zarządzana to taka, w której program IBM MQ obsługuje subskrypcję i rejestruje ją oraz wyrejestrowuje. W tym przykładzie nie jest wymagana *żadna* definicja administracyjna kolejek, tematów ani subskrypcji.

“Przykład 3: Niezarządzany subskrybent produktu MQ” na stronie 850
Niezarządzany subskrybent jest ważną klasą aplikacji subskrybenta. Dzięki temu można połączyć korzyści wynikające z publikowania/subskrybowania z *sterowaniem* kolejkowaniem i korzystaniem z publikacji. Niezarządzana subskrypcja jest miejscem, w którym aplikacja jest odpowiedzialna. w celu określenia kolejki, w której przechowywane są subskrypcje. W przykładzie przedstawiono różne sposoby łączenia subskrypcji i kolejek.

“Pisanie aplikacji publikatora” na stronie 835
Zacznij pisać aplikacje wydawcy, analizując dwa przykłady. Pierwszy z nich jest modelowany tak blisko, jak to możliwe, w aplikacji umieszczającej komunikaty w kolejce, a drugi demonstruje dynamiczne tworzenie tematów-bardziej powszechny wzorzec dla aplikacji publikujących.

*Przykład 2: Zarządzany subskrybent produktu MQ*
Zarządzany subskrybent produktu MQ jest preferowanym wzorcem dla większości aplikacji subskrybenta. Subskrypcja zarządzana to taka, w której program IBM MQ obsługuje subskrypcję i rejestruje ją oraz wyrejestrowuje. W tym przykładzie nie jest wymagana *żadna* definicja administracyjna kolejek, tematów ani subskrypcji.

Ten najprostszy rodzaj zarządzanego subskrybenta zazwyczaj używa subskrypcji *nietrwałej* . Przykład koncentruje się na nietrwałej subskrypcji. Subskrypcja trwa tylko przez cały czas trwania uchwytu subskrypcji z programu MQSUB. Wszystkie publikacje, które są zgodne z łańcuchem tematu w czasie życia subskrypcji, są wysyłane do kolejki subskrypcji (i ewentualnie z zachowaną publikacją, jeśli flaga MQSO_NEW_PUBLICATIONS_ONLY nie jest ustawiona lub ustawiona jako domyślna, poprzednia publikacja zgodna z łańcuchem tematu została zachowana, a publikacja była trwała lub menedżer kolejek nie zakończył działania od momentu utworzenia publikacji).

Z tym wzorcem można również użyć *trwałej* subskrypcji. Zwykle, jeśli używana jest zarządzana subskrypcja trwała, jest ona wykonywana ze względu na niezawodność, a nie w celu ustanowienia subskrypcji, która bez wystąpienia błędów przewyższa subskrybent. Więcej informacji na temat różnych

cykli życia powiązanych z subskrypcjami zarządzanymi, niezarządzanymi, trwałymi i nietrwałymi można znaleźć w sekcji tematów pokrewnych.

Subskrypcje trwałe są często powiązane z publikacjami trwałymi, a subskrypcje nietrwałe z publikacjami nietrwałymi, ale nie ma koniecznej relacji między trwałością subskrypcji a trwałością publikacji. Wszystkie cztery kombinacje trwałości i trwałości są możliwe.

W przypadku rozważanego przypadku nietrwałego zarządzania menedżer kolejek tworzy kolejkę subskrypcji, która jest czyszczona i usuwana po zamknięciu kolejki. Publikacje są usuwane z kolejki po zamknięciu subskrypcji nietrwałej.

Cenne aspekty zarządzanego wzorca nietrwałego, których przykładem jest ten kod, są następujące:

1. Subskrypcja na żądanie: łańcuch tematu subskrypcji jest dynamiczny. Jest on udostępniany przez aplikację podczas działania.
2. Kolejka samozarządzająca: kolejka subskrypcji jest samodefiniująca i zarządzająca.
3. Cykl życia subskrypcji samozarządzającej: subskrypcje *nietrwałe* istnieją tylko na czas trwania aplikacji subskrybenta.
   - Jeśli zostanie zdefiniowana *trwała* subskrypcja zarządzana, spowoduje to utworzenie trwałej kolejki subskrypcji, w której będą przechowywane publikacje bez aktywnych programów subskrybentów. Menedżer kolejek usuwa kolejkę (i usuwa z niej wszystkie niepobrane publikacje) dopiero po usunięciu subskrypcji przez aplikację lub administratora. Subskrypcję można usunąć za pomocą komendy administracyjnej lub przez zamknięcie subskrypcji z opcją `MQCO_REMOVE_SUB` .
   - Należy rozważyć ustawienie parametru `SubExpiry` dla trwałych subskrypcji, tak aby publikacje przestały być wysyłane do kolejki, a subskrybent może wykorzystać wszystkie pozostałe publikacje przed usunięciem subskrypcji i spowodować, że menedżer kolejek usunie kolejkę i wszystkie pozostałe publikacje z tej kolejki.
4. Elastyczne wdrażanie łańcucha tematu: zarządzanie tematem subskrypcji jest uproszczone przez zdefiniowanie głównej części subskrypcji przy użyciu tematu zdefiniowanego administracyjnie. Główna część drzewa tematów jest następnie ukryta przed aplikacją. Ukrywając część główną, można wdrożyć aplikację bez nieumyślnego tworzenia przez aplikację drzewa tematów, które nakłada się na inne drzewo tematów utworzone przez inną instancję lub inną aplikację.
5. Tematy administrowane: przy użyciu łańcucha tematu, w którym pierwsza część jest zgodna z administracyjnym obiektem tematu, publikacje są zarządzane zgodnie z atrybutami obiektu tematu.
   - Jeśli na przykład pierwsza część łańcucha tematu jest zgodna z łańcuchem tematu powiązanym z klastrowym obiektem tematu, subskrypcja może odbierać publikacje z innych elementów klastra.
   - Selektywne dopasowywanie zdefiniowanych administracyjnie obiektów tematów i zdefiniowanych programowo subskrypcji umożliwia połączenie obu tych korzyści. Administrator udostępnia atrybuty dla tematów, a programista dynamicznie definiuje podtematy bez konieczności przejmowania się zarządzaniem tematami.
   - Jest to wynikowy łańcuch tematu, który jest używany w celu dopasowania do obiektu tematu, który udostępnia atrybuty powiązane z tematem, a niekoniecznie obiekt tematu o nazwie określonej w sekcji `sd.Objectname`, chociaż zwykle jest to jeden i ten sam obiekt tematu. Patrz “Przykład 2: publikator tematu zmiennej” na stronie 839.

Dzięki ustawieniu w przykładzie trwałej subskrypcji publikacje będą nadal wysyłane do kolejki subskrypcji po zamknięciu subskrypcji przez subskrybenta z opcją `MQCO_KEEP_SUB` . Kolejka nadal odbiera publikacje, gdy subskrybent nie jest aktywny. To zachowanie można przesłonić, tworząc subskrypcję za pomocą opcji `MQSO_PUBLICATIONS_ON_REQUEST` i używając komendy `MQSUBRQ` w celu zażądania zachowanej publikacji.

Subskrypcję można wznowić później, otwierając ją z opcją `MQCO_RESUME` .

Uchwytu kolejki `Hobj`, zwróconego przez `MQSUB` , można użyć na wiele sposobów. Uchwyt kolejki jest używany w przykładzie do uzyskiwania informacji o nazwie kolejki subskrypcji. Kolejki zarządzane są otwierane za pomocą domyślnych kolejek modelowych `SYSTEM.NDURABLE.MODEL.QUEUE` lub `SYSTEM.DURABLE.MODEL.QUEUE`. Wartości domyślne można przesłonić, udostępniając własne trwałe

i nietrwałe kolejki modelowe tematu według tematu jako właściwości obiektu tematu powiązanego z subskrypcją.

Niezależnie od atrybutów dziedziczonych z kolejek modelowych nie można ponownie wykorzystać uchwytu kolejki zarządzanej w celu utworzenia dodatkowej subskrypcji. Nie można również uzyskać innego uchwytu dla kolejki zarządzanej, otwierając ją po raz drugi przy użyciu zwróconej nazwy kolejki. Kolejka zachowuje się tak, jakby była otwarta na wyłączne wejście.

Kolejki niezarządzane są bardziej elastyczne niż kolejki zarządzane. Można na przykład współużytkować kolejki niezarządzane lub zdefiniować wiele subskrypcji w jednej kolejce. W następnym przykładzie przedstawiono sposób łączenia subskrypcji z niezarządzaną kolejką subskrypcji.

**Uwaga:** Zwarty styl kodowania jest przeznaczony do czytelności, a nie do użytku produkcyjnego.

---

Wyniki są wyświetlane w sekcji Rysunek 73 na stronie 849.

```
#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);
}
```

*Rysunek 71. Zarządzany subskrybent produktu MQ -część 1: deklaracje i obsługa parametrów.*

---

Istnieje kilka dodatkowych komentarzy na temat deklaracji w tym przykładzie.

**MQHOBJ Hobj = MQHO_NONE;**
Nie można jawnie otworzyć nietrwałej kolejki subskrypcji zarządzanej w celu odbierania publikacji, ale należy przydzielić pamięć dla uchwytu obiektu, który jest zwracany przez menedżer kolejek po otwarciu kolejki. Ważne jest, aby zainicjować uchwyt w pliku MQHO_OBJECT. Wskazuje to menedżerowi kolejek, że musi zwrócić uchwyt kolejki do kolejki subskrypcji.

**MQSD sd = {MQSD_DEFAULT};**
Nowy deskryptor subskrypcji używany w pliku MQSUB.

**MQCHAR48 qName;**
Chociaż w przykładzie nie jest wymagana znajomość kolejki subskrypcji, w przykładzie jest to zapytanie o nazwę kolejki subskrypcji-powiązanie MQINQ jest nieco niezręczne w języku C, więc ta część przykładu może być przydatna do przestudiowania.

```
    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;
}
```

*Rysunek 72. Zarządzany subskrybent produktu MQ -część 2: treść kodu.*

```
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
```

*Rysunek 73. Subskrybent produktu MQ*

Istnieje kilka dodatkowych komentarzy dotyczących kodu w tym przykładzie.

**strncpy(sd.ObjectName, topicName, MQ_Q_NAME_LENGTH);**
Jeśli właściwość topicName ma wartość NULL lub jest pusta *(wartość domyślna)*, nazwa tematu nie jest używana do obliczania rozstrzygniętego łańcucha tematu.

**sd.ObjectString.VSPtr = topicString;**
Zamiast używać predefiniowanego obiektu tematu, w tym przykładzie programista udostępnia obiekt tematu i łańcuch tematu, które są łączone przez program MQSUB. Należy zauważyć, że łańcuch tematu jest strukturą MQCHARV .

**sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;**
Alternatywą dla ustawiania długości pola MQCHARV .

**sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING;**
Po zdefiniowaniu łańcucha tematu należy zwrócić szczególną uwagę na flagi sd.Options . Istnieje wiele opcji, przykład określa tylko najczęściej używane opcje. Inne opcje używają wartości domyślnych.

1. Ponieważ subskrypcja jest *nietrwała*, czyli ma czas życia otwartej subskrypcji w aplikacji, należy ustawić flagę MQSO_CREATE . Można również ustawić flagę *(domyślnie)* MQSO_NON_DURABLE w celu zapewnienia czytelności.
2. Dopełnieniem MQSO_CREATE jest MQSO_RESUME. Obie flagi można ustawić razem. Menedżer kolejek tworzy nową subskrypcję lub wznawia istniejącą subskrypcję, w zależności od tego, co jest odpowiednie. Jeśli jednak zostanie podana wartość MQSO_RESUME , należy również zainicjować strukturę MQCHARV dla sd.SubName, nawet jeśli nie ma subskrypcji do wznowienia. Jeśli parametr SubName nie zostanie zainicjowany, zwrócony zostanie kod powrotu 2440: MQRC_SUB_NAME_ERROR z komendy MQSUB.

   **Uwaga:** Parametr MQSO_RESUME jest zawsze ignorowany w przypadku nietrwałej subskrypcji zarządzanej, ale określenie jej bez inicjowania struktury MQCHARV dla sd.SubName powoduje wystąpienie błędu.
3. Ponadto istnieje trzecia flaga wpływająca na sposób otwierania subskrypcji: MQSO_ALTER. Jeśli uprawnienia są poprawne, właściwości wznowionej subskrypcji są zmieniane, aby były zgodne z innymi atrybutami określonymi w parametrze MQSUB.

   **Uwaga:** Należy podać co najmniej jedną z opcji MQSO_CREATE, MQSO_RESUME i MQSO_ALTER . Patrz Opcje (MQLONG). Istnieją przykłady użycia wszystkich trzech flag w pliku “Przykład 3: Niezarządzany subskrybent produktu MQ” na stronie 850.
4. Ustaw wartość MQSO_MANAGED dla menedżera kolejek, aby automatycznie zarządzać subskrypcją.

**sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;**
Opcjonalnie można pominąć ustawianie długości parametru MQCHARV dla łańcuchów zakończonych znakiem o kodzie zero i zamiast niego użyć flagi zakończenia znakiem o kodzie zero.

**sd.ResObjectString.VSPtr = resTopicStr;**
Wynikowy łańcuch tematu jest odbity w pierwszym `printf` w programie. Skonfiguruj program `MQCHARV ResObjectString` for IBM MQ , aby zwrócić przetłumaczony łańcuch z powrotem do programu.

**Uwaga:** Wartość `resTopicStringBuffer` jest inicjowana wartościami null w `memset(resTopicStr, 0, sizeof(resTopicStrBuffer))`. Zwrócone łańcuchy tematów nie kończą się wartością NULL.

**sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;**
Ustaw wielkość buforu `sd.ResObjectString` na wartość o jeden mniejszą niż rzeczywista wielkość buforu. Zapobiega to nadpisaniu udostępnionego pustego terminatora na wypadek, gdyby rozstrzygnięty łańcuch tematu zapełnił cały bufor.

**Uwaga:** Jeśli łańcuch tematu jest dłuższy niż `sizeof(resTopicStrBuffer)-1`, nie jest zwracany żaden błąd. Nawet jeśli `VSLength > VSBufSiz` długość zwrócona w `sd.ResObjectString.VSLength` jest długością całego łańcucha i niekoniecznie długością zwróconego łańcucha. Przetestuj `sd.ResObjectString.VSLength < sd.ResObjectString.VSBufSiz` , aby potwierdzić, że łańcuch tematu jest kompletny.

**MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);**
Funkcja `MQSUB` tworzy subskrypcję. Jeśli jest nietrwała, prawdopodobnie nie jest interesowana jej nazwa, ale można sprawdzić jej status w Eksploratorze IBM MQ . Jako dane wejściowe można podać parametr `sd.SubName` , dzięki czemu wiadomo, jakiej nazwy należy szukać. Należy oczywiście unikać konfliktów nazw z innymi subskrypcjami.

**MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);**
Zamknięcie zarówno subskrypcji, jak i kolejki subskrypcji jest opcjonalne. W tym przykładzie subskrypcja jest zamknięta, ale nie jest kolejką. W tym przypadku opcja `MQCLOSE MQCO_REMOVE_SUB` jest domyślna, ponieważ subskrypcja jest nietrwała. Użycie `MQCO_KEEP_SUB` jest błędem.

**Uwaga:** *Kolejka* subskrypcji nie jest zamykana przez program `MQSUB`, a jej uchwyt `Hobj`pozostaje ważny do momentu zamknięcia kolejki przez program `MQCLOSE` lub `MQDISC`. Jeśli aplikacja zostanie przedwcześnie zakończona, kolejka i subskrypcja zostaną wyczyszczone przez menedżer kolejek po zakończeniu aplikacji.

**Pojęcia pokrewne**

“Przykład 1: konsument publikacji produktu MQ” na stronie 843
Konsument publikacji MQ jest konsumentem komunikatów IBM MQ , który nie subskrybuje samych tematów.

“Przykład 3: Niezarządzany subskrybent produktu MQ” na stronie 850
Niezarządzany subskrybent jest ważną klasą aplikacji subskrybenta. Dzięki temu można połączyć korzyści wynikające z publikowania/subskrybowania z *sterowaniem* kolejkowaniem i korzystaniem z publikacji. Niezarządzana subskrypcja jest miejscem, w którym aplikacja jest odpowiedzialna. w celu określenia kolejki, w której przechowywane są subskrypcje. W przykładzie przedstawiono różne sposoby łączenia subskrypcji i kolejek.

“Pisanie aplikacji publikatora” na stronie 835
Zacznij pisać aplikacje wydawcy, analizując dwa przykłady. Pierwszy z nich jest modelowany tak blisko, jak to możliwe, w aplikacji umieszczającej komunikaty w kolejce, a drugi demonstruje dynamiczne tworzenie tematów-bardziej powszechny wzorzec dla aplikacji publikujących.

*Przykład 3: Niezarządzany subskrybent produktu MQ*
Niezarządzany subskrybent jest ważną klasą aplikacji subskrybenta. Dzięki temu można połączyć korzyści wynikające z publikowania/subskrybowania z *sterowaniem* kolejkowaniem i korzystaniem z publikacji. Niezarządzana subskrypcja jest miejscem, w którym aplikacja jest odpowiedzialna. w celu określenia kolejki, w której przechowywane są subskrypcje. W przykładzie przedstawiono różne sposoby łączenia subskrypcji i kolejek.

Wzorzec niezarządzany jest częściej powiązany z subskrypcjami *trwałymi* niż z subskrypcjami *nietrwałymi*. Zwykle cykl życia subskrypcji utworzonej przez niezarządzanego subskrybenta jest niezależny od cyklu

życia samej aplikacji subskrybującej. Ustawienie subskrypcji jako trwałej powoduje, że subskrypcja odbiera publikacje nawet wtedy, gdy żadna aplikacja subskrybująca nie jest aktywna.

W celu osiągnięcia tego samego wyniku można utworzyć trwałe *zarządzane* subskrypcje, ale niektóre aplikacje wymagają większej elastyczności i kontroli nad kolejkami i komunikatami, niż jest to możliwe w przypadku subskrypcji zarządzanej. W przypadku trwałej subskrypcji zarządzanej menedżer kolejek tworzy trwałą kolejkę dla publikacji zgodnych z tematem subskrypcji. Usuwa kolejkę i powiązane publikacje, gdy subskrypcja jest usuwana.

Zwykle trwałe *zarządzane* subskrypcje są używane, jeśli cykl życia aplikacji i subskrypcji jest zasadniczo taki sam, ale trudny do zagwarantowania. Ponieważ subskrypcja jest trwała i publikator tworzy trwałe publikacje, nie ma utraconych komunikatów, jeśli menedżer kolejek lub subskrybent zostaną przedwcześnie zakończone i będą musiały zostać odzyskane.

W przypadku aplikacji innych niż JMS lub aplikacji JMS, które nie korzystają z subskrypcji współużytkowanej, menedżer kolejek niejawnie otworzy trwałą zarządzaną kolejkę subskrypcji dla subskrybenta w taki sposób, aby nie było możliwe współużytkowane przetwarzanie kolejki. Ponadto, jeśli aplikacja nie używa współużytkowanych subskrypcji JMS, nie jest możliwe utworzenie więcej niż jednej subskrypcji dla każdej zarządzanej kolejki i może okazać się, że zarządzanie kolejkami jest trudniejsze, ponieważ użytkownik ma mniejszą kontrolę nad nazwami kolejek. Z tych powodów należy rozważyć, czy subskrybent produktu *niezarządzany* MQ jest lepiej dopasowany do aplikacji wymagających trwałych subskrypcji niż subskrybent produktu *zarządzane* MQ .

Kod w pliku Rysunek 76 na stronie 856 przedstawia wzorzec niezarządzanej trwałej subskrypcji. Na potrzeby ilustracji kod tworzy również niezarządzane, nietrwałe subskrypcje. W tym przykładzie przedstawiono następujące aspekty wzorca:

- Subskrypcje na żądanie: łańcuchy tematu subskrypcji są dynamiczne. Są one udostępniane przez aplikację podczas jej działania.
- Uproszczone zarządzanie tematami subskrypcji: Zarządzanie tematami subskrypcji jest uproszczone przez zdefiniowanie głównej części łańcucha tematu subskrypcji przy użyciu tematu zdefiniowanego administracyjnie. Spowoduje to ukrycie głównej części drzewa tematów w aplikacji. Ukrywając część główną, subskrybent może zostać wdrożony w różnych drzewach tematów.
- Elastyczne zarządzanie subskrypcjami: subskrypcję można zdefiniować administracyjnie lub utworzyć na żądanie w programie subskrybenta. Nie ma różnicy między subskrypcjami utworzonymi administracyjnie i programowo, z wyjątkiem atrybutu, który pokazuje, w jaki sposób subskrypcja została utworzona. Istnieje trzeci typ subskrypcji, która jest tworzona automatycznie przez menedżer kolejek na potrzeby dystrybucji subskrypcji. Wszystkie subskrypcje są wyświetlane w Eksploratorze IBM MQ .
- Elastyczne powiązanie subskrypcji z kolejkami: predefiniowana kolejka lokalna jest powiązana z subskrypcją przez funkcję MQSUB . Istnieją różne sposoby użycia usługi MQSUB do powiązania subskrypcji z kolejkami:
  - Powiąż subskrypcję z kolejką, która *nie* ma istniejących subskrypcji `MQSO_CREATE + (Hobj from MQOPEN)`.
  - Powiąż *nową* subskrypcję z kolejką, w której istnieją subskrypcje, `MQSO_CREATE + (Hobj from MQOPEN)`.
  - Przenieś istniejącą subskrypcję do innej kolejki, `MQSO_ALTER + (Hobj from MQOPEN)`.
  - Wznawianie istniejącej subskrypcji powiązanej z istniejącą kolejką `MQSO_RESUME + (Hobj = MQHO_NONE)` lub `MQSO_RESUME + (Hobj = from MQOPEN of queue with existing subscription)`.
  - Łącząc łańcuch `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` w różnych kombinacjach, można zaspokoić różne stany wejściowe subskrypcji i kolejki bez konieczności kodowania wielu wersji pliku MQSUB z różnymi wartościami `sd.Options` .
  - Alternatywnie, kodując konkretny wybór spośród `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` , menedżer kolejek zwraca błąd ( Tabela 123 na stronie 853 ). jeśli stany subskrypcji i kolejki podane jako dane wejściowe dla MQSUB są niespójne z wartością `sd.Options`. Rysunek 82 na stronie 859 przedstawia wyniki wprowadzenia komendy MQSUB dla subskrypcji X z różnymi indywidualnymi ustawieniami flagi `sd.Options` i przekazaniem jej trzech różnych uchwytów obiektów.

Zapoznaj się z różnymi danymi wejściowymi do przykładowego programu w programie Rysunek 75 na stronie 855 , aby zapoznać się z różnymi rodzajami błędów. Jednym z najczęstszych błędów, RC = 2440, który nie jest uwzględniany w przypadkach wymienionych w tabeli, jest błąd nazwy subskrypcji. Zwykle jest to spowodowane przekazaniem pustej lub niepoprawnej nazwy subskrypcji w programie MQSO_RESUME lub MQSO_ALTER.

- Multiprocessing: Możesz dzielić się z wieloma konsumentami pracą czytania publikacji. Wszystkie publikacje są umieszczane w pojedynczej kolejce powiązanej z tematem subskrypcji. Konsumenci mogą otwierać kolejkę bezpośrednio za pomocą programu MQOPEN lub wznawiać subskrypcję za pomocą programu MQSUB.
- Koncentracja subskrypcji: w tej samej kolejce można utworzyć wiele subskrypcji. Należy zachować ostrożność w przypadku tej możliwości, ponieważ może ona powodować nakładanie się subskrypcji i wielokrotne odbieranie tej samej publikacji. Opcja MQSO_GROUP_SUB eliminuje duplikaty publikacji spowodowane nakładającymi się subskrypcjami.
- Separacja między subskrybentami i konsumentami: Podobnie jak trzy modele konsumentów przedstawione w przykładach, innym modelem jest oddzielenie konsumenta od subskrybenta. Jest to wariant niezarządzanego subskrybenta produktu MQ , ale zamiast wydawać komendy MQOPEN i MQSUB w tym samym programie, jeden program subskrybuje publikacje, a inny program je wykorzystuje. Na przykład subskrybent może być częścią klastra publikowania/subskrybowania, a konsument może być przyłączony do menedżera kolejek poza klastrem menedżerów kolejek. Konsument odbiera publikacje za pośrednictwem standardowego rozproszonego kolejkowania przez zdefiniowanie kolejki subskrypcji jako definicji kolejki zdalnej.

Zrozumienie działania produktu MQSO_CREATE | MQSO_RESUME | MQSO_ALTER jest ważne, zwłaszcza jeśli planowane jest uproszczenie kodu za pomocą kombinacji tych opcji. Należy przestudiować tabelę Tabela 123 na stronie 853 , która przedstawia wyniki przekazania różnych uchwytów kolejek do MQSUB oraz wyniki uruchomienia przykładowego programu przedstawionego w sekcji Rysunek 77 na stronie 857 do Rysunek 82 na stronie 859.

Scenariusz używany do tworzenia tabeli ma jedną subskrypcję X i dwie kolejki, A i B. Parametr nazwy subskrypcji sd.SubName jest ustawiony na wartość X, czyli nazwę subskrypcji przyłączonej do kolejki A. Do kolejki B nie jest przyłączona żadna subskrypcja.

W programie Tabela 123 na stronie 853 MQSUB jest przekazywana subskrypcja X i uchwyt kolejki do kolejki A. Wyniki z opcji subskrypcji są następujące:

- Wykonanie komendy MQSO_CREATE nie powiodło się, ponieważ uchwyt kolejki odpowiada kolejce A , która ma już subskrypcję programu X. Porównaj to zachowanie z pomyślnym wywołaniem. Wywołanie powiodło się, ponieważ do kolejki B nie jest przyłączona subskrypcja usługi X .
- Wykonanie komendy MQSO_RESUME powiodło się, ponieważ uchwyt kolejki odpowiada kolejce A , która ma już subskrypcję programu X. Natomiast wywołanie kończy się niepowodzeniem, gdy subskrypcja X nie istnieje w kolejce A.
- Działanie programu MQSO_ALTER jest podobne do działania programu MQSO_RESUME w odniesieniu do otwierania subskrypcji i kolejki. Jeśli jednak atrybuty zawarte w deskryptorze subskrypcji przekazanym do metody MQSUB różnią się od atrybutów subskrypcji, operacja MQSO_RESUME kończy się niepowodzeniem, a operacja MQSO_ALTER kończy się niepowodzeniem, o ile instancja programu ma uprawnienia do modyfikowania atrybutów. Należy zauważyć, że nigdy nie można zmienić łańcucha tematu w subskrypcji, ale zamiast zwracać błąd, produkt MQSUB ignoruje nazwę tematu i wartości łańcucha tematu w deskryptorze subskrypcji i używa wartości z istniejącej subskrypcji.

Następnie należy spojrzeć na plik Tabela 123 na stronie 853 , w którym do usługi MQSUB przekazano subskrypcję X i uchwyt kolejki do kolejki B. Wyniki z opcji subskrypcji są następujące:

- Operacja MQSO_CREATE zakończyła się powodzeniem i tworzy subskrypcję X w kolejce B , ponieważ jest to nowa subskrypcja w kolejce B.
- Niepowodzenie MQSO_RESUME . MQSUB wyszukuje subskrypcję X w kolejce B i nie znajduje jej, ale zamiast zwracać wartość *RC = 2428-subskrypcja X nie istnieje*zwraca wartość *RC = 2019-Kolejka subskrypcji nie jest zgodna z uchwytem obiektu kolejki*. Zachowanie trzeciej opcji MQSO_ALTER sugeruje przyczynę tego nieoczekiwanego błędu. Program MQSUB oczekuje, że uchwyt kolejki będzie wskazywał

kolejkę z subskrypcją. Sprawdza to najpierw przed sprawdzeniem, czy istnieje subskrypcja o nazwie podanej w pliku `sd.SubName` .

- Operacja `MQSO_ALTER` powiedzie się i spowoduje przeniesienie subskrypcji z kolejki A do kolejki B.

Przypadek, który nie jest wyświetlany w tabeli, występuje wtedy, gdy nazwa subskrypcji w kolejce A nie jest zgodna z nazwą subskrypcji w programie `sd.SubName`. Wywołanie to kończy się niepowodzeniem z kodem powrotu = *2428-subskrypcja X nie istnieje w kolejce A*.

*Tabela 123. Błędy z MQSUB z różnymi uchwytami kolejek i kombinacjami subskrypcji*

| **Uchwyty kolejek** | **Kolejka A**<br>**Subskrypcja X**<br>**Kolejka B**<br>**Brak subskrypcji** | **Kolejka A**<br>**Brak subskrypcji**<br>**Kolejka B**<br>**Brak subskrypcji** |
|---|---|---|
| **Hobj dla *kolejki A* przekazanej do MQSUB** | **MQSO_CREATE**<br>RC = 2432-Subskrypcja X już istnieje w kolejce A<br>**MQSO_RESUME**<br>Wznawia subskrypcję X w kolejce A<br>**MQSO_ALTER**<br>Wznawia subskrypcję X w kolejce A i wprowadza dozwolone zmiany | **MQSO_CREATE**<br>Tworzy subskrypcję X w kolejce A<br>**MQSO_RESUME**<br>RC = 2428-Subskrypcja X nie istnieje w kolejce A<br>**MQSO_ALTER**<br>RC = 2428-Subskrypcja X nie istnieje w kolejce A |
| **Hobj dla *kolejki B* przekazanej do MQSUB** | **MQSO_CREATE**<br>Tworzy nową subskrypcję X w kolejce B<br>**MQSO_RESUME**<br>RC = 2019-Kolejka subskrypcji nie jest zgodna z uchwytem obiektu kolejki<br>**MQSO_ALTER**<br>Przenieś subskrypcję X z kolejki A do kolejki B | **MQSO_CREATE**<br>Tworzy nową subskrypcję X w kolejce B<br>**MQSO_RESUME**<br>RC = 2428-subskrypcja X nie istnieje w kolejce B<br>**MQSO_ALTER**<br>RC = 2428-subskrypcja X nie istnieje w kolejce B |
| **MQHO_NONE przekazano do MQSUB** | **MQSO_CREATE**<br>RC = 2019-Błędny uchwyt obiektu: ustaw opcję `MQSO_MANAGED` , aby utworzyć subskrypcję zarządzaną i utworzyć kolejkę zarządzaną<br>**MQSO_RESUME**<br>Wznawia subskrypcję X w kolejce A i zwraca Hobj do kolejki A<br>**MQSO_ALTER**<br>Wznawia subskrypcję X w kolejce A, zwraca Hobj do kolejki A i wprowadza dozwolone zmiany | **MQSO_CREATE**<br>RC = 2019-Błędny uchwyt obiektu: ustaw opcję `MQSO_MANAGED` , aby utworzyć subskrypcję zarządzaną i utworzyć kolejkę zarządzaną<br>**MQSO_RESUME**<br>RC = 2428-Brak subskrypcji X<br>**MQSO_ALTER**<br>RC = 2019-Błędny uchwyt obiektu: brak kolejki A lub B |

**Uwaga:** Zwarty styl kodowania jest przeznaczony do czytelności, a nie do użytku produkcyjnego.

```
#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));
```

*Rysunek 74. Niezarządzany subskrybent produktu MQ -część 1: deklaracje.*

```
    switch(argc){                   /* Replace defaults with args if provided */
default:
        switch((argv[5][0])) {
case('A'):  sdOptions =  MQSO_ALTER  | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('C'):  sdOptions =  MQSO_CREATE | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('R'):  sdOptions =  MQSO_RESUME | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
default:    ;
        }
case(5):
    if (strcmp(argv[4],"/")) /* "/" invalid = No subscription  */
        subscriptionQueue = argv[4];
    else {
        *subscriptionQueue = '\0';
        if (argc > 5) {
            if (argv[5][0] == 'C') {
                sdOptions = sdOptions + MQSO_MANAGED;
            }
        }
        else
            sdOptions = sdOptions + MQSO_MANAGED;
    }

case(4):
    if (strcmp(argv[3],"/")) /* "/" invalid = No subscription  */
        subscriptionName = argv[3];
    else {
        *subscriptionName = '\0';
        sdOptions = sdOptions - MQSO_DURABLE;
    }
case(3):
    if (strcmp(argv[2],"/")) /* "/" invalid = No topic string  */
        topicString = argv[2];
    else
        *topicString = '\0';
case(2):
    if (strcmp(argv[1],"/")) /* "/" invalid = No topic object  */
        topicName = argv[1];
    else
        *topicName = '\0';
case(1):
    sd.Options = sdOptions;
    printf("Optional parameters: "
    printf("topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|
R(esume)\n");
    printf("Values \"%-.48s\"  \"%s\" \"%s\"  \"%-.48s\" sd.Options=%d\n",
           topicName, topicString, subscriptionName, subscriptionQueue, sd.Options);
}
```

*Rysunek 75. Niezarządzany subskrybent produktu MQ -część 2: obsługa parametrów.*

Dodatkowe komentarze dotyczące obsługi parametrów w tym przykładzie są następujące:

**switch((argv[5][0]))**
Można wprowadzić wartość A lter | C reate | R esume w parametrze 5, aby przetestować efekt nadpisania części ustawienia opcji MQSUB użytej domyślnie w przykładzie. Ustawienie domyślne używane w tym przykładzie to MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE.

**Uwaga:** Ustawienie parametru MQSO_ALTER lub MQSO_RESUME bez ustawienia MQSO_DURABLE jest błędem, a parametr sd.SubName musi zostać ustawiony i odwoływać się do subskrypcji, która może zostać wznowiona lub zmieniona.

***subscriptionQueue = '\0';**
**sdOptions = sdOptions + MQSO_MANAGED;**
Jeśli domyślna kolejka subskrypcji, STOCKTICKER jest zastępowana przez łańcuch pusty, to dopóki ustawiona jest wartość MQSO_CREATE , w przykładzie ustawiana jest flaga MQSO_MANAGED i tworzona jest dynamiczna kolejka subskrypcji. Jeśli w piątym parametrze zostanie ustawiona wartość Alter or Resume , zachowanie przykładu będzie zależało od wartości parametru subscriptionName.

***subscriptionName = '\0';**
**sdOptions = sdOptions - MQSO_DURABLE;**

Jeśli domyślna subskrypcja, IBMSTOCKPRICESUB, zostanie zastąpiona łańcuchem o wartości NULL, w przykładzie zostanie usunięta opcja MQSO_DURABLE . W przypadku uruchomienia przykładu z wartościami domyślnymi dla innych parametrów tworzona jest dodatkowa tymczasowa subskrypcja przeznaczona dla produktu STOCKTICKER , która otrzymuje zduplikowane publikacje. Przy następnym uruchomieniu przykładu, bez żadnych parametrów, ponownie pojawi się tylko jedna publikacja.

```
    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;
}
```

*Rysunek 76. Niezarządzany subskrybent produktu MQ -część 3: treść kodu.*

Dodatkowe komentarze do kodu w tym przykładzie są następujące:

**if (strlen(subscriptionQueue))**
Jeśli nie ma nazwy kolejki subskrypcji, w przykładzie użyto wartości MQHO_NONE jako wartości parametru Hobj.

**MQOPEN(...);**
Kolejka subskrypcji zostanie otwarta, a uchwyt kolejki zostanie zapisany w programie Hobj.

**MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);**
Subskrypcja jest otwierana przy użyciu Hobj przekazanego z MQOPEN (lub MQHO_NONE , jeśli nie ma nazwy kolejki subskrypcji). Niezarządzaną kolejkę można wznowić bez jawnego otwierania jej za pomocą komendy MQOPEN.

**MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);**
Subskrypcja jest zamykana przy użyciu uchwytu subskrypcji. W zależności od tego, czy subskrypcja jest trwała, czy nie, subskrypcja jest zamykana z niejawnym MQCO_KEEP_SUB lub MQCO_REMOVE_SUB. Można zamknąć trwałą subskrypcję za pomocą programu MQCO_REMOVE_SUB, ale nie można zamknąć nietrwałej subskrypcji za pomocą programu MQCO_KEEP_SUB. Działanie programu MQCO_REMOVE_SUB polega na usunięciu subskrypcji, co powoduje zatrzymanie wysyłania kolejnych publikacji do kolejki subskrypcji.

**MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);**
Jeśli subskrypcja nie jest zarządzana, nie jest wykonywane żadne działanie specjalne. Jeśli kolejka jest zarządzana, a subskrypcja została zamknięta za pomocą jawnego lub niejawnego MQCO_REMOVE_SUB, wszystkie publikacje są usuwane z kolejki i kolejki w tym momencie.

**gmo.MatchOptions = MQMO_MATCH_CORREL_ID;**
**memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);**
Upewnij się, że odebrane komunikaty są przeznaczone dla naszej subskrypcji.

Wyniki z przykładu ilustrują aspekty publikowania/subskrybowania:

W Rysunek 77 na stronie 857 przykład rozpoczyna się od opublikowania pliku 130 w temacie NYSE/IBM/PRICE .

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

*Rysunek 77. Opublikuj 130 do NYSE/IBM/PRICE*

Podczas Rysunek 78 na stronie 857 wykonywania przykładu przy użyciu parametrów domyślnych otrzymywana jest zachowana publikacja 130. Podany obiekt tematu i łańcuch tematu są ignorowane, co przedstawia Rysunek 82 na stronie 859. Obiekt tematu i łańcuch tematu są zawsze pobierane z obiektu subskrypcji (jeśli został podany), a łańcuch tematu jest niezmienialny. Rzeczywiste zachowanie przykładu zależy od wyboru lub kombinacji parametrów MQSO_CREATE, MQSO_RESUMEi MQSO_ALTER. W tym przykładzie wybrana jest opcja MQSO_RESUME .

```
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
```

*Rysunek 78. Odbierz zachowaną publikację*

W ( Rysunek 79 na stronie 858 ) Nie odebrano żadnych publikacji, ponieważ trwała subskrypcja już otrzymała zachowaną publikację. W tym przykładzie subskrypcja jest wznawiana przez podanie tylko nazwy subskrypcji bez nazwy kolejki. Jeśli podano nazwę kolejki, zostanie ona najpierw otwarta, a uchwyt zostanie przekazany do programu MQSUB.

**Uwaga:** Błąd 2038 z MQINQ jest spowodowany niejawnym MQOPEN STOCKTICKER przez MQSUB bez opcji MQOO_INQUIRE . Unikaj kodu powrotu 2038 z programu MQINQ , jawnie otwierając kolejkę.

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE IBMSTOCKPRICESUB / Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "" sd.Options=8204
MQINQ failed with Condition code 2 and Reason 2038
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from unknown queue
Completion code 0 and Return code 0
```

*Rysunek 79. Wznów subskrypcję*

W programie Rysunek 80 na stronie 858przykład tworzy nietrwałą, niezarządzaną subskrypcję, używając STOCKTICKER jako miejsca docelowego. Ponieważ jest to nowa subskrypcja, otrzymuje ona zachowaną publikację.

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

*Rysunek 80. Odbierz zachowaną publikację z nową niezarządzaną subskrypcją nietrwałą*

W programie Rysunek 81 na stronie 858, aby zademonstrować nakładające się subskrypcje, wysyłana jest inna publikacja, która zmienia zachowaną publikację. Następnie tworzona jest nowa nietrwała, niezarządzana subskrypcja bez podawania nazwy subskrypcji. Zachowana publikacja jest odbierana dwukrotnie, raz dla nowej subskrypcji i raz dla trwałej subskrypcji IBMSTOCKPRICESUB , która jest nadal aktywna w kolejce STOCKTICKER . Przykład jest ilustracją tego, że jest to kolejka, która ma subskrypcje, a nie aplikacja. Mimo że w tym wywołaniu aplikacji nie ma odwołania do subskrypcji IBMSTOCKPRICESUB , aplikacja otrzymuje publikację dwukrotnie: raz z subskrypcji trwałej, która została utworzona administracyjnie, i raz z subskrypcji nietrwałej, która została utworzona przez samą aplikację.

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Received publication "130"
Completion code 0 and Return code 0
```

*Rysunek 81. Nakładające się subskrypcje*

W przykładzie Rysunek 82 na stronie 859 zademonstrowano, że udostępnienie nowego łańcucha tematu i istniejącej subskrypcji nie powoduje zmiany subskrypcji.

1. W pierwszym przypadku program Resume wznawia istniejącą subskrypcję zgodnie z oczekiwaniami i ignoruje zmieniony łańcuch tematu.
2. W drugim przypadku Alter powoduje błąd RC = 2510, Topic not alterable.
3. W trzecim przykładzie Create powoduje błąd RC = 2432, Sub already exists.

```
W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8204
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Alter
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8201
Completion code 2 and Return code 2510

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8202
Completion code 2 and Return code 2432
```

*Rysunek 82. Nie można zmieniać tematów subskrypcji*

**Pojęcia pokrewne**

“Przykład 1: konsument publikacji produktu MQ” na stronie 843
Konsument publikacji MQ jest konsumentem komunikatów IBM MQ , który nie subskrybuje samych tematów.

“Przykład 2: Zarządzany subskrybent produktu MQ” na stronie 845
Zarządzany subskrybent produktu MQ jest preferowanym wzorcem dla większości aplikacji subskrybenta. Subskrypcja zarządzana to taka, w której program IBM MQ obsługuje subskrypcję i rejestruje ją oraz wyrejestrowuje. W tym przykładzie nie jest wymagana *żadna* definicja administracyjna kolejek, tematów ani subskrypcji.

“Pisanie aplikacji publikatora” na stronie 835
Zacznij pisać aplikacje wydawcy, analizując dwa przykłady. Pierwszy z nich jest modelowany tak blisko, jak to możliwe, w aplikacji umieszczającej komunikaty w kolejce, a drugi demonstruje dynamiczne tworzenie tematów-bardziej powszechny wzorzec dla aplikacji publikujących.

## *Cykle życia publikowania/subskrypcji*

Podczas projektowania aplikacji publikowania/subskrybowania należy rozważyć cykle życia tematów, subskrypcji, subskrybentów, publikacji, publikatorów i kolejek.

Cykl życia obiektu, takiego jak subskrypcja, rozpoczyna się od jego utworzenia i kończy się jego usunięciem. Może on również obejmować inne stany i zmiany, przez które przechodzi, takie jak tymczasowe zawieszenie, tematy nadrzędne i podrzędne, utrata ważności i usuwanie.

Tradycyjnie obiekty IBM MQ , takie jak kolejki, są tworzone administracyjnie lub przez programy administracyjne przy użyciu formatu komend programowalnych (Programmable Command Format-PCF). Publikowanie/subskrypcja jest inna w udostępnianiu komend interfejsu API MQSUB i MQCLOSE do tworzenia i usuwania subskrypcji, z pojęciem subskrypcji zarządzanych, które nie tylko tworzą i usuwają kolejki, ale także czyszczą niewykorzystane komunikaty oraz mają powiązania między administracyjnie utworzonymi obiektami tematów a programowo lub administracyjnie utworzonymi łańcuchami tematów.

To bogactwo funkcjonalne spełnia szeroki zakres wymagań publikowania/subskrybowania, a także upraszcza projektowanie niektórych wspólnych wzorców aplikacji publikowania/subskrybowania. Subskrypcje zarządzane na przykład upraszczają zarówno programowanie, jak i administrowanie subskrypcją, która ma trwać tylko tak długo, jak program, który ją utworzył. Subskrypcje niezarządzane upraszczają programowanie, gdy istnieje luźniejsze połączenie między subskrybowaniem i korzystaniem z publikacji. Subskrypcje tworzone centralnie są przydatne, gdy wzorzec dotyczy kierowania ruchu publikacji do konsumentów w oparciu o scentralizowany model kontroli, na przykład wysyłanie informacji o locie do zautomatyzowanych bram, podczas gdy subskrypcje tworzone programowo mogą być używane, jeśli personel bramy jest odpowiedzialny za subskrybowanie rekordów pasażerów dla tego lotu, przez wprowadzenie numeru lotu przy bramie.

W tym ostatnim przykładzie zarządzana subskrypcja trwała może być odpowiednia: zarządzana, ponieważ subskrypcje są tworzone bardzo często i mają wyraźny punkt końcowy, gdy brama jest zamykana, a subskrypcja może zostać programowo usunięta; trwała, aby uniknąć utraty rekordu pasażera z powodu wyłączenia programu subskrybenta bramy z jednego lub innego powodu.[8]. Aby rozpocząć publikację danych pasażerów do bramy, możliwe byłoby, aby aplikacja bramy zasubskrybować zapisy pasażerów

[8] Publikator musi wysłać rekordy pasażerów jako trwałe komunikaty, aby uniknąć innych możliwych awarii, oczywiście.

przy użyciu numeru bramy, i opublikować zdarzenie otwarcia bramy przy użyciu numeru bramy. Wydawca odpowiada na zdarzenie otwarcia bramy, publikując rejestry pasażerów-które mogą również udać się do innych zainteresowanych stron, takich jak fakturowanie, aby zarejestrować lot odbywa się, oraz do obsługi klienta, aby otrzymywać powiadomienia tekstowe do telefonów komórkowych pasażerów o numerze bramy.

Centralnie zarządzana subskrypcja może korzystać z trwałego modelu niezarządzanego, kierując listy pasażerów do bramki przy użyciu predefiniowanej kolejki dla każdej bramki.

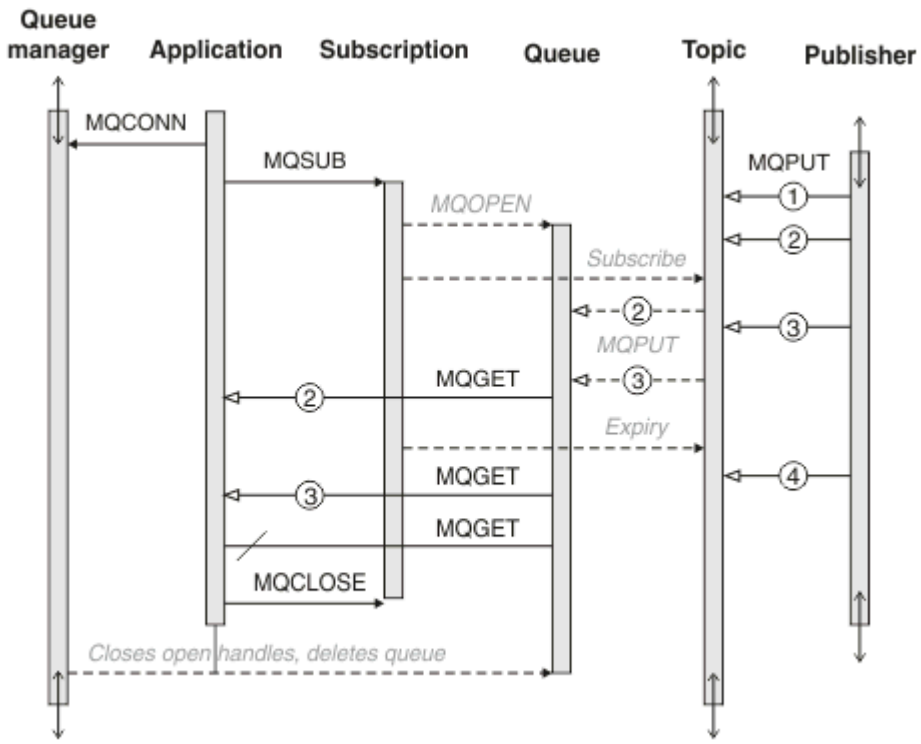
Poniższe trzy przykłady cykli życia publikowania/subskrypcji ilustrują, w jaki sposób zarządzane nietrwałe, zarządzane trwałe i niezarządzane trwałe subskrybenty współdziałają z subskrypcjami, tematami, kolejkami, publikatorami i menedżerem kolejek oraz w jaki sposób obowiązki mogą być podzielone między programy administracyjne i subskrybentów.

**Zarządzany nietrły subskrybent**

Rysunek 83 na stronie 861 przedstawia aplikację, która tworzy zarządzaną nietrwałą subskrypcję, pobierając dwa komunikaty opublikowane w temacie zidentyfikowanym w subskrypcji i kończąc działanie. Interakcje oznaczone kursywą w kolorze szarym ze strzałkami z kropkami są niejawne.

Należy zwrócić uwagę na kilka kwestii.

1. Aplikacja tworzy subskrypcję tematu, który został już dwukrotnie opublikowany. Gdy subskrybent otrzyma pierwszą publikację, otrzyma *drugą* publikację, która jest obecnie zachowaną publikacją.
2. Menedżer kolejek tworzy tymczasową kolejkę subskrypcji, a także tworzy subskrypcję dla tematu.
3. Subskrypcja traci ważność. Jeśli subskrypcja utraci ważność, do tej subskrypcji nie zostaną wysłane żadne publikacje dotyczące tego tematu, ale subskrybent będzie nadal otrzymywał komunikaty opublikowane przed utratą ważności subskrypcji. Utrata ważności subskrypcji nie ma wpływu na utratę ważności publikacji.
4. Czwarta publikacja nie jest umieszczana w kolejce subskrypcji i w związku z tym ostatnia publikacja `MQGET` nie zwraca publikacji.
5. Chociaż subskrybent zamyka swoją subskrypcję, nie zamyka połączenia z kolejką ani menedżerem kolejek.
6. Menedżer kolejek zostanie wyczyszczona wkrótce po zakończeniu działania aplikacji. Ponieważ subskrypcja jest zarządzana i nietrwała, kolejka subskrypcji jest usuwana.




*Rysunek 83. Zarządzane nietrwałe linie życia subskrybenta*

**Zarządzany trwały subskrybent**

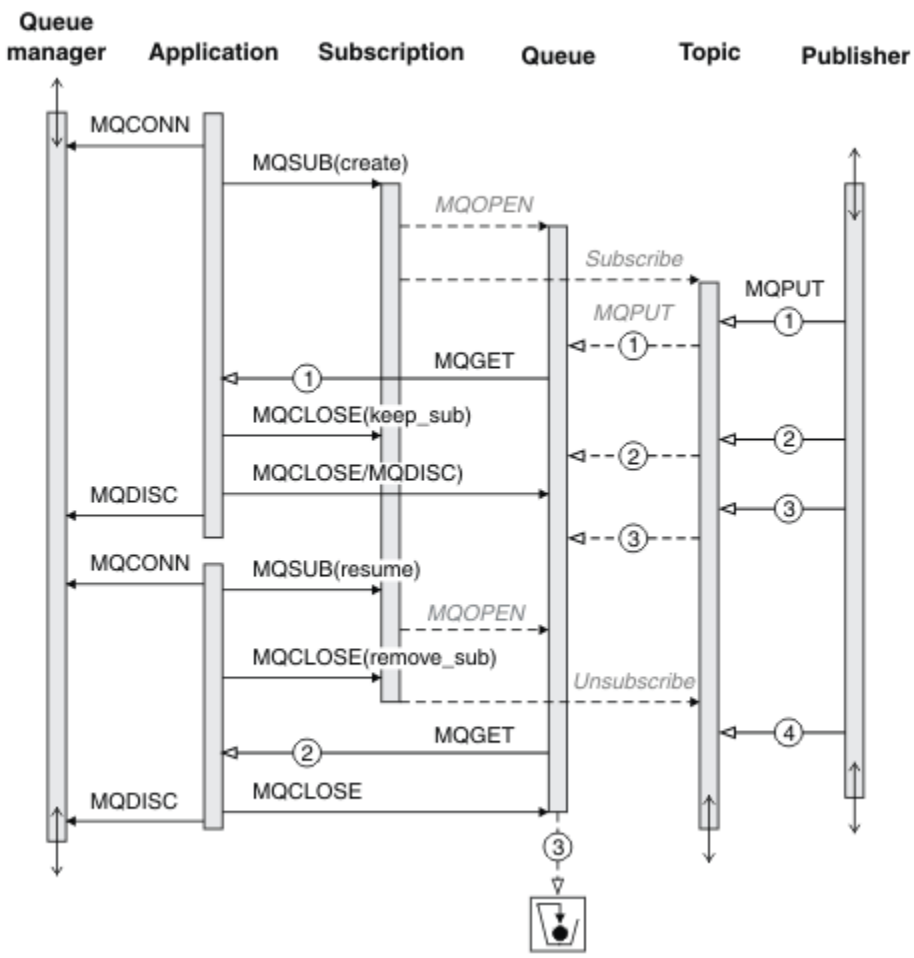
Zarządzany trwały subskrybent wykonuje poprzedni przykład krok dalej i wyświetla zarządzaną subskrypcję, która przetrwała zakończenie i restart aplikacji subskrybującej.

Należy zwrócić uwagę na kilka nowych kwestii.

1. W tym przykładzie, w przeciwieństwie do ostatniego, temat publikacji nie istniał przed zdefiniowaniem go w subskrypcji.
2. Gdy subskrybent po raz pierwszy kończy działanie, zamyka subskrypcję z opcją `MQCO_KEEP_SUB`. Jest to domyślne zachowanie w przypadku niejawnego zamykania zarządzanej trwałej subskrypcji.
3. Gdy subskrybent wznowi subskrypcję, kolejka subskrypcji zostanie ponownie otwarta.
4. Nowa publikacja 2umieszczona w kolejce przed ponownym otwarciem jest dostępna dla produktu `MQGET`, nawet po usunięciu subskrypcji.

   Mimo że subskrypcja jest trwała, subskrybent niezawodnie odbiera wszystkie komunikaty wysyłane przez publikator tylko wtedy, gdy *obie* subskrypcja jest trwała, a komunikaty trwałe. Trwałość komunikatu zależy od ustawienia pola `Persistent` w pliku `MQMD` komunikatu wysłanego przez publikator. Subskrybent nie ma nad tym kontroli.
5. Zamknięcie subskrypcji z flagą `MQCO_REMOVE_SUB` powoduje usunięcie subskrypcji, co powoduje zatrzymanie dalszych publikacji umieszczanych w kolejce subskrypcji. Po zamknięciu kolejki subskrypcji menedżer kolejek usuwa nieprzeczytaną publikację 3, a następnie usuwa kolejkę. Działanie jest równoznaczne z administracyjnym usunięciem subskrypcji.

   **Uwaga:** Nie usuwaj kolejki ręcznie ani nie wydawaj komendy `MQCLOSE` z opcją `MQCO_DELETE`lub `MQCO_PURGE_DELETE`. Widoczne szczegóły implementacji subskrypcji zarządzanej nie są częścią obsługiwanego interfejsu IBM MQ . Menedżer kolejek nie może zarządzać subskrypcją w sposób niezawodny, chyba że ma pełną kontrolę.




*Rysunek 84. Linie życia zarządzanych trwałych subskrybentów*

**Niezarządzany trwały subskrybent**

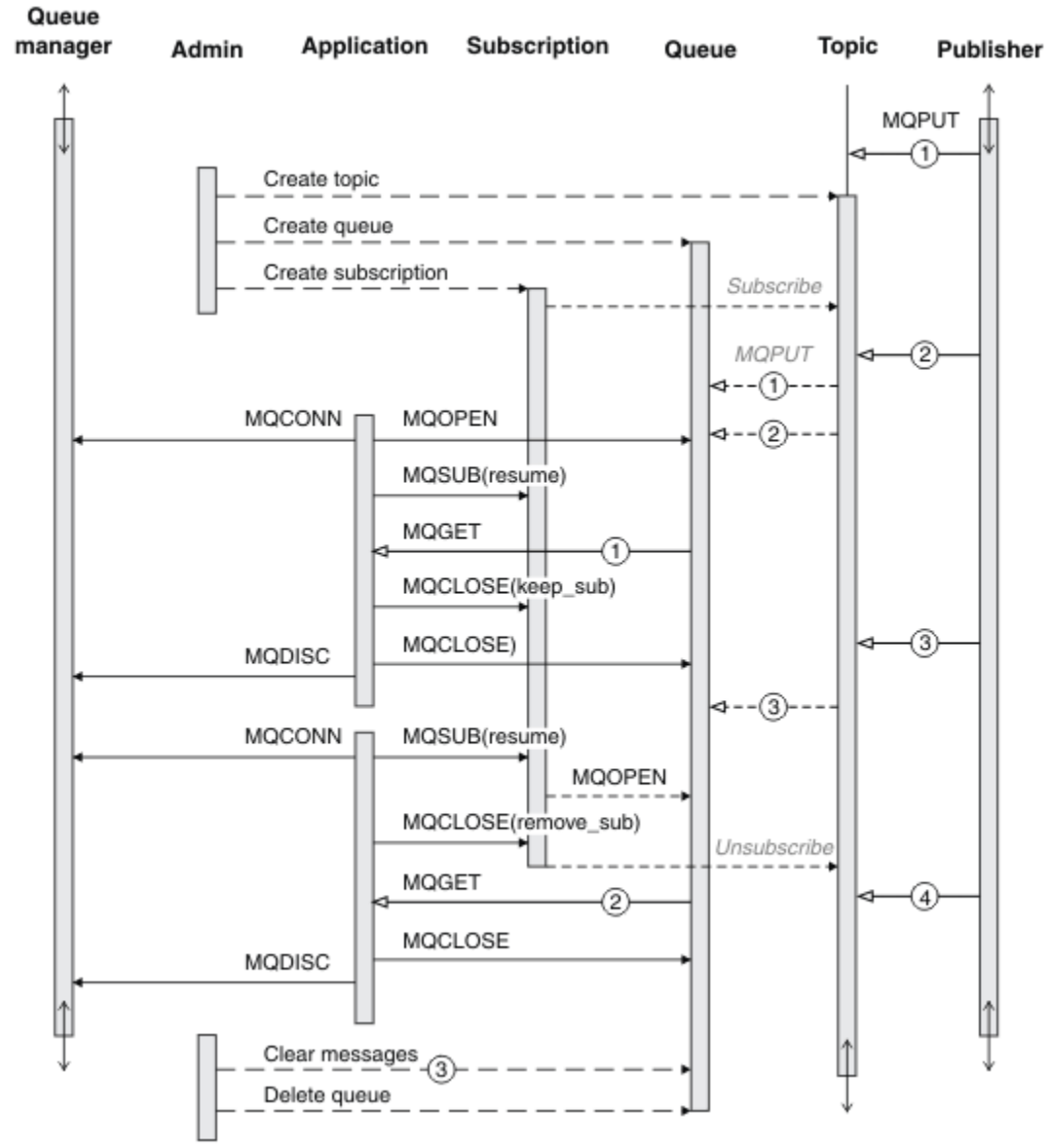
W trzecim przykładzie dodano administratora: niezarządzany trwały subskrybent. Dobrym przykładem jest sposób, w jaki administrator może współpracować z aplikacją publikowania/subskrybowania.

Wyświetlane są punkty, które należy zanotować.

1. Publikator umieszcza komunikat 1w temacie, który zostanie później powiązany z obiektem tematu używanym na potrzeby subskrypcji. Obiekt tematu definiuje łańcuch tematu, który jest zgodny z tematem opublikowanym przy użyciu znaków wieloznacznych.
2. Temat zawiera zachowaną publikację.
3. Administrator tworzy obiekt tematu, kolejkę i subskrypcję. Obiekt tematu i kolejkę należy zdefiniować przed subskrypcją.
4. Aplikacja otwiera kolejkę powiązaną z subskrypcją i przekazuje `MQSUB` uchwyt kolejki. Można też po prostu otworzyć subskrypcję, przekazując do niej uchwyt kolejki `MQHO_NONE`. Konwersacja nie jest prawdziwa, nie może wznowić subskrypcji, przekazując tylko uchwyt kolejki bez nazwy subskrypcji- kolejka może mieć wiele subskrypcji.
5. Aplikacja otwiera subskrypcję przy użyciu opcji `MQSO_RESUME` , nawet jeśli jest to pierwsza otwarta subskrypcja. Wznawia on subskrypcję utworzoną administracyjnie.
6. Subskrybent otrzymuje zachowaną publikację 1. Publikacja 2, mimo że została opublikowana przed odebraniem publikacji przez subskrybenta, została opublikowana po rozpoczęciu subskrypcji i jest drugą publikacją w kolejce subskrypcji.

**Uwaga:** Jeśli zachowana publikacja nie jest publikowana jako komunikat trwały, zostanie utracona po restarcie menedżera kolejek.

7. W tym przykładzie subskrypcja jest trwała. Program może utworzyć niezarządzaną, nietrwałą subskrypcję. To oczywiste, że administrator nie może tego zrobić.
8. Efektem opcji MQCO_REMOVE_SUB przy zamykaniu subskrypcji jest usunięcie subskrypcji tak, jakby została usunięta przez administratora. Powoduje to zatrzymanie wszystkich kolejnych publikacji wysyłanych do kolejki, ale nie ma wpływu na publikacje, które już znajdują się w kolejce, nawet jeśli kolejka jest zamknięta, w przeciwieństwie do trwałej subskrypcji *zarządzanej* .
9. Następnie administrator usuwa pozostały komunikat 3i usuwa kolejkę.




*Rysunek 85. Niezarządzane linie życia trwałego subskrybenta*

Normalnym wzorcem dla subskrypcji niezarządzanej jest konserwacja kolejek i subskrypcji, która ma być wykonywana przez administratora. Zwykle nie jest podejmowana próba emulowania zachowania zarządzanego subskrybenta i programowego porządkowania kolejek i subskrypcji w kodzie aplikacji. Jeśli zachodzi potrzeba napisania logiki zarządzania, należy zapytać, czy można osiągnąć te same wyniki przy użyciu wzorca zarządzanego. Nie jest łatwo napisać ściśle zsynchronizowany, całkowicie niezawodny kod zarządzania. Łatwiej jest później uporządkować komunikaty, subskrypcje i kolejki, ręcznie lub za pomocą

zautomatyzowanego programu do zarządzania, gdy można mieć pewność, że komunikaty, subskrypcje i kolejki mogą zostać po prostu usunięte, niezależnie od ich stanu.

### *Właściwości komunikatu publikowania/subskrypcji*

Kilka właściwości komunikatu odnosi się do przesyłania komunikatów typu publikowanie/subskrypcja produktu IBM MQ .

## Token PubAccounting

Jest to wartość, która będzie znajdować się w polu AccountingToken deskryptora komunikatu (MQMD) wszystkich komunikatów publikacji zgodnych z tą subskrypcją. AccountingToken jest częścią kontekstu tożsamości komunikatu. Więcej informacji na temat kontekstu komunikatu zawiera sekcja “Kontekst komunikatu” na stronie 48. Więcej informacji na temat pola AccountingToken w strukturze MQMD zawiera sekcja AccountingToken.

## PubApplIdentityData

Jest to wartość, która będzie znajdować się w polu danych ApplIdentitydeskryptora komunikatu (MQMD) wszystkich komunikatów publikacji zgodnych z tą subskrypcją. ApplIdentityDane są częścią kontekstu tożsamości komunikatu. Więcej informacji na temat kontekstu komunikatu zawiera sekcja “Kontekst komunikatu” na stronie 48. Więcej informacji na temat pola danych ApplIdentityw strukturze MQMD zawiera sekcja DaneApplIdentity.

Jeśli opcja MQSO_SET_IDENTITY_CONTEXT nie zostanie określona, dane ApplIdentity, które zostaną ustawione w każdym komunikacie publikowanym dla tej subskrypcji, będą puste jako domyślne informacje o kontekście.

Jeśli podano opcję MQSO_SET_IDENTITY_CONTEXT, użytkownik generuje dane PubApplIdentityData , a to pole jest polem wejściowym zawierającym dane ApplIdentity, które mają zostać ustawione w każdej publikacji dla tej subskrypcji.

## PubPriority

Jest to wartość, która będzie znajdować się w polu Priorytet deskryptora komunikatu (MQMD) wszystkich komunikatów publikacji zgodnych z tą subskrypcją. Więcej informacji na temat pola Priorytet w strukturze MQMD zawiera sekcja Priorytet.

Wartość musi być większa lub równa zero; zero jest najniższym priorytetem. Można również użyć następujących wartości specjalnych:

- MQPRI_PRIORITY_AS_Q_DEF-Jeśli kolejka subskrypcji jest podana w polu Hobj wywołania MQSUB i nie jest zarządzanym uchwytem, priorytet komunikatu jest pobierany z atrybutu DefProirity tej kolejki. Jeśli tak zidentyfikowana kolejka jest kolejką klastra lub istnieje więcej niż jedna definicja w ścieżce rozstrzygania nazw kolejek, priorytet jest określany, gdy komunikat publikacji jest umieszczany w kolejce zgodnie z opisem w sekcji Priorytet w deskryptorze MQMD. Jeśli wywołanie MQSUB używa zarządzanego uchwytu, priorytet komunikatu jest pobierany z atrybutu DefPriority kolejki modelowej powiązanej z subskrybowanym tematem.
- MQPRI_PRIORITY_AS_PUBLISHED-priorytetem komunikatu jest priorytet oryginalnej publikacji. Jest to wartość początkowa tego pola.

## SubCorrelIdentyfikator

**Ostrzeżenie:** Identyfikator korelacji może być przekazywany tylko między menedżerami kolejek w klastrze publikowania/subskrybowania, a nie w hierarchii.

Wszystkie publikacje wysłane w celu dopasowania do tej subskrypcji będą zawierać ten identyfikator korelacji w deskryptorze komunikatu. Jeśli wiele subskrypcji używa tej samej kolejki do pobierania swoich publikacji, użycie komendy MQGET według identyfikatora korelacji umożliwia uzyskanie tylko publikacji dla konkretnej subskrypcji. Ten identyfikator korelacji może zostać wygenerowany przez menedżera kolejek lub przez użytkownika.

Jeśli opcja MQSO_SET_CORREL_ID nie jest określona, identyfikator korelacji jest generowany przez menedżer kolejek, a to pole jest polem wyjściowym zawierającym identyfikator korelacji, który zostanie ustawiony w każdym komunikacie opublikowanym dla tej subskrypcji.

Jeśli podano opcję MQSO_SET_CORREL_ID, identyfikator korelacji jest generowany przez użytkownika, a to pole jest polem wejściowym zawierającym identyfikator korelacji, który ma zostać ustawiony w każdej publikacji dla tej subskrypcji. W takim przypadku, jeśli pole zawiera wartość MQCI_NONE, identyfikator korelacji, który zostanie ustawiony w każdym komunikacie opublikowanym dla tej subskrypcji, będzie identyfikatorem korelacji utworzonym przez oryginalne umieszczenie komunikatu.

Jeśli podano opcję MQSO_GROUP_SUB, a określony identyfikator korelacji jest taki sam, jak istniejąca zgrupowana subskrypcja używająca tej samej kolejki i nakładającego się łańcucha tematu, z kopią publikacji udostępniana jest tylko najbardziej znacząca subskrypcja w grupie.

## Dane SubUser

Są to dane użytkownika subskrypcji. Dane udostępnione w subskrypcji w tym polu zostaną dołączone jako właściwość komunikatu danych MQSubUserdla każdej publikacji wysyłanej do tej subskrypcji.

## Właściwości publikacji

Tabela 124 na stronie 865 zawiera listę właściwości publikacji, które są dostarczane z komunikatem publikacji.

Można uzyskać dostęp do tych właściwości bezpośrednio z folderu **MQRFH2** lub pobrać je za pomocą programu MQINQMP. Program MQINQMP akceptuje nazwę właściwości lub nazwę **MQRFH2** jako nazwę właściwości, do której ma zostać wykonane zapytanie.

*Tabela 124. Właściwości publikacji*

| Nazwa właściwości | Nazwa MQRFH2 | Typ | Opis |
|---|---|---|---|
| MQTopicString | mqps.Top | MQTYPE_STRING | Łańcuch tematu |
| MQSubUserData | mqps.Sud | MQTYPE_STRING | Dane użytkownika subskrybenta |
| MQIsRetained | mqps.Ret | MQTYPE_BOOLEAN | Zachowana publikacja |
| MQPubOptions | mqps.Pub | MQTYPE_INT32 | Opcje publikacji |
| MQPubLevel | mqps.Pbl | MQTYPE_INT32 | Poziom publikacji |
| MQPubTime | mqpse.Pts | MQTYPE_STRING | Czas publikacji |
| MQPubSeqNum | mqpse.Seq | MQTYPE_INT32 | Numer kolejny publikacji |
| MQPubStrIntData | mqpse.Sid | MQTYPE_STRING | Dane łańcuchowe/ całkowite dodane przez publikatora |
| MQPubFormat | mqpse.Pfmt | MQTYPE_INT32 | Format komunikatu:<br>MQRFH1<br>MQRFH2<br>PCF |

### *Porządkowanie komunikatów*

W przypadku konkretnego tematu komunikaty są publikowane przez menedżer kolejek w tej samej kolejności, w jakiej są odbierane z aplikacji publikujących (z zastrzeżeniem zmiany kolejności na podstawie priorytetu komunikatu).

Porządkowanie komunikatów zwykle oznacza, że każdy subskrybent odbiera komunikaty z konkretnego menedżera kolejek w konkretnym temacie od konkretnego publikatora w kolejności, w jakiej zostały opublikowane przez ten publikator.

Jednak tak jak w przypadku wszystkich komunikatów IBM MQ , czasami komunikaty mogą być dostarczane w nieodpowiedniej kolejności. Taka sytuacja może wystąpić w następujących sytuacjach:

- Jeśli łącze w sieci zostanie wyłączone, a kolejne komunikaty będą przekierowywane przez inne łącze
- Jeśli kolejka zostanie tymczasowo zapełniona lub zablokowana, komunikat jest umieszczany w kolejce niedostarczonych komunikatów i z tego powodu jest opóźniony, podczas gdy kolejne komunikaty są przesyłane bezpośrednio.
- Jeśli administrator usunie menedżer kolejek, gdy publikatory i subskrybenci nadal działają, powodując umieszczenie komunikatów w kolejce niedostarczonych komunikatów i przerwanie subskrypcji.

Jeśli takie okoliczności nie mogą wystąpić, publikacje są zawsze dostarczane w kolejności.

**Uwaga:** Z publikowaniem/subskrybowaniem nie można używać zgrupowanych lub posegmentowanych komunikatów.

## *Przechwytywanie publikacji*

Można przechwycić publikację, zmodyfikować ją, a następnie opublikować ją ponownie, zanim dotrze do innego subskrybenta.

Można przechwycić publikację, zanim dotrze ona do subskrybenta, aby wykonać jedną z następujących czynności:

- Dołącz do komunikatu dodatkowe informacje
- Zablokuj komunikat
- Transformowanie komunikatu

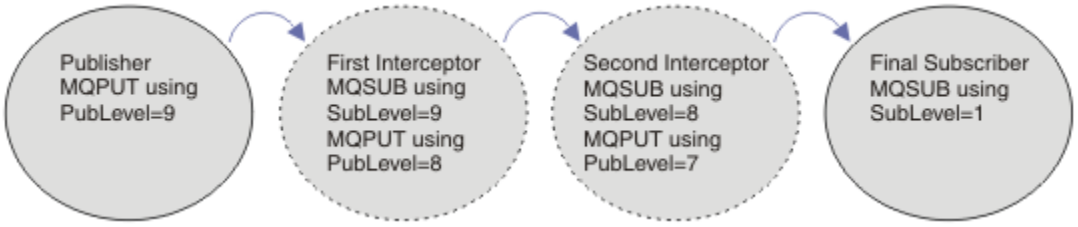
Tę samą operację można wykonać dla każdego komunikatu lub zmienić operację w zależności od subskrypcji, komunikatu lub nagłówka komunikatu.

**Odsyłacze pokrewne**

MQ_PUBLISH_EXIT-Wyjście publikowania

*Poziomy subskrypcji*

Ustaw poziom subskrypcji, aby przechwytywać publikację przed osiągnięciem subskrybentów końcowych. Przechwytujący subskrybent subskrybuje dane na wyższym poziomie subskrypcji, a następnie publikuje dane na niższym poziomie publikacji. Budowanie łańcucha przechwytywania subskrybentów w celu przetwarzania komunikatów dla publikacji przed jej dostarczeniem do subskrybentów końcowych.




*Rysunek 86. Sekwencja przechwytujących subskrybentów*

Aby przechwycić publikację, należy użyć atrybutu **MQSD** `SubLevel` . Po przechwyceniu komunikatu można go transformować, a następnie ponownie publikować na niższym poziomie, zmieniając atrybut **MQPMO** `PubLevel` . Komunikat jest następnie przesyłany do subskrybentów końcowych lub jest przechwytywany ponownie przez subskrybent pośredni na niższym poziomie subskrypcji.

Przechwytujący subskrybent zwykle transformuje komunikat przed jego ponownym opublikowaniem. Sekwencja przechwytywania subskrybentów tworzy przepływ komunikatów. Można również nie publikować ponownie przechwyconej publikacji: subskrybenci na niższych poziomach subskrypcji nie otrzymają komunikatu.

Upewnij się, że przechwytywacz odbiera publikacje przed wszystkimi innymi subskrybentami. Ustaw poziom subskrypcji przechwytywacza na wyższy niż w przypadku innych subskrybentów. Domyślnie subskrybenci mają SubLevel o wartości 1. Najwyższa wartość to 9. Publikacja musi zaczynać się od łańcucha PubLevel co najmniej tak wysokiego, jak najwyższy SubLevel. Początkowo publikuj z domyślną wartością PubLevel 9.

- Jeśli w temacie występuje jeden subskrybent przechwytywania, należy ustawić właściwość SubLevel na wartość 9.
- W przypadku wielu przechwytujących aplikacji w temacie należy ustawić niższy poziom SubLevel dla każdego kolejnego przechwytywacza.
- Można zaimplementować maksymalnie 8 przechwytujących aplikacje z poziomami subskrypcji od 9 do 2 włącznie. Odbiorca końcowy komunikatu ma SubLevel o wartości 1.

Przechwytywacz o najwyższym poziomie subskrypcji, który jest równy lub niższy od poziomu PubLevel publikacji, otrzymuje publikację jako pierwszy. Skonfiguruj tylko jeden subskrybent przechwytywania dla tematu na określonym poziomie subskrypcji. Posiadanie wielu subskrybentów na określonym poziomie subskrypcji powoduje wysłanie wielu kopii publikacji do końcowego zestawu aplikacji subskrybujących.

Subskrybent z SubLevel o wartości 0 jest używany jako catchall. Odbiera ona publikację, jeśli żaden subskrybent końcowy nie otrzyma komunikatu. Subskrybent z SubLevel o wartości 0 może być używany do monitorowania publikacji, które nie zostały odebrane przez innych subskrybentów.

## Programowanie przechwytywacza

Użyj opcji subskrypcji opisanych w sekcji Tabela 125 na stronie 867.

*Tabela 125. Opcje subskrypcji na potrzeby przechwytywania subskrybentów*

| **Opcja subskrypcji** | **Uwagi** |
|---|---|
| MQSO_SET_CORREL_ID i SubCorrelId ustawione na MQCI_NONE | Zachowaj wartość CorrelId przechwyconej publikacji w taki sam sposób, jak w oryginalnej publikacji.<br>**Uwaga:** Nie można przekazać identyfikatora korelacji publikacji w hierarchii. Pole jest używane przez menedżer kolejek. |
| PubPriority ustaw na MQPRI_PRIORITY_AS_PUBLISHED | Priorytet przechwyconej publikacji jest taki sam, jak priorytet oryginalnej publikacji. |

Opcje w pliku Tabela 125 na stronie 867 muszą być używane przez wszystkie przechwytywane subskrybenty. W wyniku tego identyfikator korelacji i priorytet komunikatu nie są modyfikowane w stosunku do ustawienia oryginalnego publikatora.

Gdy przechwytujący subskrybent przetworzył publikację, ponownie publikuje komunikat w tym samym temacie na poziomie PubLevel o jeden niższym niż SubLevel własnej subskrypcji. Jeśli przechwytujący subskrybent ustawił wartość SubLevel na 9, ponownie publikuje komunikat z wartością PubLevel ustawioną na 8.

Aby poprawnie opublikować komunikat, należy podać kilka informacji z oryginalnej publikacji. Użyj ponownie tego samego parametru **MQMD** , co w oryginalnym komunikacie, i ustaw parametr MQPMO_PASS_ALL_CONTEXT , aby wszystkie informacje w pliku **MQMD** zostały przekazane do następnego subskrybenta. Skopiuj wartości z właściwości komunikatu, które przedstawia Tabela 126 na stronie 868 , do odpowiednich pól ponownie opublikowanego komunikatu. Subskrybent przechwytujący może zmienić te wartości. Użyj operatora OR, aby dodać dodatkowe wartości do **MQPMO**. Pole Opcje służące do łączenia opcji umieszczania komunikatu.

Kolejkę publikacji należy otworzyć jawnie, zamiast używać zarządzanej kolejki publikacji. Nie można ustawić parametru MQSO_SET_CORREL_ID dla kolejki zarządzanej. Nie można również ustawić

parametru MQOO_SAVE_ALL_CONTEXT w kolejce zarządzanej. Patrz fragmenty kodu wymienione w sekcji “Przykłady” na stronie 868.

*Tabela 126. Wartości parametru MQPUT dla ponownie opublikowanych komunikatów*

| **Ponownie publikuj komunikat przy użyciu komendy MQPUT** | **Informacje w komunikacie publikacji** |
|---|---|
| **MQOD**. ObjectString | Właściwość komunikatu MQTopicString |
| **MQPMO**. Options | Właściwość komunikatu MQPubOptions |

Subskrybent końcowy może ustawić opcje subskrypcji w inny sposób. Na przykład może on jawnie ustawić priorytet publikacji, a nie MQPRI_PRIORITY_AS_PUBLISHED. Ustawienia subskrybenta końcowego mają wpływ tylko na publikację z ostatniego subskrybenta przechwytującego w łańcuchu.

## Zachowane publikacje

Zachowana publikacja musi zostać zachowana po przechwyceniu przez skopiowanie oryginalnych opcji put-message do ponownie opublikowanego komunikatu.

Opcja MQPMO_RETAIN jest ustawiana przez publikator. Każdy przechwytujący subskrybent musi przekazać parametr MQPubOptions do opcji put-message ponownie opublikowanego komunikatu, jak to pokazano na ilustracji Tabela 126 na stronie 868. Kopiowanie opcji put-message powoduje zachowanie opcji ustawionych przez oryginalny publikator, w tym informacji o tym, czy publikacja ma być zachowana.

Gdy publikacja kończy przejście w dół łańcucha przechwytujących subskrybentów i jest dostarczana do subskrybentów końcowych, jest ona ostatecznie zachowywana. Nowi subskrybenci, na SubLevel 1, żądający zachowanej publikacji, otrzymują ją bez dalszego przechwytywania. Subskrybenci na poziomie SubLevel większym niż 1 nie są wysyłani do zachowanej publikacji. W rezultacie zachowana publikacja nie jest modyfikowana przez łańcuch przechwytywania subskrybentów po raz drugi.

### Przykłady

Przykładami są fragmenty kodu, które można łączyć w celu zbudowania przechwytywającego subskrybenta. Kod jest napisany jako zwięzły, a nie jako jakość produkcji.

Dyrektywy preprocesora w sekcji Rysunek 87 na stronie 868 definiują dwie właściwości, które mają zostać wyodrębnione z komunikatów publikacji wymaganych przez wywołanie MQI produktu MQINQMP .

```
#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
```

*Rysunek 87. Dyrektywy preprocesora*

Rysunek 88 na stronie 869 zawiera listę deklaracji używanych we fragmentach kodu. Oprócz podświetlonych terminów deklaracje są standardowe dla aplikacji IBM MQ .

Podświetlone opcje Put i Get są inicjowane w celu przekazania całego kontekstu. Podświetlone MQTOPICSTRING i MQPUBOPTIONS są inicjatorami MQCHARV dla nazw właściwości zdefiniowanych w dyrektywach preprocesora. Nazwy są przekazywane do MQINQMP.

```
int main(int argc, char **argv) {
    MQLONG   Reason            = MQRC_NONE;
    MQLONG   CompCode          = MQCC_OK;
    MQHCONN  Hcon              = MQHC_UNUSABLE_HCONN;
    MQCHAR   QMName[49]        = " ";
    MQCMHO   CrtMsgHOpts       = {MQCMHO_DEFAULT};
    MQHMSG   Hmsg              = MQHM_NONE;
    MQMD     md                = {MQMD_DEFAULT};
    MQHOBJ   gHobj             = MQHO_NONE;
    MQOD     getOD             = {MQOD_DEFAULT};
    MQGMO    gmo               = {MQGMO_DEFAULT};
    MQLONG   GO_Options        = MQOO_INPUT_AS_Q_DEF
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_SAVE_ALL_CONTEXT;
    MQLONG   GC_Options        = MQCO_DELETE_PURGE;
    MQHOBJ   Hsub              = MQHO_NONE;
    MQSD     sd                = {MQSD_DEFAULT};
    MQLONG   SC_Options        = MQCO_NONE;
    MQHOBJ   pHobj             = MQHO_NONE;
    MQOD     putOD             = {MQOD_DEFAULT};
    MQLONG   PO_Options        = MQOO_OUTPUT
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_PASS_ALL_CONTEXT;
    MQLONG   PC_Options        = MQCO_NONE;
    MQPMO    pmo               = {MQPMO_DEFAULT};
    MQIMPO   InqPropOpts       = {MQIMPO_DEFAULT};
    MQPD     PropDesc          = {MQPD_DEFAULT};
    MQLONG   Type              = MQTYPE_AS_SET;
    MQCHARV  TopStrProp        = {MQTOPICSTRING};
    MQCHARV  PubOptProp        = {MQPUBOPTIONS};
    MQLONG   DataLength        = 0;
    MQBYTE   buffer[256]       = "";
    MQLONG   buflen            = sizeof(buffer) - 1;
    MQLONG   messlen           = 0;
    char     TopStrBuf[256]    = "Initial value";
    int      i                 = 0;
```

*Rysunek 88. Deklaracje*

Inicjalizacje, które nie są łatwo wykonywane w deklaracjach, są przedstawione w sekcji Rysunek 89 na stronie 870. Podświetlone wartości wymagają wyjaśnienia.

**SYSTEM.NDURABLE.MODEL.QUEUE**
W tym przykładzie zamiast używania programu MQSUB do otwierania zarządzanej subskrypcji nietrwałej, do utworzenia tymczasowej kolejki dynamicznej używana jest kolejka modelowa SYSTEM.NDURABLE.MODEL.QUEUE. Jego uchwyt jest przekazywany do MQSUB. Bezpośrednie otwarcie kolejki umożliwia zapisanie całego kontekstu komunikatu i ustawienie opcji subskrypcji MQSO_SET_CORREL_ID.

**MQGMO_CURRENT_VERSION**
Ważne jest, aby użyć bieżącej wersji większości struktur IBM MQ . Pola, takie jak gmo.MsgHandle , są dostępne tylko w najnowszej wersji struktur sterujących.

**MQGMO_PROPERTIES_IN_HANDLE**
Łańcuch tematu i opcje umieszczania komunikatu ustawione w oryginalnej publikacji mają zostać pobrane przez przechwytującego subskrybenta przy użyciu właściwości komunikatu. Alternatywą jest odczytanie bezpośrednio struktury **MQRFH2** w komunikacie.

**MQSO_SET_CORREL_ID**
Użyj MQSO_SET_CORREL_ID w połączeniu z,

```
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
```

Efektem tych opcji jest przekazanie identyfikatora korelacji. Identyfikator korelacji ustawiony przez oryginalny publikator jest umieszczany w polu identyfikatora korelacji publikacji, która jest odbierana przez przechwytującego subskrybenta. Każdy przechwytujący subskrybent przekazuje ten sam identyfikator korelacji. Subskrybent końcowy ma wtedy możliwość odebrania tego samego identyfikatora korelacji.

**Uwaga:** Jeśli publikacja jest przekazywana za pośrednictwem hierarchii publikowania/subskrypcji, identyfikator korelacji nigdy nie jest zachowywany.

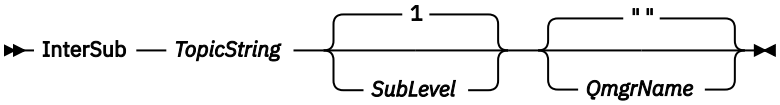
**MQPRI_PRIORITY_AS_PUBLISHED**
Publikacja jest umieszczana w kolejce publikacji z tym samym priorytetem komunikatu, z którym została opublikowana.

```
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;
```

*Rysunek 89. Inicjacje*

Rysunek 90 na stronie 871 przedstawia fragment kodu służący do odczytywania parametrów wiersza komend, kończenia inicjowania i tworzenia subskrypcji przechwytywania.

Uruchom program za pomocą komendy,




Aby zapewnić jak najbardziej niezauważalną obsługę błędów, kod przyczyny z każdego wywołania MQI jest przechowywany w innym elemencie tablicy. Po każdym wywołaniu sprawdzany jest kod zakończenia, a jeśli wartością jest MQCC_FAIL, sterowanie wychodzi z bloku kodu `do { } while(0)`.

Dwa godne uwagi wiersze kodu to:

**pmo.PubLevel = sd.SubLevel - 1;**
Ustawia poziom publikacji dla ponownie opublikowanego komunikatu na jeden niższy niż poziom subskrypcji przechwytywacza.

**gmo.MsgHandle = Hmsg;**
Udostępnia uchwyt komunikatu dla MQGET w celu zwrócenia właściwości komunikatu.

```
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;
```

*Rysunek 90. Przygotowanie do przechwytywania publikacji*

Główny fragment kodu, Rysunek 91 na stronie 872, pobiera komunikaty z kolejki publikacji. Wysyła ona zapytanie do właściwości komunikatu i ponownie publikuje komunikaty przy użyciu łańcucha tematu i oryginalnego pliku **MQPMO**. Właściwości opcji publikacji.

W tym przykładzie dla publikacji nie jest wykonywana żadna transformacja. Łańcuch tematu ponownie opublikowanej publikacji jest zawsze zgodny z łańcuchem tematu subskrybowanego przez przechwytującego subskrybenta. Jeśli przechwytujący subskrybent jest odpowiedzialny za przechwytywanie wielu subskrypcji wysyłanych do tej samej kolejki publikacji, może być konieczne wysłanie zapytania do łańcucha tematu w celu rozróżnienia publikacji zgodnych z różnymi subskrypcjami.

Wywołania do MQINQMP są podświetlane. Właściwości łańcucha tematu i opcji umieszczania komunikatu w publikacji są zapisywane bezpośrednio w wyjściowych strukturach sterujących. Jedynym powodem zmiany pola długości MQCHARV w programie putOD.ObjectString z jawnej długości na łańcuch zakończony znakiem o kodzie zero jest użycie funkcji printf do wyprowadzania łańcucha.

```
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;
}
```

*Rysunek 91. Przechwyć publikację i opublikuj ponownie*

Ostatni fragment kodu jest przedstawiony w sekcji Rysunek 92 na stronie 872.

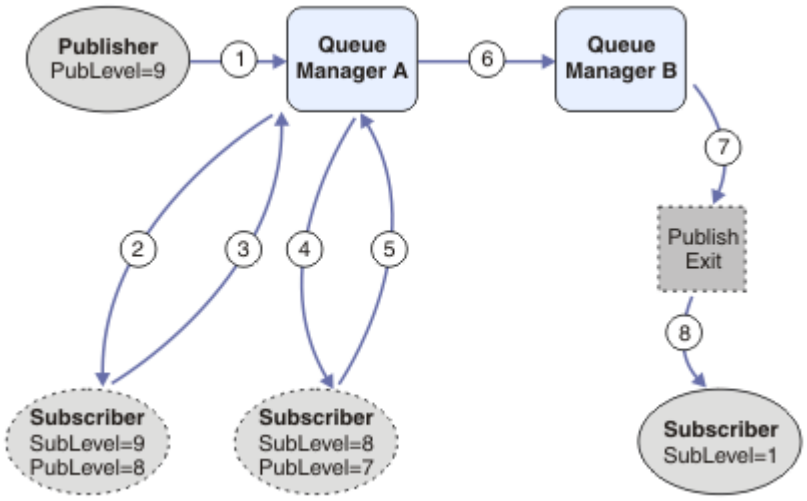
```
    } while (0);
    if (CompCode == MQCC_FAILED && Reason != MQRC_NO_MSG_AVAILABLE)
        printf("MQI Call failed with reason code %d\n", Reason);
    if (Hsub!= MQHO_NONE)
        MQCLOSE(Hcon, &Hsub, SC_Options, &CompCode, &Reason);
    if (Hcon!= MQHC_UNUSABLE_HCONN)
        MQDISC(&Hcon, &CompCode, &Reason);
}
```

*Rysunek 92. Zakończenie*

*Przechwytywanie publikacji i rozproszona publikacja/subskrypcja*
Podczas wdrażania przechwytujących subskrybentów lub wyjść publikowania w topologii rozproszonej publikowania/subskrypcji należy postępować zgodnie z prostym wzorcem. Należy wdrożyć przechwytywane subskrybenty w tych samych menedżerach kolejek, w których znajdują się publikatory, oraz wyjść publikowania w tych samych menedżerach kolejek, w których znajdują się subskrybenci końcowi.

Na rysunku Rysunek 93 na stronie 873 pokazano dwa menedżery kolejek połączone w klastrze publikowania/subskrybowania. Publikator tworzy publikację w temacie klastra na poziomie publikacji 9. Numerowane strzałki przedstawiają sekwencję kroków wykonywanych przez publikację w miarę jej przepływu do subskrybentów tematu klastra. Publikacja jest przechwytywana przez subskrybenta za pomocą opcji `Sublevel` 9 i ponownie publikowana za pomocą opcji `Publevel` 8. Jest on przechwytywany ponownie przez subskrybent na `poziomie podrzędnym` 8. Subskrybent ponownie publikuje dane na poziomie `Publevel` 7. Subskrybent proxy udostępniany przez menedżer kolejek przekazuje publikację do menedżera kolejek B, w którym oprócz subskrybenta końcowego wdrożono również wyjście publikowania. Publikacja jest przetwarzana przez wyjście publikowania, zanim zostanie

ostatecznie odebrana przez subskrybent końcowy na `poziomie podrzędnym` 1. Przechwytywane subskrybenty i wyjście publikowania są wyświetlane z niepoprawnymi konturami.




*Rysunek 93. Przechwytywanie i publikowanie wyjścia w klastrze*

Celem prostego wzorca jest, aby każdy subskrybent otrzymujący publikację otrzymał identyczną publikację. Publikacja przechodzi przez tę samą sekwencję transformacji bez względu na to, gdzie subskrybent jest połączony. Prawdopodobnie należy unikać sytuacji, w której sekwencja transformacji jest różna, w zależności od tego, gdzie są połączone publikatory lub subskrybenci końcowi. Zasadnym wyjątkiem byłoby dostosowanie publikacji ostatecznie dostarczonej każdemu indywidualnemu subskrybentowi. Użyj wyjścia publikowania, aby dostosować publikację na podstawie kolejki, do której publikacja została ostatecznie dostarczona.

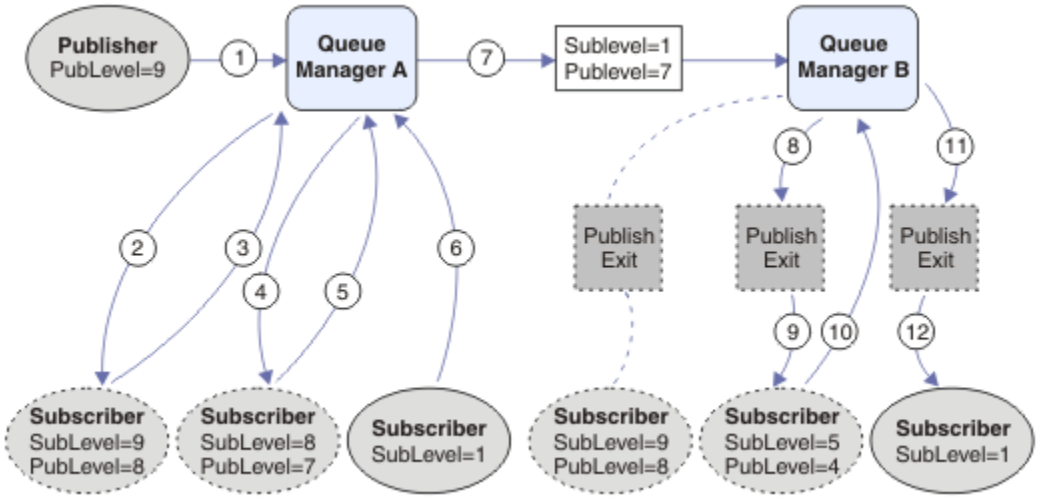
Należy dokładnie rozważyć, gdzie wdrożyć przechwytywane subskrybenty i wyjścia publikowania w topologii rozproszonej publikowania/subskrypcji. Prosty wzorzec wdraża przechwytywane subskrybenty w tym samym menedżerze kolejek, co publikatory, i publikuje wyjścia w tych samych menedżerach kolejek, co subskrybenci końcowi.

## Anty-wzór

Rysunek 94 na stronie 874 pokazuje, jak sprawy mogą się nie układać, jeśli nie podąża się za prostym wzorcem. Aby skomplikować wdrożenie, do menedżera kolejek A dodawany jest subskrybent końcowy, a do menedżera kolejek B dodawane są dwa dodatkowe subskrybenty przechwytujące.

Publikacja jest przekazywana do menedżera kolejek B pod adresem `PubLevel` 7, gdzie jest przechwytywana przez subskrybent pod adresem `SubLevel` 5 przed użyciem jej przez ostatniego subskrybenta pod adresem `SubLevel` 1. Wyjście publikowania przechwytuje publikację przed przekazaniem jej zarówno do konsumenta przechwytującego, jak i do konsumenta końcowego w menedżerze kolejek B. Publikacja dociera do końcowego subskrybenta w menedżerze kolejek A bez przetwarzania przez wyjście publikowania.

W topologii publikowania/subskrypcji subskrybuj subskrybenty proxy na poziomie `SubLevel` 1i przekaż ustawienie `PubLevel` ustawione przez ostatniego przechwytującego subskrybenta. W produkcie Rysunek 94 na stronie 874publikacja nie jest przechwytywana przez subskrybent przy użyciu poziomu `SubLevel` 9 w menedżerze kolejek B.




*Rysunek 94. Złożone wdrażanie przechwytujących subskrybentów*

## *Opcje publikowania*

Dostępnych jest kilka opcji sterujących sposobem publikowania komunikatów.

### Wstrzymanie odpowiedzi od abonentów

Jeśli subskrybenty nie mają mieć możliwości odpowiadania na odebrane publikacje, można wstrzymać informacje w polach ReplyToQ i ReplyTomenedżera kolejek MQMD za pomocą opcji MQPMO_SUPPRESS_REPLYTO put-message. Jeśli ta opcja jest używana, menedżer kolejek usuwa te informacje z deskryptora MQMD po odebraniu publikacji przed przekazaniem jej do subskrybentów.

Ta opcja nie może być używana w połączeniu z opcją raportu, która wymaga zapytania ReplyToQ, jeśli próba wywołania zakończy się niepowodzeniem z błędem MQRC_MISSING_REPLY_TO_Q.

### Poziom publikacji

Użycie poziomów publikacji jest sposobem na kontrolowanie, którzy subskrybenci otrzymują publikację. Poziom publikacji oznacza poziom subskrypcji, której dotyczy publikacja. Tylko subskrypcje, których najwyższy poziom subskrypcji jest niższy lub równy poziomowi publikacji, otrzymają publikację.
Ta wartość musi należeć do zakresu od 0 do 9; zero oznacza najniższy poziom publikacji.
Wartością początkową tego pola jest 9. Jednym z zastosowań poziomów publikacji i subskrypcji jest przechwytywanie publikacji.

### Sprawdzanie, czy publikacja nie została dostarczona do żadnego z subskrybentów

Aby sprawdzić, czy publikacja nie została dostarczona do żadnego z subskrybentów, należy użyć opcji MQPMO_WARN_IF_NO_SUBS_MATCHED put-message z wywołaniem MQPUT. Jeśli operacja put zwróciła kod zakończenia MQCC_WARNING i kod przyczyny MQRC_NO_SUBS_MATCHED, publikacja nie została dostarczona do żadnych subskrypcji. Jeśli opcja MQPMO_RETAIN jest określona w operacji put, komunikat jest zachowywany i dostarczany do każdej następnie zdefiniowanej zgodnej subskrypcji. W rozproszonym systemie publikowania/subskrypcji kod przyczyny MQRC_NO_SUBS_MATCHED jest zwracany tylko wtedy, gdy w menedżerze kolejek nie zarejestrowano żadnych subskrypcji proxy dla tematu.

## *Opcje subskrypcji*

Dostępnych jest kilka opcji sterujących sposobem obsługi subskrypcji komunikatów.

### Trwałość komunikatu

Menedżery kolejek zachowują trwałość publikacji, które przekazują do subskrybentów zgodnie z ustawionym przez publikatora. Publikator ustawia trwałość na jedną z następujących opcji:

**0**
Nietrwałe

**1**
Trwałe

**2**
Trwałość jako definicja kolejki/tematu

W przypadku publikowania/subskrypcji publikator tłumaczy obiekt tematu i **topicString** na rozstrzygnięty obiekt tematu. Jeśli publikator określa trwałość jako definicję kolejki/tematu, dla publikacji jest ustawiana domyślna trwałość z rozstrzygniętego obiektu tematu.

## Zachowane publikacje

Aby sterować momentem odbierania zachowanych publikacji, subskrybenci mogą korzystać z dwóch opcji subskrypcji:

**Publikuj tylko na żądanie, MQSO_PUBLIC ATIONS_ON_REQUEST**

Jeśli subskrybent ma mieć kontrolę nad momentem odbierania publikacji, można użyć opcji subskrypcji MQSO_PUBLIC ATIONS_ON_REQUEST. Subskrybent może następnie sterować tym, kiedy odbiera publikacje za pomocą wywołania MQSUBRQ (określającego uchwyt Hsub, który został zwrócony z oryginalnego wywołania MQSUB) w celu żądania przesłania zachowanej publikacji tematu. Subskrybenci korzystający z opcji subskrypcji MQSO_PUBLIC ATIONS_ON_REQUEST nie otrzymują żadnych niezachowanych publikacji.

Jeśli określono wartość MQSO_PUBLIC ATIONS_ON_REQUEST, do pobrania dowolnej publikacji należy użyć komendy MQSUBRQ. Jeśli nie jest używana opcja MQSO_PUBLIC ATIONS_ON_REQUEST, komunikaty są wyświetlane w miarę ich publikowania.

Jeśli subskrybent używa wywołania MQSUBRQ i używa znaków wieloznacznych w temacie subskrypcji, subskrypcja może być zgodna z wieloma tematami lub węzłami w drzewie tematów, a wszystkie tematy z zachowanymi komunikatami (jeśli istnieją) zostaną wysłane do subskrybenta.

Ta opcja może być szczególnie pomocna w przypadku używania jej z trwałymi subskrypcjami, ponieważ menedżer kolejek będzie nadal wysyłać publikacje do subskrybenta, jeśli został on trwale zasubskrybowany, nawet jeśli ta aplikacja subskrybenta nie jest uruchomiona. Może to prowadzić do gromadzenia komunikatów w kolejce subskrybenta. Tej operacji budowania można uniknąć, jeśli subskrybent rejestruje się przy użyciu opcji MQSO_PUBLIC ATIONS_ON_REQUEST. Alternatywnie można użyć nietrwałych subskrypcji, jeśli jest to odpowiednie dla aplikacji, aby uniknąć budowania niechcianych komunikatów.

Jeśli subskrypcja jest trwała, a publikator używa zachowanych publikacji, aplikacja subskrybenta może użyć wywołania MQSUBRQ do odświeżenia informacji o stanie po restarcie. Następnie subskrybent musi okresowo odświeżać swój stan przy użyciu wywołania MQSUBRQ.

W wyniku wywołania MQSUB przy użyciu tej opcji nie będą wysyłane żadne publikacje. Trwała subskrypcja, która została wznowiona po rozłączeniu, będzie korzystać z opcji MQSO_PUBLIC ATIONS_ON_REQUEST, jeśli pierwotna subskrypcja została skonfigurowana do używania tej opcji.

**Tylko nowe publikacje, MQSO_NEW_PUBLIC**

Jeśli zachowana publikacja istnieje w temacie, każdy subskrybent, który dokonał subskrypcji po publikacji, otrzyma kopię tej publikacji. Jeśli subskrybent nie chce otrzymywać żadnych publikacji, które zostały wykonane wcześniej niż subskrypcja, może użyć opcji subskrypcji MQSO_NEW_PUBLIC ONLY.

## Grupowanie subskrypcji

Należy rozważyć grupowanie subskrypcji, jeśli skonfigurowano kolejkę do odbierania publikacji i istnieje pewna liczba nakładających się subskrypcji dostarczających publikacje do tej samej kolejki. Ta sytuacja jest podobna do przykładu w sekcji Nakładanie subskrypcji.

Aby uniknąć odbierania zduplikowanych publikacji, należy ustawić opcję MQSO_GROUP_SUB podczas subskrybowania tematu. W rezultacie, jeśli więcej niż jedna subskrypcja w grupie jest zgodna z tematem publikacji, tylko jedna subskrypcja jest odpowiedzialna za umieszczenie publikacji w kolejce. Pozostałe subskrypcje, które są zgodne z tematem publikacji, są ignorowane.

Subskrypcja odpowiedzialna za umieszczenie publikacji w kolejce jest wybierana na podstawie tego, że ma najdłuższy pasujący łańcuch tematu przed napotkaniem znaków wieloznacznych. Można go traktować jako najbliższą zgodną subskrypcję. Jego właściwości są propagowane do publikacji, w tym informacje o tym, czy ma ona właściwość MQSO_NOT_OWN_PUBS . Jeśli tak, żadna publikacja nie zostanie dostarczona do kolejki, nawet jeśli inne zgodne subskrypcje mogą nie mieć właściwości MQSO_NOT_OWN_PUBS .

Nie można umieścić wszystkich subskrypcji w jednej grupie w celu wyeliminowania zduplikowanych publikacji. Zgrupowane subskrypcje muszą spełniać następujące warunki:

1. Żadna z subskrypcji nie jest zarządzana.
2. Grupa subskrypcji dostarcza publikacje do tej samej kolejki.
3. Każda subskrypcja musi mieć ten sam poziom subskrypcji.
4. Komunikat publikacji dla każdej subskrypcji w grupie ma ten sam identyfikator korelacji.

   Aby zapewnić, że każda subskrypcja będzie skutkować komunikatem publikacji o takim samym identyfikatorze korelacji, należy ustawić właściwość MQSO_SET_CORREL_ID w celu utworzenia własnego identyfikatora korelacji w publikacji i ustawić tę samą wartość w polu **SubCorrelId** w każdej subskrypcji. Nie należy ustawiać parametru **SubCorrelId** na wartość MQCI_NONE.

Więcej informacji na ten temat zawiera sekcja ../refdev/q100080_.dita#q100080_/mqso_group_sub .

## Uzyskiwanie informacji o atrybutach obiektu i ustawianie ich

Atrybuty są właściwościami definiującymi charakterystykę obiektu IBM MQ .

Wpływają one na sposób, w jaki menedżer kolejek przetwarza obiekt. Atrybuty każdego typu obiektu IBM MQ zostały szczegółowo opisane w sekcji Atrybuty obiektów.

Niektóre atrybuty są ustawiane podczas definiowania obiektu i mogą być zmieniane tylko za pomocą komend IBM MQ . Przykładem takiego atrybutu jest domyślny priorytet komunikatów umieszczanych w kolejce. Działanie menedżera kolejek ma wpływ na inne atrybuty i może zmieniać się w czasie. Przykładem może być bieżące zapełnienie kolejki.

Za pomocą wywołania MQINQ można uzyskać informacje o bieżących wartościach większości atrybutów. Interfejs MQI udostępnia również wywołanie MQSET, za pomocą którego można zmienić niektóre atrybuty kolejki. Nie można użyć wywołań MQI do zmiany atrybutów żadnego innego typu obiektu. Zamiast tego należy użyć jednego z następujących zasobów:

- ALW Narzędzie MQSC opisane w sekcji Komendy MQSC.
- IBM i Komendy CL CHGMQMx, które zostały opisane w sekcji Skorowidz komend CL dla produktu IBM ilub narzędzia MQSC.
- z/OS Komendy operatora ALTER lub komendy DEFINE z opcją REPLACE, które są opisane w sekcji Komendy MQSC.

**Uwaga:** Nazwy atrybutów obiektów są wyświetlane w tej dokumentacji w postaci, w jakiej są używane w wywołaniach MQINQ i MQSET. Jeśli do definiowania, modyfikowania lub wyświetlania atrybutów używane są komendy systemu IBM MQ , należy zidentyfikować atrybuty przy użyciu słów kluczowych przedstawionych w opisach komend w odsyłaczach do tematów.

Zarówno wywołania MQINQ, jak i wywołania MQSET używają tablic selektorów do identyfikowania tych atrybutów, które mają zostać wyświetlone lub ustawione. Dla każdego atrybutu, z którym można pracować, istnieje selektor. Nazwa selektora ma przedrostek określony przez rodzaj atrybutu:

| *Tabela 127. Przedrostki nazw selektorów* | |
|---|---|
| **Przedrostek** | **Opis** |
| MQCA_ | Te selektory odwołują się do atrybutów, które zawierają dane znakowe (na przykład nazwę kolejki). |
| MQIA_ | Te selektory odwołują się do atrybutów, które zawierają wartości liczbowe (na przykład *CurrentQueueDepth*, liczbę komunikatów w kolejce) lub wartość stałą (na przykład *SyncPoint*, niezależnie od tego, czy menedżer kolejek obsługuje punkty synchronizacji). |

Przed użyciem wywołań MQINQ lub MQSET aplikacja musi być połączona z menedżerem kolejek i należy użyć wywołania MQOPEN, aby otworzyć obiekt w celu ustawienia lub zapytania o atrybuty. Te operacje są opisane w sekcjach "Nawiązywanie i rozłączanie połączenia z menedżerem kolejek" na stronie 758 i "Otwieranie i zamykanie obiektów" na stronie 766.

Aby uzyskać więcej informacji na temat uzyskiwania informacji o atrybutach obiektów i ich ustawiania, należy użyć następujących odsyłaczy:

- "Uzyskiwanie informacji o atrybutach obiektu" na stronie 878
- "Niektóre przypadki, w których wywołanie MQINQ nie powiodło się" na stronie 878
- "Ustawianie atrybutów kolejki" na stronie 879

**Pojęcia pokrewne**

"Przegląd interfejsu kolejki komunikatów" na stronie 745
Sekcja zawiera informacje na temat komponentów interfejsu kolejki komunikatów (Message Queue Interface-MQI).

"Nawiązywanie i rozłączanie połączenia z menedżerem kolejek" na stronie 758
Aby można było używać usług programistycznych IBM MQ , program musi mieć połączenie z menedżerem kolejek. Ten temat zawiera informacje o nawiązywaniu połączenia z menedżerem kolejek i rozłączaniu się z nim.

"Otwieranie i zamykanie obiektów" na stronie 766
Informacje te udostępniają wgląd w otwieranie i zamykanie obiektów IBM MQ .

"Umieszczanie komunikatów w kolejce" na stronie 777
Ta sekcja zawiera informacje na temat umieszczania komunikatów w kolejce.

"Pobieranie komunikatów z kolejki" na stronie 792
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

"Zatwierdzanie i wycofywanie jednostek pracy" na stronie 879
W tej sekcji opisano sposób zatwierdzania i wycofywania wszelkich odtwarzalnych operacji pobierania i umieszczania, które wystąpiły w jednostce pracy.

"Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy" na stronie 891
Informacje o wyzwalaczach i sposobie uruchamiania aplikacji IBM MQ za pomocą wyzwalaczy.

"Praca z funkcją MQI i klastrami" na stronie 911
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do łączenia w klastry.

"Używanie i pisanie aplikacji w systemie IBM MQ for z/OS" na stronie 916
Aplikacje IBM MQ for z/OS mogą być tworzone z programów działających w wielu różnych środowiskach. Oznacza to, że mogą korzystać z udogodnień dostępnych w więcej niż jednym środowisku.

"Aplikacje mostu IMS i IMS w systemie IBM MQ for z/OS" na stronie 71
Informacje te są pomocne podczas pisania aplikacji IMS przy użyciu produktu IBM MQ.

### *Uzyskiwanie informacji o atrybutach obiektu*

Wywołanie MQINQ służy do uzyskiwania informacji o atrybutach dowolnego typu produktu IBM MQ.

Jako dane wejściowe dla tego wywołania należy podać:

- Uchwyt połączenia.
- Uchwyt obiektu.
- Liczba selektorów.
- Tablica selektorów atrybutów, każdy selektor ma postać MQCA_ * lub MQIA_ *. Każdy selektor reprezentuje atrybut z wartością, której dotyczy zapytanie, a każdy selektor musi być poprawny dla typu obiektu reprezentowanego przez uchwyt obiektu. Selektory można określić w dowolnej kolejności.
- Liczba atrybutów będących liczbami całkowitymi, których dotyczy zapytanie. Należy podać wartość zero, jeśli nie jest wykonywane zapytanie o atrybuty całkowitoliczbowe.
- Długość buforu atrybutów znakowych w pliku *CharAttrLength*. Musi to być co najmniej suma długości wymaganych do przechowania każdego łańcucha atrybutu znakowego. Jeśli nie jest wykonywane zapytanie o atrybuty znaków, należy podać wartość zero.

Dane wyjściowe komendy MQINQ to:

- Zestaw wartości atrybutów całkowitych skopiowanych do tablicy. Liczba wartości jest określana przez parametr *IntAttrCount*. Jeśli *IntAttrCount* lub *SelectorCount* ma wartość zero, ten parametr nie jest używany.
- Bufor, w którym zwracane są atrybuty znakowe. Długość buforu jest określona przez parametr **CharAttrLength** . Jeśli *CharAttrLength* lub *SelectorCount* ma wartość zero, ten parametr nie jest używany.
- Kod zakończenia. Jeśli kod zakończenia wyświetla ostrzeżenie, oznacza to, że wywołanie zostało zakończone tylko częściowo. W takim przypadku należy sprawdzić kod przyczyny.
- Kod przyczyny. Istnieją trzy sytuacje częściowego zakończenia:
  - Selektor nie ma zastosowania do typu kolejki
  - Brak wystarczającej ilości miejsca dla atrybutów całkowitoliczbowych
  - Brak wystarczającej ilości miejsca dla atrybutów znakowych

  Jeśli wystąpi więcej niż jedna z tych sytuacji, zwracana jest pierwsza, która ma zastosowanie.

Jeśli kolejka zostanie otwarta dla danych wyjściowych lub zapytania i zostanie przetłumaczona na nielokalne kolejki klastra, można tylko zapytać o nazwę kolejki, typ kolejki i atrybuty wspólne. Wartości wspólnych atrybutów są wartościami dla wybranej kolejki, jeśli użyto opcji MQOO_BIND_ON_OPEN. Wartości są wartościami dowolnej z możliwych kolejek klastra, jeśli użyto MQOO_BIND_NOT_FIXED lub MQOO_BIND_ON_GROUP lub MQOO_BIND_AS_AS_Q_DEF, a atrybut kolejki **DefBind** to MQBND_BIND_NOT_FIXED. Więcej informacji na ten temat zawierają “MQOPEN i klastry” na stronie 912 i MQOPEN .

**Uwaga:** Wartości zwracane przez wywołanie są obrazem stanu wybranych atrybutów. Atrybuty mogą ulec zmianie, zanim program zacznie działać na zwracane wartości.

Istnieje opis wywołania MQINQ w MQINQ.

### *Niektóre przypadki, w których wywołanie MQINQ nie powiodło się*

Jeśli zostanie otwarty alias w celu uzyskania informacji o jego atrybutach, zostaną zwrócone atrybuty kolejki aliasowej (obiekt IBM MQ używany w celu uzyskania dostępu do innej kolejki), a nie atrybuty kolejki podstawowej.

Jednak definicja kolejki podstawowej, do której alias jest tłumaczony, jest również otwierana przez menedżer kolejek, a jeśli inny program zmieni użycie kolejki podstawowej w odstępie czasu między wywołaniami MQOPEN i MQINQ, wywołanie MQINQ nie powiedzie się i zwróci kod przyczyny MQRC_OBJECT_CHANGED. Wywołanie nie powiedzie się również wtedy, gdy atrybuty obiektu kolejki aliasowej zostaną zmienione.

Podobnie po otwarciu kolejki zdalnej w celu uzyskania informacji o jej atrybutach zwracane są tylko atrybuty lokalnej definicji kolejki zdalnej.

Jeśli zostanie podany co najmniej jeden selektor, który nie jest poprawny dla typu atrybutów kolejki, którego dotyczy zapytanie, wywołanie MQINQ zostanie zakończone z ostrzeżeniem i ustawi dane wyjściowe w następujący sposób:

- W przypadku atrybutów całkowitoliczbowych odpowiednie elementy *IntAttrs* mają ustawioną wartość MQIAV_NOT_APPLICABLE.
- W przypadku atrybutów znakowych odpowiednie części łańcucha *CharAttrs* są ustawione na gwiazdki.

Jeśli zostanie podany co najmniej jeden selektor, który nie jest poprawny dla typu atrybutów obiektu, którego dotyczy zapytanie, wywołanie MQINQ nie powiedzie się i zwróci kod przyczyny MQRC_SELECTOR_ERROR.

Nie można wywołać komendy MQINQ w celu wyszukania kolejki modelowej. Należy użyć narzędzia MQSC lub komend dostępnych na platformie.

### *Ustawianie atrybutów kolejki*

Ten temat zawiera informacje dotyczące ustawiania atrybutów kolejki za pomocą wywołania MQSET.

Za pomocą wywołania MQSET można ustawić tylko następujące atrybuty kolejki:

- *InhibitGet* (ale nie dla kolejek zdalnych)
- *DistList* (nie w systemie z/OS )
- *InhibitPut*
- *TriggerControl*
- *TriggerType*
- *TriggerDepth*
- *TriggerMsgPriority*
- *TriggerData*

Wywołanie MQSET ma takie same parametry jak wywołanie MQINQ. Jednak w przypadku komendy MQSET wszystkie parametry z wyjątkiem kodu zakończenia i kodu przyczyny są parametrami wejściowymi. Nie ma sytuacji częściowego zakończenia.

**Uwaga:** Interfejsu MQI nie można używać do ustawiania atrybutów obiektów IBM MQ innych niż kolejki zdefiniowane lokalnie.

Więcej informacji na temat wywołania MQSET zawiera sekcja MQSET.

## Zatwierdzanie i wycofywanie jednostek pracy

W tej sekcji opisano sposób zatwierdzania i wycofywania wszelkich odtwarzalnych operacji pobierania i umieszczania, które wystąpiły w jednostce pracy.

W tym temacie używane są następujące terminy:

- Zatwierdzanie
- Wycofanie
- Koordynacja punktów synchronizacji
- Punkt synchronizacji
- Jednostka pracy
- zatwierdzanie jednofazowe
- zatwierdzania dwufazowe.

Jeśli użytkownik jest zaznajomiony z tymi warunkami przetwarzania transakcji, może przejść do sekcji “Uwagi dotyczące punktu synchronizacji w aplikacjach IBM MQ” na stronie 881.

**Zatwierdź i wycofaj**
Gdy program umieszcza komunikat w kolejce wewnątrz jednostki pracy, komunikat ten jest widoczny dla innych programów tylko wtedy, gdy program zatwierdza jednostkę pracy. Aby zatwierdzić jednostkę pracy, wszystkie aktualizacje muszą zakończyć się pomyślnie w celu zachowania integralności danych. Jeśli program wykryje błąd i uzna, że operacja umieszczania nie jest trwała, może wycofać jednostkę pracy. Gdy program wykonuje wycofanie, program IBM MQ odtwarza kolejkę, usuwając komunikaty, które zostały umieszczone w kolejce przez tę jednostkę pracy. Sposób, w jaki program wykonuje operacje zatwierdzania i cofania zależy od środowiska, w którym program jest uruchomiony.

Podobnie, gdy program pobiera komunikat z kolejki w ramach jednostki pracy, komunikat ten pozostaje w kolejce do momentu zatwierdzenia przez program jednostki pracy, ale komunikat nie jest dostępny do wczytania przez inne programy. Komunikat jest trwale usuwany z kolejki, gdy program zatwierdza jednostkę pracy. Jeśli program wycofa jednostkę pracy, program IBM MQ odtworzy kolejkę, udostępniając komunikaty do wczytania przez inne programy.

**Koordynacja punktów synchronizacji, punkt synchronizacji, jednostka pracy**
*Koordynacja Syncpoint* to proces, w którym jednostki pracy są zatwierdzane lub wycofywane z zachowaniem integralności danych.

Decyzja o zatwierdzeniu lub wycofaniu zmian jest podejmowana, w najprostszym przypadku, po zakończeniu transakcji. Jednak bardziej użyteczne może być synchronizowanie przez aplikację zmian danych w innych punktach logicznych w ramach transakcji. Te punkty logiczne są nazywane *punktami synchronizacji* (lub *punktami synchronizacji* ). a okres przetwarzania zestawu aktualizacji między dwoma punktami synchronizacji jest nazywany *jednostką pracy*. Kilka wywołań MQGET i MQPUT może być częścią pojedynczej jednostki pracy.

Maksymalna liczba komunikatów w jednostce pracy może być kontrolowana przez atrybut MAXUMSGS komendy ALTER QMGR .

**zatwierdzanie jednofazowe**
Proces *zatwierdzania jednofazowego* to proces, w którym program może zatwierdzać aktualizacje w kolejce bez koordynowania swoich zmian z innymi menedżerami zasobów.

**zatwierdzania dwufazowe.**
Proces *zatwierdzania dwufazowego* to proces, w którym aktualizacje wprowadzane przez program do kolejek systemu IBM MQ mogą być koordynowane z aktualizacjami innych zasobów (na przykład baz danych sterowanych przez produkt Db2 ). W takim procesie aktualizacje wszystkich zasobów są zatwierdzane lub wycofywane razem.

Aby ułatwić obsługę jednostek pracy, produkt IBM MQ udostępnia atrybut **BackoutCount** . Wartość ta jest zwiększana za każdym razem, gdy komunikat w jednostce pracy jest wycofywany. Jeśli komunikat wielokrotnie powoduje nieprawidłowe zakończenie jednostki pracy, wartość parametru *BackoutCount* na koniec przekracza wartość parametru *BackoutThreshold*. Ta wartość jest ustawiana podczas definiowania kolejki. W takiej sytuacji aplikacja może usunąć komunikat z jednostki pracy i umieścić go w innej kolejce, zgodnie z definicją w sekcji *BackoutRequeueQName*. Po przeniesieniu komunikatu jednostka pracy może zostać zatwierdzona.

Aby uzyskać więcej informacji na temat zatwierdzania i wycofywania jednostek pracy, należy użyć następujących odsyłaczy:

- “Uwagi dotyczące punktu synchronizacji w aplikacjach IBM MQ” na stronie 881
- z/OS “Punkty synchronizacji w aplikacjach IBM MQ for z/OS” na stronie 882
- IBM i “Punkty synchronizacji w produkcie CICS dla aplikacji IBM i” na stronie 885
- “Punkty synchronizacji w produkcie IBM MQ for Multiplatforms” na stronie 885
- IBM i “Interfejsy do zewnętrznego menedżera punktów synchronizacji IBM i” na stronie 890

**Pojęcia pokrewne**
“Przegląd interfejsu kolejki komunikatów” na stronie 745

Sekcja zawiera informacje na temat komponentów interfejsu kolejki komunikatów (Message Queue Interface-MQI).

“Nawiązywanie i rozłączanie połączenia z menedżerem kolejek” na stronie 758
Aby można było używać usług programistycznych IBM MQ , program musi mieć połączenie z menedżerem kolejek. Ten temat zawiera informacje o nawiązywaniu połączenia z menedżerem kolejek i rozłączaniu się z nim.

“Otwieranie i zamykanie obiektów” na stronie 766
Informacje te udostępniają wgląd w otwieranie i zamykanie obiektów IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 777
Ta sekcja zawiera informacje na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 792
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Uzyskiwanie informacji o atrybutach obiektu i ustawianie ich” na stronie 876
Atrybuty są właściwościami definiującymi charakterystykę obiektu IBM MQ .

“Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy” na stronie 891
Informacje o wyzwalaczach i sposobie uruchamiania aplikacji IBM MQ za pomocą wyzwalaczy.

“Praca z funkcją MQI i klastrami” na stronie 911
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do łączenia w klastry.

“Używanie i pisanie aplikacji w systemie IBM MQ for z/OS” na stronie 916
Aplikacje IBM MQ for z/OS mogą być tworzone z programów działających w wielu różnych środowiskach. Oznacza to, że mogą korzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje mostu IMS i IMS w systemie IBM MQ for z/OS” na stronie 71
Informacje te są pomocne podczas pisania aplikacji IMS przy użyciu produktu IBM MQ.

## *Uwagi dotyczące punktu synchronizacji w aplikacjach IBM MQ*

Ten temat zawiera informacje o korzystaniu z punktów synchronizacji w aplikacjach IBM MQ .

Zatwierdzanie dwufazowe jest obsługiwane w następujących środowiskach:

- Multi IBM MQ for Multiplatforms
- z/OS CICS Transaction Server dla systemu z/OS
- z/OS TXSeries
- z/OS IMS/ESA
- z/OS Zadanie wsadowe z/OS z RRS
- Inni zewnętrzni koordynatorzy korzystający z interfejsu XA X/Open

Zatwierdzanie jednofazowe jest obsługiwane w następujących środowiskach:

- Multi IBM MQ for Multiplatforms
- z/OS z/OS wsadowe

Więcej informacji na temat interfejsów zewnętrznych zawiera sekcja “Interfejsy do zewnętrznych menedżerów punktów synchronizacji na wielu platformach” na stronie 888oraz dokumentacja interfejsu XA *CAE Specification Distributed Transaction Processing: The XA Specification*(Przetwarzanie rozproszonej transakcji specyfikacji CAE: Specyfikacja interfejsu XA) opublikowana przez grupę otwartą. Menedżery transakcji (takie jak CICS, IMS, Encina i Tuxedo) mogą uczestniczyć w zatwierdzaniu dwufazowym, koordynowanym z innymi zasobami odtwarzalnymi. Oznacza to, że funkcje kolejkowania udostępniane przez produkt IBM MQ mogą zostać wprowadzone w zakres jednostki pracy zarządzanej przez menedżer transakcji.

Przykłady dostarczane z produktem IBM MQ przedstawiają IBM MQ koordynowanie baz danych zgodnych z interfejsem XA. Więcej informacji na temat tych przykładów zawiera sekcja “Korzystanie z przykładowych programów proceduralnych IBM MQ” na stronie 1089.

W aplikacji IBM MQ można określić w każdym wywołaniu put i get, czy wywołanie ma być objęte kontrolą punktu synchronizacji. Aby operacja umieszczania działała pod kontrolą punktu synchronizacji, podczas wywoływania operacji MQPUT należy użyć wartości MQPMO_SYNCPOINT z pola *Options* struktury MQPMO. W przypadku operacji pobierania należy użyć wartości MQGMO_SYNCPOINT z pola *Options* w strukturze MQGMO. Jeśli opcja nie zostanie jawnie wybrana, działanie domyślne zależy od platformy:

- Multi Wartością domyślną elementu sterującego punktu synchronizacji jest NO.
- z/OS Wartością domyślną elementu sterującego punktu synchronizacji jest YES.

Po wywołaniu wywołania MQPUT1 z opcją MQPMO_SYNCPOINT domyślne zachowanie zmienia się, tak że operacja umieszczania jest wykonywana asynchronicznie. Może to spowodować zmianę zachowania niektórych aplikacji, które opierają się na pewnych polach w zwracanych strukturach MQOD i MQMD, ale które teraz zawierają niezdefiniowane wartości. Aplikacja może określić parametr MQPMO_SYNC_RESPONSE, aby zapewnić, że operacja umieszczania jest wykonywana synchronicznie i że wszystkie odpowiednie wartości pól są zakończone.

Po odebraniu przez aplikację kodu przyczyny MQRC_BACKED_OUT w odpowiedzi na wywołanie MQPUT lub MQGET w punkcie synchronizacji aplikacja powinna zwykle wycofać bieżącą transakcję za pomocą wywołania MQBACK, a następnie, w razie potrzeby, ponowić całą transakcję. Jeśli aplikacja odbiera wywołanie MQRC_BACKED_OUT w odpowiedzi na wywołanie MQCMIT lub MQDISC, nie musi wywoływać wywołania MQBACK.

Za każdym razem, gdy wywołanie MQGET jest wycofane, pole *BackoutCount* struktury MQMD danego komunikatu jest zwiększane. Wysoka wartość *BackoutCount* wskazuje komunikat, który był wielokrotnie wycofywany. Może to wskazywać na problem z tym komunikatem, który należy zbadać. Szczegółowe informacje na temat parametru *BackoutCount*zawiera sekcja BackoutCount .

Jeśli program wywoła wywołanie MQDISC w czasie, gdy istnieją niezatwierdzone żądania, występuje niejawny punkt synchronizacji, z wyjątkiem zadania wsadowego systemu z/OS z usługą RRS. Jeśli program zakończy działanie nieprawidłowo, nastąpi niejawne wycofanie.

z/OS W systemie z/OSniejawny punkt synchronizacji występuje również wtedy, gdy program kończy działanie normalnie bez uprzedniego wywołania MQDISC. Program zostanie uznany za zakończony normalnie, jeśli baza TCB połączona z produktem MQ zostanie zakończona normalnie. W przypadku działania w środowisku z/OS UNIX System Services i środowisku językowym (LE) domyślna obsługa warunków jest wywoływana w przypadku nieprawidłowego zakończenia lub sygnałów. Procedury obsługi warunku LE przetwarzają warunek błędu, a baza TCB kończy się normalnie. W tych warunkach produkt MQ zatwierdza jednostkę pracy. Więcej informacji na ten temat zawiera sekcja Wprowadzenie do obsługi warunków środowiska językowego.

z/OS W przypadku programów IBM MQ for z/OS można użyć opcji MQGMO_MARK_SKIP_BACKOUT, aby określić, że komunikat nie może zostać wycofany w przypadku wystąpienia wycofania (w celu uniknięcia pętli *MQGET-error-backout* ). Więcej informacji na temat korzystania z tej opcji zawiera sekcja “Pomijanie wycofywania” na stronie 824.

Na zmiany atrybutów kolejki (za pomocą wywołania MQSET lub komend) nie ma wpływu zatwierdzanie lub wycofywanie jednostek pracy.

## z/OS *Punkty synchronizacji w aplikacjach IBM MQ for z/OS*

W tym temacie opisano sposób używania punktów synchronizacji w menedżerze transakcji ( CICS i IMS ) i aplikacji wsadowych.

### z/OS *Punkty synchronizacji w aplikacjach produktu CICS Transaction Server for z/OS*

W aplikacji CICS punkt synchronizacji jest ustanawiany za pomocą komendy EXEC CICS SYNCPOINT.

Aby wycofać wszystkie zmiany wprowadzone w poprzednim punkcie synchronizacji, można użyć komendy EXEC CICS SYNCPOINT ROLLBACK. Więcej informacji na ten temat zawiera publikacja *CICS Application Programming Reference*.

Jeśli w jednostce pracy uczestniczą inne zasoby odtwarzalne, menedżer kolejek (w połączeniu z menedżerem punktów synchronizacji CICS ) uczestniczy w protokole zatwierdzania dwufazowego. W przeciwnym razie menedżer kolejek wykonuje proces zatwierdzania jednofazowego.

Jeśli aplikacja CICS wysyła wywołanie MQDISC, nie jest używany żaden niejawny punkt synchronizacji. Jeśli aplikacja zostanie zamknięta normalnie, wszystkie otwarte kolejki zostaną zamknięte i nastąpi niejawne zatwierdzenie. Jeśli aplikacja zostanie zamknięta nieprawidłowo, wszystkie otwarte kolejki zostaną zamknięte i nastąpi niejawne wycofanie.

z/OS *Punkty synchronizacji w aplikacjach IMS*

W aplikacji IMS należy ustanowić punkt synchronizacji przy użyciu wywołań IMS , takich jak GU (get unique) do IOPCB i CHKP (checkpoint).

Aby wycofać wszystkie zmiany od poprzedniego punktu kontrolnego, można użyć wywołania IMS ROLB (wycofanie zmian). Więcej informacji na ten temat zawiera dokumentacja produktu IMS .

Menedżer kolejek (w połączeniu z menedżerem punktów synchronizacji IMS ) uczestniczy w protokole zatwierdzania dwufazowego, jeśli w jednostce pracy uczestniczą także inne zasoby odtwarzalne.

Wszystkie otwarte uchwyty są zamykane przez adapter IMS w punkcie synchronizacji (z wyjątkiem środowiska BMP sterowanego wsadowo lub niesterowanego komunikatami). Jest to spowodowane tym, że inny użytkownik może zainicjować następną jednostkę pracy, a sprawdzanie zabezpieczeń produktu IBM MQ jest wykonywane podczas wykonywania wywołań MQCONN, MQCONNX i MQOPEN, a nie podczas wykonywania wywołań MQPUT lub MQGET.

Jednak w środowisku typu Wait-for-Input (WFI) lub pseudo-Wait-for-Input (PWFI) IMS nie powiadamia programu IBM MQ o zamknięciu uchwytów do momentu pojawienia się następnego komunikatu lub zwrócenia do aplikacji kodu statusu QC. Jeśli aplikacja oczekuje w regionie IMS i dowolny z tych uchwytów należy do kolejek wyzwalanych, wyzwalanie nie nastąpi, ponieważ kolejki są otwarte. Z tego powodu aplikacje działające w środowisku WFI lub PWFI powinny jawnie MQCLOSE uchwyty kolejki przed wykonaniem operacji GU na IOPCB dla następnego komunikatu.

Jeśli aplikacja IMS (BMP lub MPP) wysyła wywołanie MQDISC, otwarte kolejki są zamykane, ale nie jest pobierany niejawny punkt synchronizacji. Jeśli aplikacja zostanie zamknięta normalnie, wszystkie otwarte kolejki zostaną zamknięte i nastąpi niejawne zatwierdzenie. Jeśli aplikacja zostanie zamknięta nieprawidłowo, wszystkie otwarte kolejki zostaną zamknięte i nastąpi niejawne wycofanie.

z/OS *Punkty synchronizacji w aplikacjach wsadowych systemu z/OS*

W przypadku aplikacji wsadowych można użyć wywołań zarządzania punktem synchronizacji systemu IBM MQ : MQCMIT i MQBACK. W celu zachowania zgodności z wcześniejszymi wersjami, CSQBCMT i CSQBBAK są dostępne jako synonimy.

**Uwaga:** Jeśli konieczne jest zatwierdzenie lub wycofanie aktualizacji zasobów zarządzanych przez różne menedżery zasobów, takie jak IBM MQ i Db2, w ramach pojedynczej jednostki pracy można użyć usługi RRS. Więcej informacji na ten temat zawiera sekcja “Usługi zarządzania transakcjami i odtwarzalnego menedżera zasobów” na stronie 884.

## Zatwierdzanie zmian za pomocą wywołania MQCMIT

Jako dane wejściowe należy podać uchwyt połączenia (*Hconn*), który jest zwracany przez wywołanie MQCONN lub MQCONNX.

Dane wyjściowe MQCMIT są kodem zakończenia i kodem przyczyny. Wywołanie zostanie zakończone z ostrzeżeniem, jeśli punkt synchronizacji został zakończony, ale menedżer kolejek wycofał operacje umieszczania i pobierania od poprzedniego punktu synchronizacji.

Pomyślne zakończenie wywołania MQCMIT wskazuje menedżerowi kolejek, że aplikacja osiągnęła punkt synchronizacji i że wszystkie operacje umieszczania i pobierania wykonane od poprzedniego punktu synchronizacji zostały utrwalone.

Nie wszystkie odpowiedzi o niepowodzeniu oznaczają, że komenda MQCMIT nie została zakończona. Na przykład aplikacja może odebrać komunikat MQRC_CONNECTION_BROKEN.

Opis wywołania MQCMIT znajduje się w sekcji MQCMIT.

## Wycofywanie zmian przy użyciu wywołania MQBACK

Jako dane wejściowe należy podać uchwyt połączenia (*Hconn*). Użyj uchwytu zwróconego przez wywołanie MQCONN lub MQCONNX.

Dane wyjściowe komendy MQBACK są kodem zakończenia i kodem przyczyny.

Dane wyjściowe wskazują menedżerowi kolejek, że aplikacja osiągnęła punkt synchronizacji oraz że wszystkie operacje pobierania i umieszczania, które zostały wykonane od ostatniego punktu synchronizacji, zostały wycofane.

Opis wywołania MQBACK znajduje się w sekcji MQBACK.

## Usługi zarządzania transakcjami i odtwarzalnego menedżera zasobów

Zarządzanie transakcjami i odtwarzalne usługi menedżera zasobów (RRS) to narzędzie produktu z/OS umożliwiające obsługę dwufazowego punktu synchronizacji między uczestniczącymi menedżerami zasobów.

Aplikacja może aktualizować odtwarzalne zasoby zarządzane przez różne menedżery zasobów z/OS , takie jak IBM MQ i Db2, a następnie zatwierdzać lub wycofywać te aktualizacje jako pojedynczą jednostkę pracy. Usługa RRS udostępnia niezbędne rejestrowanie statusu jednostki pracy podczas normalnego wykonywania, koordynuje przetwarzanie punktu synchronizacji i udostępnia odpowiednie informacje o statusie jednostki pracy podczas restartowania podsystemu.

Obsługa uczestników IBM MQ for z/OS RRS umożliwia aplikacjom IBM MQ w środowiskach zadań wsadowych, TSO i procedur składowanych Db2 aktualizowanie zarówno zasobów IBM MQ , jak i zasobów innych niżIBM MQ (na przykład Db2 ) w obrębie pojedynczej logicznej jednostki pracy. Informacje na temat obsługi uczestników RRS zawiera publikacja *z/OS MVS Programming: Resource Recovery*.

Aplikacja IBM MQ może używać wywołań MQCMIT i MQBACK lub równoważnych wywołań RRS, SRRCMIT i SRRBACK. Więcej informacji zawiera temat “Adapter zadania wsadowego RRS” na stronie 919.

**Dostępność RSR**

Jeśli usługa RRS nie jest aktywna w systemie z/OS , każde wywołanie funkcji IBM MQ wysłane z programu połączonego z kodem pośredniczącym RRS (CSQBRSTB lub CSQBRRSI) zwraca błąd MQRC_ENVIRONMENT_ERROR.

**Db2Procedury składowane**

Jeśli z produktem RRS używane są procedury składowane Db2 , należy pamiętać o następujących sytuacjach:

- Procedury składowane Db2 używające usługi RRS muszą być zarządzane przez menedżer obciążenia (WLM-managed).
- Jeśli procedura składowana zarządzana przez Db2zawiera wywołania IBM MQ i jest powiązana z kodem pośredniczącym RRS (CSQBRSTB lub CSQBRRSI), wywołanie MQCONN lub MQCONNX zwraca błąd MQRC_ENVIRONMENT_ERROR.
- Jeśli procedura składowana zarządzana przez WLM zawiera wywołania IBM MQ i jest połączona z kodem pośredniczącym innym niż RRS, wywołanie MQCONN lub MQCONNX zwraca błąd MQRC_ENVIRONMENT_ERROR, chyba że jest to pierwsze wywołanie IBM MQ wykonane od momentu uruchomienia przestrzeni adresowej procedury składowanej.

- Jeśli procedura składowana Db2 zawiera wywołania języka IBM MQ i jest połączona z kodem pośredniczącym innym niż RRS, zasoby produktu IBM MQ zaktualizowane w tej procedurze składowanej nie są zatwierdzane do momentu zakończenia przestrzeni adresowej procedury składowanej lub do czasu, gdy kolejna procedura składowana wykona komendę MQCMIT (przy użyciu kodu pośredniczącego IBM MQ Batch/TSO).
- Wiele kopii tej samej procedury składowanej może być wykonywanych współbieżnie w tej samej przestrzeni adresowej. Należy upewnić się, że program jest zakodowany w sposób wielobieżny, jeśli produkt Db2 ma używać pojedynczej kopii procedury składowanej. W przeciwnym razie w dowolnym wywołaniu IBM MQ w programie może zostać odebrany komunikat MQRC_HCONN_ERROR.
- Nie należy kodować MQCMIT ani MQBACK w procedurze składowanej Db2 zarządzanej przez WLM.
- Zaprojektuj wszystkie programy do uruchamiania w środowisku językowym (Language Environment-LE).

### IBM i *Punkty synchronizacji w produkcie CICS dla aplikacji IBM i*

IBM MQ for IBM i uczestniczy w programie CICS dla IBM i jednostek pracy. Można użyć interfejsu MQI w aplikacji CICS for IBM i w celu umieszczania i pobierania komunikatów w bieżącej jednostce pracy.

Za pomocą komendy EXEC CICS SYNCPOINT można ustanowić punkt synchronizacji, który obejmuje operacje IBM MQ for IBM i . Aby wycofać wszystkie zmiany do poprzedniego punktu synchronizacji, można użyć komendy EXEC CICS SYNCPOINT ROLLBACK.

Jeśli w aplikacji CICS for IBM i używane są opcje MQPUT, MQPUT1lub MQGET z opcją MQPMO_SYNCPOINT lub MQGMO_SYNCPOINT, nie można wylogować się CICS z systemu IBM i , dopóki IBM MQ for IBM i nie usunie swojej rejestracji jako zasobu zatwierdzania transakcji API. Przed rozłączeniem z menedżerem kolejek zatwierdź lub wycofaj wszystkie oczekujące operacje umieszczania lub pobierania. Dzięki temu można wylogować się z programu CICS for IBM i.

### Multi *Punkty synchronizacji w produkcie IBM MQ for Multiplatforms*

Obsługa punktu synchronizacji działa na dwóch typach jednostek pracy: lokalnej i globalnej.

*Lokalna* jednostka pracy to jednostka, w której aktualizowane są tylko zasoby menedżera kolejek produktu IBM MQ . W tym przypadku koordynacja punktów synchronizacji jest udostępniana przez sam menedżer kolejek przy użyciu procedury zatwierdzania jednofazowego.

*Globalna* jednostka pracy to jednostka, w której aktualizowane są również zasoby należące do innych menedżerów zasobów, takich jak bazy danych. IBM MQ może koordynować takie jednostki pracy. Mogą być one również koordynowane przez zewnętrznego kontrolera zobowiązań. Na przykład:

- Inny menedżer transakcji
- IBM i Kontroler zatwierdzania transakcji IBM i

Aby zachować pełną integralność, należy użyć procedury zatwierdzania dwufazowego. Zatwierdzanie dwufazowe może być udostępniane przez menedżery transakcji i bazy danych zgodne z interfejsem XA. Na przykład:

- TXSeries
- Baza danych UDB
- IBM i kontroler zatwierdzania transakcji IBM i

ALW Produkty IBM MQ mogą koordynować globalne jednostki pracy za pomocą procesu zatwierdzania dwufazowego.

IBM i Produkt IBM MQ for IBM i może działać jako menedżer zasobów dla globalnych jednostek pracy w środowisku WebSphere Application Server , ale nie może działać jako menedżer transakcji.

## Niejawny punkt synchronizacji

Podczas umieszczania trwałych komunikatów produkt IBM MQ jest zoptymalizowany pod kątem umieszczania trwałych komunikatów w punkcie synchronizacji. Wiele aplikacji umieszczających komunikaty trwałe w tej samej kolejce działa lepiej, jeśli te aplikacje używają punktu synchronizacji. Wynika to z mniejszej rywalizacji o kolejkę, jeśli do umieszczania trwałych komunikatów używany jest punkt synchronizacji.

Produkt **ImplSyncOpenOutput** dodaje niejawny punkt synchronizacji, gdy aplikacje umieszczają trwałe komunikaty poza punktem synchronizacji. Zapewnia to zwiększenie wydajności, bez konieczności poinformowania aplikacji o niejawnym punkcie synchronizacji.

Niejawny punkt synchronizacji zwiększa wydajność tylko wtedy, gdy wiele aplikacji umieszcza dane w kolejce, ponieważ zmniejsza rywalizację o kolejkę. Dlatego parametr **ImplSyncOpenOutput** określa minimalną liczbę aplikacji, które mają otwartą kolejkę dla danych wyjściowych przed dodaniem niejawnego punktu synchronizacji. Domyślna wartość to 2. Oznacza to, że jeśli nie zostanie podany parametr **ImplSyncOpenOutput**, niejawny punkt synchronizacji jest dodawany tylko wtedy, gdy wiele aplikacji umieszcza w kolejce.

Więcej informacji na ten temat zawiera sekcja Parametry strojenia .

*Lokalne jednostki pracy na wielu platformach*
Jednostki pracy, które dotyczą tylko menedżera kolejek, są nazywane *lokalnymi* jednostkami pracy. Koordynacja punktów synchronizacji jest udostępniana przez sam menedżer kolejek (koordynacja wewnętrzna) przy użyciu procesu zatwierdzania jednofazowego.

Aby uruchomić lokalną jednostkę pracy, aplikacja wysyła żądania MQGET, MQPUT lub MQPUT1 , określając odpowiednią opcję punktu synchronizacji. Jednostka pracy jest zatwierdzana za pomocą MQCMIT lub wycofywana za pomocą MQBACK. Jednak jednostka pracy kończy się również wtedy, gdy połączenie między aplikacją i menedżerem kolejek zostanie zerwane, umyślnie lub nieumyślnie.

Jeśli aplikacja rozłączy się (MQDISC) z menedżerem kolejek, gdy globalna jednostka pracy koordynowana przez produkt IBM MQ jest nadal aktywna, podejmowana jest próba zatwierdzenia jednostki pracy. Jeśli jednak aplikacja zakończy działanie bez rozłączenia, jednostka pracy zostanie wycofana, ponieważ zostanie uznana za zakończoną nieprawidłowo.

*Globalne jednostki pracy na wielu platformach*
Globalnych jednostek pracy należy używać, gdy konieczne jest również uwzględnienie aktualizacji zasobów należących do innych menedżerów zasobów.

W tym przypadku koordynacja może być wewnętrzna lub zewnętrzna w stosunku do menedżera kolejek:

## Wewnętrzna koordynacja punktów synchronizacji

**Koordynacja globalnych jednostek pracy menedżera kolejek nie jest obsługiwana przez IBM MQ for IBM i ani IBM MQ for z/OS. Nie jest on obsługiwany w IBM MQ MQI client środowisku**.

W tym przypadku koordynacja jest zapewniona przez IBM MQ . Aby uruchomić globalną jednostkę pracy, aplikacja wysyła wywołanie MQBEGIN.

Jako dane wejściowe wywołania MQBEGIN należy podać uchwyt połączenia (*Hconn*), który jest zwracany przez wywołanie MQCONN lub MQCONNX. Ten uchwyt reprezentuje połączenie z menedżerem kolejek produktu IBM MQ .

Aplikacja wysyła żądania MQGET, MQPUT lub MQPUT1 , określając odpowiednią opcję punktu synchronizacji. Oznacza to, że można użyć komendy MQBEGIN do zainicjowania globalnej jednostki pracy, która aktualizuje zasoby lokalne i/lub zasoby należące do innych menedżerów zasobów. Aktualizacje zasobów należących do innych menedżerów zasobów są wykonywane przy użyciu interfejsu API tego menedżera zasobów. Nie można jednak używać interfejsu MQI do aktualizowania kolejek należących do innych menedżerów kolejek. Przed rozpoczęciem dalszych jednostek pracy (lokalnej lub globalnej) wydaj komendę MQCMIT lub MQBACK.

Globalna jednostka pracy jest zatwierdzana przy użyciu MQCMIT; inicjuje to dwufazowe zatwierdzanie wszystkich menedżerów zasobów uczestniczących w tej jednostce pracy. Używany jest proces zatwierdzania dwufazowego, w którym menedżery zasobów (na przykład menedżery baz danych zgodne z interfejsem XA, takie jak Db2, Oraclei Sybase) są najpierw proszone o przygotowanie do zatwierdzenia. Tylko jeśli wszyscy są przygotowani, proszeni są o zatwierdzenie. Jeśli jakiś menedżer zasobów zasygnalizuje, że nie może zatwierdzić, każdy z nich jest proszony o cofanie się. Alternatywnie można użyć komendy MQBACK, aby wycofać aktualizacje wszystkich menedżerów zasobów.

Jeśli aplikacja rozłącza się (MQDISC), gdy globalna jednostka pracy jest nadal aktywna, jednostka pracy jest zatwierdzana. Jeśli jednak aplikacja zakończy działanie bez rozłączenia, jednostka pracy zostanie wycofana, ponieważ zostanie uznana za zakończoną nieprawidłowo.

Dane wyjściowe komendy MQBEGIN są kodem zakończenia i kodem przyczyny.

Jeśli komenda MQBEGIN jest używana do uruchamiania globalnej jednostki pracy, uwzględniane są wszystkie zewnętrzne menedżery zasobów, które zostały skonfigurowane z menedżerem kolejek. Jednak wywołanie rozpoczyna jednostkę pracy, ale kończy się z ostrzeżeniem, jeśli:

- Nie ma uczestniczących menedżerów zasobów (to znaczy nie skonfigurowano menedżerów zasobów z menedżerem kolejek)

lub wersji

- Co najmniej jeden menedżer zasobów jest niedostępny.

W takich przypadkach jednostka pracy musi zawierać aktualizacje tylko tych menedżerów zasobów, które były dostępne podczas uruchamiania jednostki pracy.

Jeśli jeden z menedżerów zasobów nie może zatwierdzić swoich aktualizacji, wszystkie menedżery zasobów zostaną poinstruowane, aby wycofać swoje aktualizacje, a program MQCMIT zakończy działanie z ostrzeżeniem. W nietypowych okolicznościach (zazwyczaj jest to interwencja operatora) wywołanie MQCMIT może zakończyć się niepowodzeniem, jeśli niektóre menedżery zasobów zatwierdzą swoje aktualizacje, ale inne wycofają je. Praca zostanie uznana za zakończoną z wynikiem *mieszanym* . Takie wystąpienia są diagnozowane w dzienniku błędów menedżera kolejek, aby można było podjąć działanie naprawcze.

Operacja MQCMIT globalnej jednostki pracy powiedzie się, jeśli wszystkie menedżery zasobów zatwierdzą swoje aktualizacje.

Opis wywołania komendy MQBEGIN zawiera sekcja MQBEGIN.

## Koordynacja zewnętrznego punktu synchronizacji

Dzieje się tak, gdy wybrano koordynatora punktu synchronizacji innego niż IBM MQ , na przykład CICS, Encina lub Tuxedo.

W takiej sytuacji systemy IBM MQ for AIX, Linux, and Windows rejestrują swoje zainteresowanie wynikiem jednostki pracy z koordynatorem punktu synchronizacji, aby mogły zatwierdzać lub wycofywać wszystkie niezatwierdzone operacje pobierania lub umieszczania. Zewnętrzny koordynator punktu synchronizacji określa, czy udostępniane są protokoły zatwierdzania jednofazowego, czy dwufazowego.

Jeśli używany jest zewnętrzny koordynator, nie można wydać komend MQCMIT, MQBACK i MQBEGIN. Wywołania tych funkcji kończą się niepowodzeniem z kodem przyczyny MQRC_ENVIRONMENT_ERROR.

Sposób uruchamiania zewnętrznie koordynowanej jednostki pracy zależy od interfejsu programistycznego udostępnianego przez koordynatora punktu synchronizacji. Może być wymagane jawne wywołanie. Jeśli wymagane jest jawne wywołanie i zostanie uruchomione wywołanie MQPUT z opcją MQPMO_SYNCPOINT, gdy jednostka pracy nie jest uruchomiona, zwracany jest kod zakończenia MQRC_SYNCPOINT_NOT_AVAILABLE.

Zasięg jednostki pracy jest określany przez koordynatora punktu synchronizacji. Stan połączenia między aplikacją i menedżerem kolejek ma wpływ na powodzenie lub niepowodzenie wywołań MQI wywołanych przez aplikację, a nie na stan jednostki pracy. Aplikacja może na przykład rozłączyć i ponownie nawiązać połączenie z menedżerem kolejek podczas aktywnej jednostki pracy i wykonać dalsze operacje MQGET i MQPUT w obrębie tej samej jednostki pracy. Jest to nazywane oczekującym rozłączeniem.

Wywołań funkcji API IBM MQ można używać w programach CICS , niezależnie od tego, czy mają być używane funkcje interfejsu XA produktu CICS. Jeśli interfejs XA nie jest używany, operacje umieszczania i pobierania komunikatów do i z kolejek nie będą zarządzane w ramach niepodzielnych jednostek pracy systemu CICS . Jedną z przyczyn wyboru tej metody jest to, że ogólna spójność jednostki pracy nie jest ważna dla użytkownika.

Jeśli integralność jednostek pracy jest ważna, należy użyć interfejsu XA. W przypadku korzystania z interfejsu XA produkt CICS używa protokołu zatwierdzania dwufazowego, aby zapewnić, że wszystkie zasoby w jednostce pracy są aktualizowane razem.

Więcej informacji na temat konfigurowania obsługi transakcyjnej zawiera sekcja Scenariusze obsługi transakcyjnej, a także dokumentacja produktu TXSeries CICS , na przykład *TXSeries for Multiplatforms CICS Administration Guide for Open Systems*.

Multi *Niejawny punkt synchronizacji na wielu platformach*

Obsługa niejawnego punktu synchronizacji włącza trwałe umieszczanie komunikatów poza punktem synchronizacji.

Podczas umieszczania trwałych komunikatów produkt IBM MQ jest zoptymalizowany pod kątem umieszczania trwałych komunikatów w punkcie synchronizacji. Wiele aplikacji jednocześnie umieszczających komunikaty trwałe w tej samej kolejce zwykle działa lepiej, jeśli te aplikacje używają punktu synchronizacji. Jest to spowodowane tym, że strategia blokowania IBM MQ jest bardziej wydajna, jeśli punkt synchronizacji jest używany podczas umieszczania trwałych komunikatów.

Parametr **ImplSyncOpenOutput** w pliku `qm.ini` określa, czy można dodać niejawny punkt synchronizacji, gdy aplikacje umieszczają trwałe komunikaty poza punktem synchronizacji. Może to spowodować zwiększenie wydajności, bez konieczności poinformowania aplikacji o niejawnym punkcie synchronizacji.

Niejawny punkt synchronizacji zwiększa wydajność tylko wtedy, gdy wiele aplikacji jednocześnie umieszcza dane w kolejce, ponieważ zmniejsza rywalizację o blokady. Parametr **ImplSyncOpenOutput** określa minimalną liczbę aplikacji, które mają otwartą kolejkę dla danych wyjściowych przed dodaniem niejawnego punktu synchronizacji. Wartością domyślną jest *2*. Oznacza to, że jeśli nie zostanie jawnie określona opcja **ImplSyncOpenOutput**, niejawny punkt synchronizacji jest dodawany tylko wtedy, gdy wiele aplikacji umieszcza w kolejce.

Jeśli zostanie dodany niejawny punkt synchronizacji, statystyki odzwierciedlają ten stan i mogą zostać wyświetlone dane wyjściowe transakcji z produktu **runmqsc display conn**.

Aby nigdy nie dodawać niejawnego punktu synchronizacji, należy ustawić wartość **ImplSyncOpenOutput**=*OFF* .

Więcej informacji na ten temat zawiera sekcja Parametry strojenia .

*Interfejsy do zewnętrznych menedżerów punktów synchronizacji na wielu platformach*

Produkt IBM MQ for Multiplatforms obsługuje koordynację transakcji przez zewnętrzne menedżery punktów synchronizacji, które używają interfejsu XA X/Open.

Niektóre menedżery transakcji XA (TXSeries) wymagają, aby każdy menedżer zasobów XA dostarczał swoją nazwę. Jest to łańcuch o nazwie `name` w strukturze przełącznika XA.

- ALW Menedżer zasobów dla produktu IBM MQ w systemie AIX, Linux, and Windows nosi nazwę MQSeries_XA_RMI.
- IBM i W systemie IBM inazwą menedżera zasobów jest MQSeries XA RMI.

Więcej informacji na temat interfejsów XA można znaleźć w dokumentacji interfejsu XA *CAE Specification Distributed Transaction Processing: The XA Specification*(Przetwarzanie rozproszonej transakcji specyfikacji CAE: specyfikacja interfejsu XA) opublikowanej przez grupę otwartą (Open Group).

W konfiguracji XA produkt IBM MQ for Multiplatforms spełnia rolę menedżera zasobów XA. Koordynator punktu synchronizacji XA może zarządzać zestawem menedżerów zasobów XA i synchronizować

zatwierdzanie lub wycofywanie transakcji w obu menedżerach zasobów. Tak to działa w przypadku statycznie zarejestrowanego menedżera zasobów:

1. Aplikacja powiadamia koordynatora punktu synchronizacji, że chce uruchomić transakcję.
2. Koordynator punktu synchronizacji wysyła wywołanie do menedżerów zasobów, o których wie, w celu powiadomienia ich o bieżącej transakcji.
3. Aplikacja wywołuje wywołania w celu zaktualizowania zasobów zarządzanych przez menedżery zasobów powiązane z bieżącą transakcją.
4. Aplikacja żąda od koordynatora punktu synchronizacji zatwierdzenia lub wycofania transakcji.
5. Koordynator punktu synchronizacji wysyła wywołania do każdego menedżera zasobów przy użyciu protokołów zatwierdzania dwufazowego w celu zakończenia transakcji zgodnie z żądaniem.

Specyfikacja XA wymaga, aby każdy menedżer zasobów udostępnił strukturę zwaną przełącznikiem XA. Ta struktura deklaruje możliwości menedżera zasobów oraz funkcje, które mają być wywoływane przez koordynatora punktu synchronizacji.

Istnieją dwie wersje tej struktury:

*Tabela 128. Wersje przełącznika XA*

| **Wersja** | **Opis** |
|---|---|
| Komenda MQRMIXASwitch | Zarządzanie statycznymi zasobami XA |
| MQRMIXASwitchDynamic | Dynamiczne zarządzanie zasobami XA |

Listę bibliotek zawierających tę strukturę zawiera sekcja IBM MQ Struktura przełącznika XA.

Metoda, której należy użyć do połączenia ich z koordynatorem punktu synchronizacji XA, jest zdefiniowana przez koordynatora. Informacje na temat włączania współpracy produktu IBM MQ z koordynatorem punktu synchronizacji XA można znaleźć w dokumentacji tego koordynatora.

Struktura *xa_info* przekazywana w dowolnym wywołaniu *xa_open* przez koordynatora punktu synchronizacji może być nazwą menedżera kolejek, który ma być administrowany. Ma ona taką samą postać jak nazwa menedżera kolejek przekazana do MQCONN lub MQCONNX i może być pusta, jeśli ma być używany domyślny menedżer kolejek. Można jednak użyć dwóch dodatkowych parametrów TPM i AXLIB

TPM umożliwia określenie IBM MQ nazwy menedżera transakcji, na przykład CICS. AXLIB umożliwia określenie rzeczywistej nazwy biblioteki w menedżerze transakcji, w którym znajdują się punkty wejścia XA AX.

Jeśli używany jest jeden z tych parametrów lub inny niż domyślny menedżer kolejek, należy określić nazwę menedżera kolejek za pomocą parametru QMNAME. Więcej informacji na ten temat zawiera sekcja Parametry CHANNEL, TRPTYPE, CONNAME i QMNAME łańcucha xa_open.

## Ograniczenia

1. Globalne jednostki pracy nie są dozwolone ze współużytkowaną Hconn (zgodnie z opisem w sekcji “Współużytkowane (niezależne od wątku) połączenia z MQCONNX” na stronie 762).
2. IBM i Produkt IBM MQ for IBM i nie obsługuje dynamicznej rejestracji menedżerów zasobów XA.

   Jedynym obsługiwanym menedżerem transakcji jest WebSphere Application Server.
3. Windows W systemach Windows wszystkie funkcje zadeklarowane w przełączniku XA są zadeklarowane jako funkcje _cdecl.
4. Zewnętrzny koordynator punktu synchronizacji może jednocześnie administrować tylko jednym menedżerem kolejek. Wynika to z faktu, że koordynator ma efektywne połączenie z każdym menedżerem kolejek i dlatego podlega regule, że w danym momencie dozwolone jest tylko jedno połączenie.

**Uwaga:** Uwaga: Aplikacja kliencka JMS (aplikacja CLIENT JEE ) działająca na serwerze JEE nie ma tego ograniczenia, dlatego pojedyncza transakcja zarządzana przez serwer JEE może koordynować wiele menedżerów kolejek w tej samej transakcji. Jednak aplikacja serwera JMS działająca w trybie powiązań nadal podlega regule, że w danym momencie dozwolone jest tylko jedno połączenie.

5. Wszystkie aplikacje uruchomione przy użyciu koordynatora punktu synchronizacji mogą nawiązać połączenie tylko z menedżerem kolejek, który jest administrowany przez koordynatora, ponieważ są już efektywnie połączone z tym menedżerem kolejek. Muszą one wydać komendę MQCONN lub MQCONNX, aby uzyskać uchwyt połączenia, i muszą wydać komendę MQDISC przed wyjściem. Alternatywnie można użyć wyjścia UE014015 dla produktu TXSeries CICS.

### IBM i *Interfejsy do zewnętrznego menedżera punktów synchronizacji IBM i*

Produkt IBM MQ for IBM i może używać rodzimej kontroli transakcji IBM i jako zewnętrznego koordynatora punktu synchronizacji.

Połączenia niezależne od wątku (współużytkowane) nie są dozwolone z kontrolą transakcji. Więcej informacji na temat możliwości kontroli transakcji systemu IBM izawiera podręcznik *IBM i Programming: Backup and Recovery Guide, SC21-8079* .

Aby uruchomić narzędzia kontroli transakcji systemu IBM i , należy użyć komendy systemowej STRCMTCTL. Aby zakończyć kontrolę transakcji, należy użyć komendy systemowej ENDCMTCTL.

**Uwaga:** Wartością domyślną parametru *Zasięg definicji kontroli transakcji* jest *ACTGRP. Musi być ona zdefiniowana jako *JOB dla systemu IBM MQ for IBM i. Na przykład:

```
STRCMTCTL LCKLVL(*ALL) CMTSCOPE(*JOB)
```

IBM MQ for IBM i może również wykonywać lokalne jednostki pracy zawierające tylko aktualizacje zasobów IBM MQ . Wybór między lokalnymi jednostkami pracy i udziałem w globalnych jednostkach pracy koordynowanych przez produkt IBM i jest dokonywany w każdej aplikacji, gdy aplikacja wywołuje wywołania MQPUT, MQPUT1lub MQGET, podając opcję MQPMO_SYNCPOINT lub MQGMO_SYNCPOINT albo MQBEGIN. Jeśli kontrola transakcji nie jest aktywna podczas pierwszego wywołania, program IBM MQ uruchamia lokalną jednostkę pracy, a wszystkie dalsze jednostki pracy dla tego połączenia z programem IBM MQ również używają lokalnych jednostek pracy, niezależnie od tego, czy kontrola transakcji została uruchomiona. Aby zatwierdzić lokalną jednostkę pracy, należy użyć MQCMIT. Aby wycofać lokalną jednostkę pracy, należy użyć komendy MQBACK. Wywołania zatwierdzenia i wycofania IBM i , takie jak komenda języka CL COMMIT, nie mają wpływu na lokalne jednostki pracy systemu IBM MQ .

Jeśli produkt IBM MQ for IBM i ma być używany z rodzimą kontrolą transakcji IBM i jako zewnętrzny koordynator punktu synchronizacji, należy upewnić się, że wszystkie zadania z kontrolą transakcji są aktywne i że produkt IBM MQ jest używany w zadaniu jednowątkowym. W przypadku wywołania MQPUT, MQPUT1lub MQGET z podaniem MQPMO_SYNCPOINT lub MQGMO_SYNCPOINT w zadaniu wielowątkowym, w którym uruchomiono kontrolę transakcji, wywołanie kończy się niepowodzeniem z kodem przyczyny MQRC_SYNCPOINT_NOT_AVAILABLE.

W zadaniu wielowątkowym można używać lokalnych jednostek pracy oraz wywołań MQCMIT i MQBACK.

W przypadku wywołania MQPUT, MQPUT1lub MQGET z określeniem MQPMO_SYNCPOINT lub MQGMO_SYNCPOINT, po uruchomieniu kontroli transakcji program IBM MQ for IBM i dodaje siebie jako zasób kontroli transakcji API do definicji kontroli transakcji. Jest to zazwyczaj pierwsze takie wywołanie w zadaniu. Chociaż istnieją jakiekolwiek zasoby kontroli transakcji API zarejestrowane w ramach określonej definicji kontroli transakcji, nie można zakończyć kontroli transakcji dla tej definicji.

Produkt IBM MQ for IBM i usuwa swoją rejestrację jako zasób zatwierdzania transakcji API po rozłączeniu z menedżerem kolejek, jeśli w bieżącej jednostce pracy nie ma oczekujących operacji MQI.

W przypadku rozłączenia z menedżerem kolejek, gdy w bieżącej jednostce pracy istnieją oczekujące operacje MQPUT, MQPUT1lub MQGET, produkt IBM MQ for IBM i pozostaje zarejestrowany jako zasób zatwierdzania API, dzięki czemu jest powiadamiany o następnym zatwierdzeniu lub wycofaniu zmian. Po osiągnięciu następnego punktu synchronizacji program IBM MQ for IBM i zatwierdza lub wycofuje zmiany zgodnie z wymaganiami. Aplikacja może rozłączyć się i ponownie nawiązać połączenie z menedżerem

kolejek podczas aktywnej jednostki pracy oraz wykonywać dalsze operacje MQGET i MQPUT w obrębie tej samej jednostki pracy (jest to oczekiwanie na rozłączenie).

Jeśli dla tej definicji kontroli transakcji zostanie wydana komenda systemowa ENDCMTCTL, zostanie wyświetlony komunikat CPF8355 , wskazujący, że oczekujące zmiany były aktywne. Komunikat ten pojawia się również w protokole zadania po zakończeniu zadania. Aby tego uniknąć, należy zatwierdzić lub wycofać wszystkie oczekujące operacje IBM MQ for IBM i i rozłączyć się z menedżerem kolejek. Dlatego użycie komend COMMIT lub ROLLBACK przed wykonaniem komendy ENDCMTCTL umożliwia pomyślne zakończenie kontroli transakcji.

Jeśli kontrola transakcji produktu IBM i jest używana jako zewnętrzny koordynator punktu synchronizacji, nie można wywołać wywołań MQCMIT, MQBACK ani MQBEGIN. Wywołania tych funkcji kończą się niepowodzeniem z kodem przyczyny MQRC_ENVIRONMENT_ERROR.

Aby zatwierdzić lub wycofać zmiany (czyli wycofać zmiany) jednostki pracy, należy użyć jednego z języków programowania, który obsługuje kontrolę transakcji. Na przykład:

- Komendy CL: COMMIT i ROLLBACK
- Funkcje programistyczne ILE C: _Rcommit i _Rrollback
- ILE RPG: COMMIT i ROLBK
- COBOL/400: zatwierdzanie i wycofywanie zmian

Jeśli kontrola transakcji IBM i jest używana jako zewnętrzny koordynator punktu synchronizacji z produktem IBM MQ for IBM i, produkt IBM i wykonuje protokół zatwierdzania dwufazowego, w którym uczestniczy produkt IBM MQ . Ponieważ każda jednostka pracy jest zatwierdzana w dwóch fazach, menedżer kolejek może stać się niedostępny dla drugiej fazy po głosowaniu za zatwierdzeniem w pierwszej fazie. Może się tak zdarzyć na przykład wtedy, gdy wewnętrzne zadania menedżera kolejek zostały zakończone. W takiej sytuacji protokół zadania przeprowadzający zatwierdzenie zawiera komunikat CPF835F wskazujący, że operacja zatwierdzania lub wycofywania zmian nie powiodła się. Komunikaty poprzedzające ten komunikat wskazują przyczynę problemu, niezależnie od tego, czy wystąpił on podczas operacji zatwierdzania lub wycofywania zmian, a także identyfikator logicznej jednostki pracy (LUWID) dla uszkodzonej jednostki pracy.

Jeśli problem został spowodowany przez awarię zasobu zatwierdzania transakcji API IBM MQ podczas zatwierdzania lub wycofywania przygotowanej jednostki pracy, można użyć komendy WRKMQMTRN do zakończenia operacji i przywrócenia integralności transakcji. Komenda wymaga, aby użytkownik znał identyfikator LUWID jednostki pracy, która ma zostać zatwierdzona i wycofany.

## Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy

Informacje o wyzwalaczach i sposobie uruchamiania aplikacji IBM MQ za pomocą wyzwalaczy.

Niektóre aplikacje IBM MQ , które udostępniają kolejki, działają w sposób ciągły, dlatego są zawsze dostępne do pobierania komunikatów przychodzących do kolejek. Nie jest to jednak pożądane, gdy liczba komunikatów przychodzących do kolejek jest nieprzewidywalna. W takim przypadku aplikacje mogą zużywać zasoby systemowe nawet wtedy, gdy nie ma żadnych komunikatów do pobrania.

Produkt IBM MQ udostępnia narzędzie, które umożliwia automatyczne uruchamianie aplikacji, gdy są dostępne komunikaty do pobrania. Ta funkcja jest nazywana *wyzwalaniem*.

Informacje o wyzwalaniu kanałów zawiera sekcja Wyzwalanie kanałów.

### Co to jest wyzwalanie?

Menedżer kolejek definiuje pewne warunki jako *zdarzenia wyzwalające*.

Jeśli wyzwalanie jest włączone dla kolejki i wystąpi zdarzenie wyzwalające, menedżer kolejek wysyła *komunikat wyzwalacza* do kolejki o nazwie *kolejka inicjująca*. Obecność komunikatu wyzwalacza w kolejce inicjującej wskazuje, że wystąpiło zdarzenie wyzwalające.

Komunikaty wyzwalacza wygenerowane przez menedżer kolejek nie są trwałe. Zmniejsza to rejestrowanie (co powoduje zwiększenie wydajności) i minimalizuje duplikaty podczas restartowania, co skraca czas restartowania.

Program, który przetwarza kolejkę inicjującą, jest nazywany *aplikacją monitorującą wyzwalacz*, a jego funkcją jest odczytywanie komunikatu wyzwalacza i podejmowanie odpowiednich działań na podstawie informacji zawartych w komunikacie wyzwalacza. Zwykle to działanie polega na uruchomieniu innej aplikacji w celu przetworzenia kolejki, która wygenerowała komunikat wyzwalacza. Z punktu widzenia menedżera kolejek nie ma nic specjalnego w aplikacji monitora wyzwalacza; jest to po prostu inna aplikacja, która odczytuje komunikaty z kolejki (kolejka inicjująca).

Jeśli wyzwalanie jest włączone dla kolejki, można utworzyć powiązany z nią *obiekt definicji procesu* . Ten obiekt zawiera informacje o aplikacji, która przetwarza komunikat, który spowodował zdarzenie wyzwalające. Jeśli obiekt definicji procesu został utworzony, menedżer kolejek wyodrębnia te informacje i umieszcza je w komunikacie wyzwalacza w celu użycia przez aplikację monitora wyzwalacza. Nazwa definicji procesu powiązanej z kolejką jest nadawana przez atrybut kolejki lokalnej *ProcessName* . Każda kolejka określa inną definicję procesu lub kilka kolejek może współużytkować definicję procesu.

Aby wyzwolić uruchomienie kanału, nie trzeba definiować obiektu definicji procesu. Zamiast niej używana jest definicja kolejki transmisji.

Wyzwalanie jest obsługiwane przez klienty IBM MQ działające w systemie AIX, Linux, and Windows. Aplikacja działająca w środowisku klienckim jest taka sama, jak aplikacja działająca w pełnym środowisku IBM MQ , z tą różnicą, że należy ją połączyć z bibliotekami klienta. Jednak monitor wyzwalacza i aplikacja, która ma zostać uruchomiona, muszą znajdować się w tym samym środowisku.

Wyzwalanie obejmuje:

**Kolejka aplikacji**
*Kolejka aplikacji* jest kolejką lokalną, która po ustawieniu wyzwalania i spełnieniu warunków wymaga zapisania komunikatów wyzwalacza.

**Definicja procesu**
Z kolejką aplikacji może być powiązany *obiekt definicji procesu* , który zawiera szczegóły aplikacji pobierającej komunikaty z kolejki aplikacji. (Listę atrybutów zawiera sekcja Atrybuty definicji procesów ).

**Jeśli wyzwalacz ma uruchamiać kanał, nie trzeba definiować obiektu definicji procesu.**

**Kolejka transmisji**
**Kolejka transmisji jest potrzebna, jeśli do uruchomienia kanału ma zostać uruchomiony wyzwalacz.**

W przypadku kolejki transmisji na dowolnej platformie innej niż Linuxatrybut *TriggerData* kolejki transmisji może określać nazwę kanału, który ma zostać uruchomiony. Może to zastąpić definicję procesu dla kanałów wyzwalających, ale jest używane tylko wtedy, gdy definicja procesu nie została utworzona.

**zdarzenie wyzwalające**
*Zdarzenie wyzwalające* jest zdarzeniem, które powoduje wygenerowanie komunikatu wyzwalacza przez menedżer kolejek. Zwykle jest to komunikat, który pojawia się w kolejce aplikacji, ale może również wystąpić w innym czasie. Na przykład patrz “Warunki dla zdarzenia wyzwalającego” na stronie 898.

IBM MQ udostępnia szereg opcji umożliwiających sterowanie warunkami powodującymi wystąpienie zdarzenia wyzwalającego (patrz sekcja “Kontrolowanie zdarzeń wyzwalających” na stronie 902 ).

**komunikat wyzwalacza**
Po rozpoznaniu zdarzenia wyzwalającego menedżer kolejek tworzy *komunikat wyzwalacza* . Jest on kopiowany do informacji komunikatu wyzwalacza o aplikacji, która ma zostać uruchomiona. Informacje te pochodzą z kolejki aplikacji i obiektu definicji procesu powiązanego z kolejką aplikacji.

Komunikaty wyzwalacza mają stały format (patrz sekcja “Format komunikatów wyzwalacza” na stronie 910 ).

**Kolejka inicjująca**

*Kolejka inicjująca* jest kolejką lokalną, w której menedżer kolejek umieszcza komunikaty wyzwalacza. Należy zauważyć, że kolejka inicjująca nie może być kolejką aliasową ani kolejką modelową.

Menedżer kolejek może być właścicielem więcej niż jednej kolejki inicjującej, a każda z nich jest powiązana z jedną lub większą liczbą kolejek aplikacji.

z/OS Kolejka współużytkowana, kolejka lokalna dostępna dla menedżerów kolejek w grupie współużytkowania kolejek, może być kolejką inicjującą w systemie IBM MQ for z/OS.

**monitor wyzwalacza**

*Monitor wyzwalacza* jest działającym w sposób ciągły programem, który obsługuje jedną lub więcej kolejek inicjujących. Gdy komunikat wyzwalacza przybywa do kolejki inicjującej, monitor wyzwalacza wczytuje ten komunikat. Monitor wyzwalacza używa informacji zawartych w komunikacie wyzwalacza. Uruchamia on komendę uruchamiającą aplikację, która pobiera komunikaty przychodzące do kolejki aplikacji, przekazując informacje zawarte w nagłówku komunikatu wyzwalacza, w tym nazwę kolejki aplikacji.

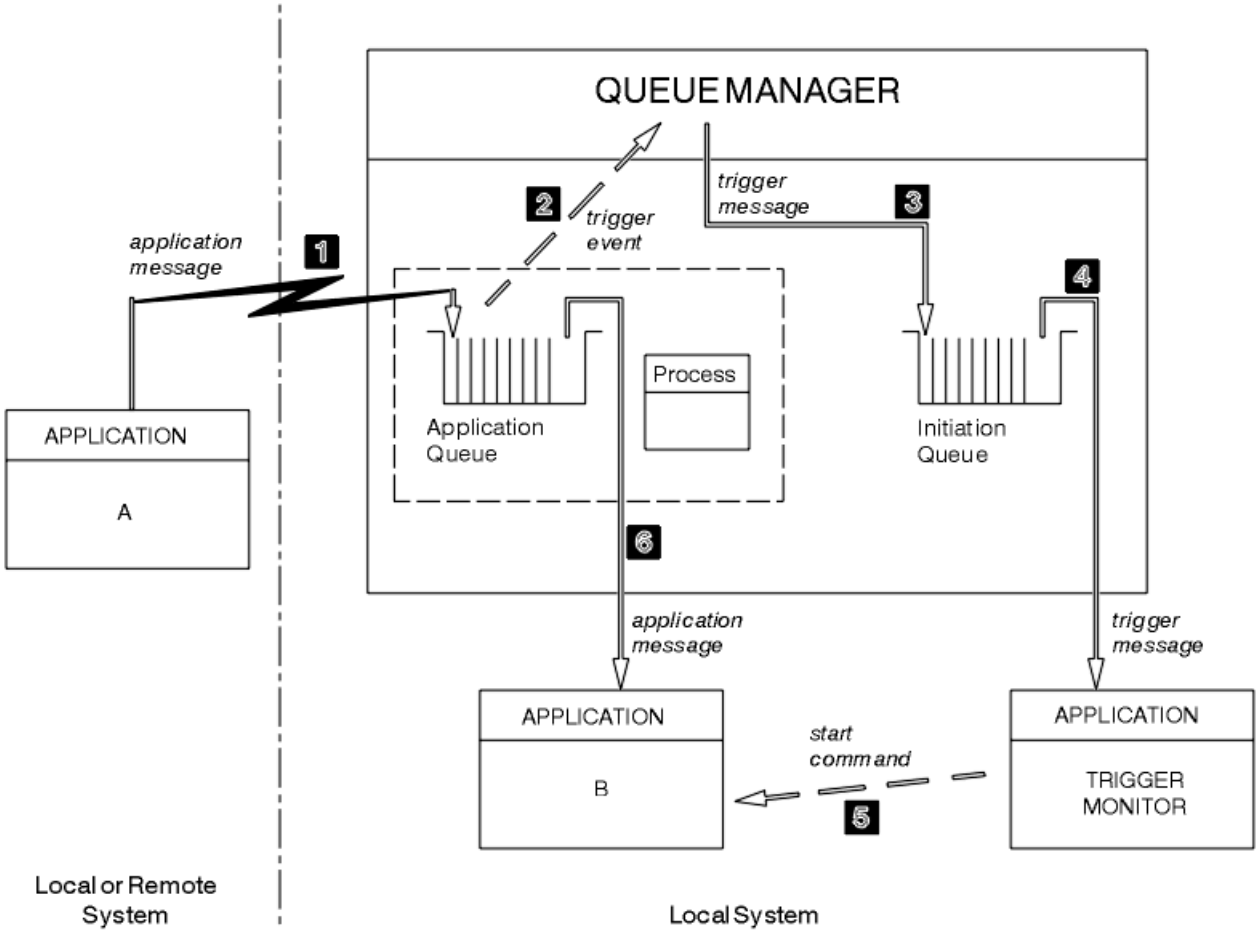
Na wszystkich platformach za uruchamianie kanałów odpowiada specjalny monitor wyzwalacza znany jako inicjator kanału.

z/OS W systemie z/OSinicjator kanału jest zwykle uruchamiany ręcznie lub może zostać uruchomiony automatycznie po uruchomieniu menedżera kolejek przez zmianę wartości CSQINP2 w kodzie JCL uruchamiania menedżera kolejek.

Multi W systemie Wiele platforminicjator kanału jest uruchamiany automatycznie podczas uruchamiania menedżera kolejek lub można go uruchomić ręcznie za pomocą komendy **runmqchi** .

Więcej informacji na ten temat zawiera “Przetwarzanie kolejki inicjującej przez monitory wyzwalacza” na stronie 906.

Aby zrozumieć działanie wyzwalania, należy rozważyć użycie parametru Rysunek 95 na stronie 894, który jest przykładem wyzwalacza typu FIRST (MQTT_FIRST).




*Rysunek 95. Przepływ komunikatów aplikacji i wyzwalacza*

W produkcie Rysunek 95 na stronie 894sekwencja zdarzeń jest następująca:

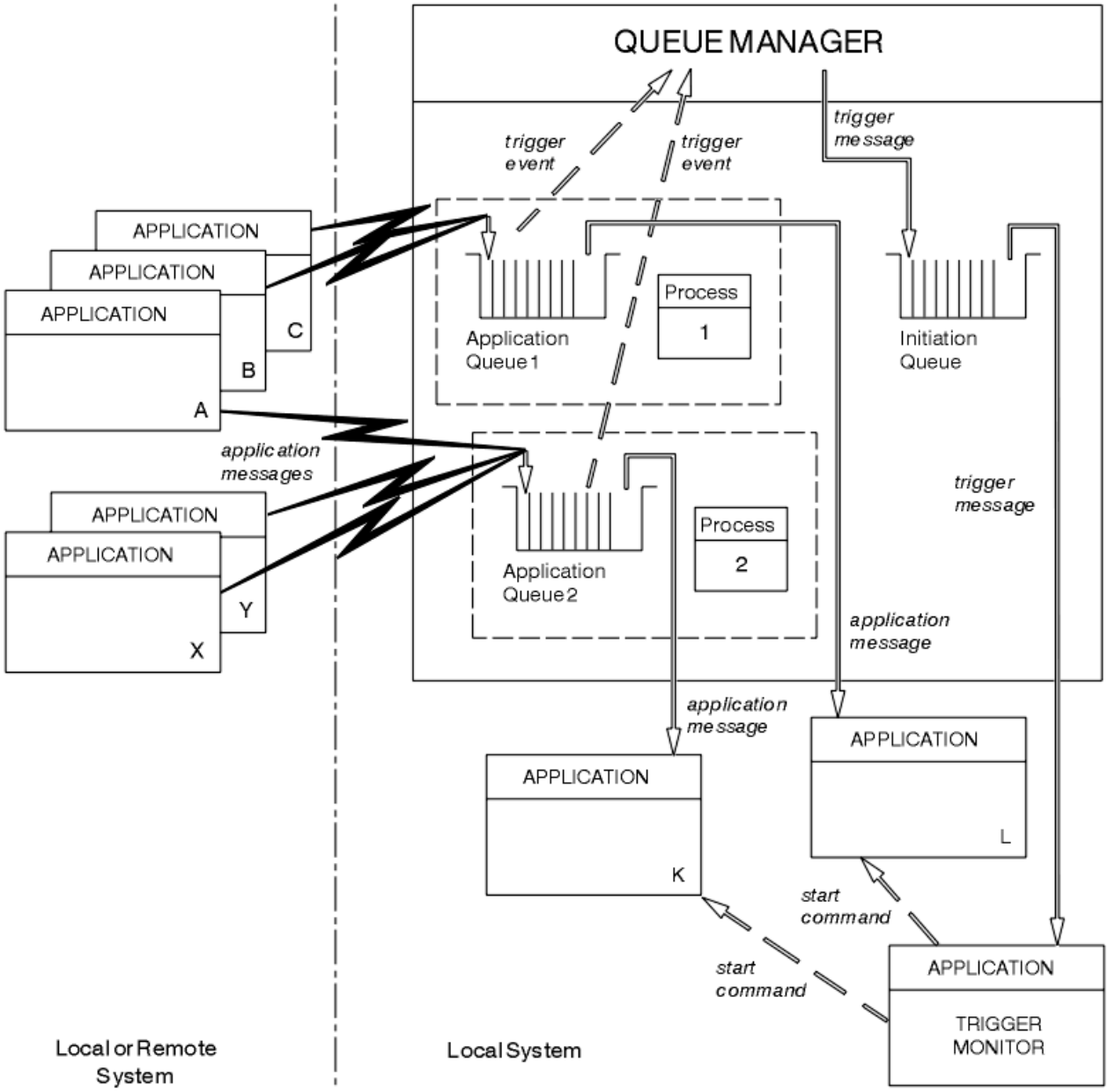
1. Aplikacja A, która może być lokalna lub zdalna względem menedżera kolejek, umieszcza komunikat w kolejce aplikacji. Żadna aplikacja nie ma tej kolejki otwartej do wprowadzania. Jednak ten fakt ma znaczenie tylko w przypadku wyzwalania typu FIRST i DEPTH.
2. Menedżer kolejek sprawdza, czy spełnione są warunki, w których musi wygenerować zdarzenie wyzwalające. Są one i generowane jest zdarzenie wyzwalające. Informacje przechowywane w powiązanym obiekcie definicji procesu są używane podczas tworzenia komunikatu wyzwalacza.
3. Menedżer kolejek tworzy komunikat wyzwalacza i umieszcza go w kolejce inicjującej powiązanej z tą kolejką aplikacji, ale tylko wtedy, gdy aplikacja (monitor wyzwalacza) ma otwartą kolejkę inicjującą dla wejścia.
4. Monitor wyzwalacza pobiera komunikat wyzwalacza z kolejki inicjującej.
5. Monitor wyzwalacza wydaje komendę uruchamiającą aplikację B (aplikację serwera).
6. Aplikacja B otwiera kolejkę aplikacji i pobiera komunikat.

**Uwaga:**

1. Jeśli kolejka aplikacji jest otwarta dla wejścia, przez dowolny program i ma ustawione wyzwalanie dla FIRST lub DEPTH, nie występuje zdarzenie wyzwalające, ponieważ kolejka jest już obsługiwana.
2. Jeśli kolejka inicjująca nie jest otwarta dla danych wejściowych, menedżer kolejek nie generuje żadnych komunikatów wyzwalacza; oczekuje, aż aplikacja otworzy kolejkę inicjującą dla danych wejściowych.
3. Jeśli używane jest wyzwalanie dla kanałów, należy użyć wyzwalacza typu FIRST lub DEPTH.

4. Aplikacje wyzwalane są uruchamiane przy użyciu identyfikatora użytkownika i grupy użytkownika, który uruchomił monitor wyzwalacza, użytkownika CICS lub użytkownika, który uruchomił menedżer kolejek.

Do tej pory relacja między kolejkami w wyzwalaniu była tylko jedna do jednej. Należy rozważyć Rysunek 96 na stronie 895.




*Rysunek 96. Relacja kolejek w wyzwalaniu*

Z kolejką aplikacji jest powiązany obiekt definicji procesu, który zawiera szczegóły aplikacji przetwarzającej komunikat. Menedżer kolejek umieszcza informacje w komunikacie wyzwalacza, dlatego wymagana jest tylko jedna kolejka inicjująca. Monitor wyzwalacza wyodrębnia te informacje z komunikatu wyzwalacza i uruchamia odpowiednią aplikację w celu obsługi komunikatu w każdej kolejce aplikacji.

Należy pamiętać, że aby wyzwolić uruchomienie kanału, nie trzeba definiować obiektu definicji procesu. Definicja kolejki transmisji może określać kanał, który ma zostać wyzwolony.

Aby uzyskać więcej informacji na temat uruchamiania aplikacji IBM MQ za pomocą wyzwalaczy, należy skorzystać z następujących odsyłaczy:

- "Wymagania wstępne dla wyzwalania" na stronie 896

- “Warunki dla zdarzenia wyzwalającego” na stronie 898
- “Kontrolowanie zdarzeń wyzwalających” na stronie 902
- “Projektowanie aplikacji korzystającej z kolejek wyzwalanych” na stronie 905
- “Przetwarzanie kolejki inicjującej przez monitory wyzwalacza” na stronie 906
- “Właściwości komunikatów wyzwalacza” na stronie 909
- “Gdy wyzwalanie nie działa” na stronie 911

**Pojęcia pokrewne**

“Przegląd interfejsu kolejki komunikatów” na stronie 745
Sekcja zawiera informacje na temat komponentów interfejsu kolejki komunikatów (Message Queue Interface-MQI).

“Nawiązywanie i rozłączanie połączenia z menedżerem kolejek” na stronie 758
Aby można było używać usług programistycznych IBM MQ , program musi mieć połączenie z menedżerem kolejek. Ten temat zawiera informacje o nawiązywaniu połączenia z menedżerem kolejek i rozłączaniu się z nim.

“Otwieranie i zamykanie obiektów” na stronie 766
Informacje te udostępniają wgląd w otwieranie i zamykanie obiektów IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 777
Ta sekcja zawiera informacje na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 792
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Uzyskiwanie informacji o atrybutach obiektu i ustawianie ich” na stronie 876
Atrybuty są właściwościami definiującymi charakterystykę obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 879
W tej sekcji opisano sposób zatwierdzania i wycofywania wszelkich odtwarzalnych operacji pobierania i umieszczania, które wystąpiły w jednostce pracy.

“Praca z funkcją MQI i klastrami” na stronie 911
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do łączenia w klastry.

“Używanie i pisanie aplikacji w systemie IBM MQ for z/OS” na stronie 916
Aplikacje IBM MQ for z/OS mogą być tworzone z programów działających w wielu różnych środowiskach. Oznacza to, że mogą korzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje mostu IMS i IMS w systemie IBM MQ for z/OS” na stronie 71
Informacje te są pomocne podczas pisania aplikacji IMS przy użyciu produktu IBM MQ.

## *Wymagania wstępne dla wyzwalania*

Te informacje umożliwiają zapoznanie się z krokami, które należy wykonać przed użyciem wyzwalania.

Aby aplikacja mogła skorzystać z możliwości wyzwalania, wykonaj następujące kroki:

1. Albo:
   a. Utwórz kolejkę inicjującą dla kolejki aplikacji. Na przykład:

```
DEFINE QLOCAL (initiation.queue) REPLACE      +
       LIKE (SYSTEM.DEFAULT.INITIATION.QUEUE)      +
       DESCR ('initiation queue description')
```

   lub wersji

   b. Określ nazwę kolejki lokalnej, która istnieje i może być używana przez aplikację (zwykle jest to SYSTEM.DEFAULT.INITIATION.QUEUE lub, jeśli kanały są uruchamiane z wyzwalaczami, SYSTEM.CHANNEL.INITQ) i podaj jego nazwę w polu *InitiationQName* kolejki aplikacji.

2. Powiąż kolejkę inicjującą z kolejką aplikacji. Menedżer kolejek może być właścicielem więcej niż jednej kolejki inicjującej. Niektóre kolejki aplikacji mogą być obsługiwane przez różne programy. W takim przypadku można użyć jednej kolejki inicjującej dla każdego programu obsługującego, chociaż nie jest to konieczne. Poniżej przedstawiono przykład tworzenia kolejki aplikacji:

```
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 Poniżej przedstawiono fragment programu CL dla systemu IBM MQ for IBM i , który tworzy kolejkę inicjującą:

```
/*   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. Jeśli wyzwalana jest aplikacja, należy utworzyć obiekt definicji procesu, który będzie zawierał informacje dotyczące aplikacji, która ma obsługiwać kolejkę aplikacji. Na przykład, aby wyzwolić uruchom transakcję listy płac CICS o nazwie PAYR:

```
DEFINE PROCESS (process.name) +
       REPLACE +
       DESCR ('process description') +
       APPLICID ('PAYR') +
       APPLTYPE (CICS) +
       USERDATA ('Payroll data')
```

IBM i Poniżej przedstawiono fragment programu CL dla systemu IBM MQ for IBM i , który tworzy obiekt definicji procesu:

```
/*   Process definition                                            */
            CRTMQMPRC   PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                        REPLACE(*YES)                               +
                        MQMNAME                                     +
                        TEXT('trigger process for AMQSINQA')        +
                        ENVDATA('JOBPTY(3)') /* Submit parameter  */+
                        APPID('AMQSINQA')    /* Program name      */
```

Gdy menedżer kolejek tworzy komunikat wyzwalacza, kopiuje informacje z atrybutów obiektu definicji procesu do komunikatu wyzwalacza.

| Platforma | Aby utworzyć obiekt definicji procesu |
|---|---|
| Systemy AIX, Linux, and Windows | Użyj opcji DEFINE PROCESS lub użyj opcji SYSTEM.DEFAULT.PROCESS i modify using ALTER PROCESS |
| z/OS z/OS | Użyj komendy DEFINE PROCESS (patrz przykładowy kod w kroku “3” na stronie 897 ) lub użyj paneli sterowania i operacji. |

| Platforma | Aby utworzyć obiekt definicji procesu |
|---|---|
| IBM i<br>IBM i IBM i | Użyj programu CL zawierającego kod, jak w kroku “3” na stronie 897. |

4. Opcjonalnie: utwórz definicję kolejki transmisji i użyj odstępów dla atrybutu **ProcessName** .

   Atrybut **TrigData** może zawierać nazwę kanału, który ma zostać wyzwolony, lub może pozostać pusty. Z wyjątkiem systemu IBM MQ for z/OS, jeśli pole pozostanie puste, inicjator kanału będzie przeszukiwał pliki definicji kanału, dopóki nie znajdzie kanału powiązanego z nazwaną kolejką transmisji. Gdy menedżer kolejek tworzy komunikat wyzwalacza, kopiuje informacje z atrybutu **TrigData** definicji kolejki transmisji do komunikatu wyzwalacza.

5. Jeśli utworzono obiekt definicji procesu w celu określenia właściwości aplikacji, która ma obsługiwać kolejkę aplikacji, należy powiązać obiekt procesu z kolejką aplikacji, nadając mu nazwę w atrybucie **ProcessName** kolejki.

| Platforma | Użyj komend |
|---|---|
| Systemy AIX, Linux, and Windows | ZMIEŃ QLOCAL |
| z/OS<br>z/OS z/OS | ZMIEŃ QLOCAL |
| IBM i<br>IBM i IBM i | Komenda CHGMQMQ |

6. Uruchom instancje monitorów wyzwalaczy IBM i (lub serwerów wyzwalaczy w produkcie IBM MQ for IBM i ) , które mają obsługiwać zdefiniowane kolejki inicjujące. Więcej informacji zawiera sekcja “Przetwarzanie kolejki inicjującej przez monitory wyzwalacza” na stronie 906.

Aby mieć informacje o niedostarczonych komunikatach wyzwalacza, należy upewnić się, że menedżer kolejek ma zdefiniowaną kolejkę niedostarczonych komunikatów (niedostarczonych komunikatów). Podaj nazwę kolejki w polu *DeadLetterQName* menedżera kolejek.

Następnie można ustawić wymagane warunki wyzwalacza, używając atrybutów obiektu kolejki, który definiuje kolejkę aplikacji. Więcej informacji na ten temat zawiera sekcja “Kontrolowanie zdarzeń wyzwalających” na stronie 902.

### *Warunki dla zdarzenia wyzwalającego*

Menedżer kolejek tworzy komunikat wyzwalacza, jeśli spełnione są warunki opisane w tym temacie.

Odwołania do kolejek współużytkowanych w tym temacie oznaczają kolejki współużytkowane w grupie współużytkowania kolejek, które są dostępne tylko w systemie IBM MQ for z/OS.

Następujące warunki powodują, że menedżer kolejek tworzy komunikat wyzwalacza:

1. Komunikat jest *umieszczany* w kolejce.
2. Komunikat ma priorytet większy lub równy priorytetowi wyzwalacza progu kolejki. Ten priorytet jest ustawiany w atrybucie kolejki lokalnej **TriggerMsgPriority** . Jeśli jest ustawiony na zero, kwalifikuje się każdy komunikat.
3. Liczba komunikatów w kolejce o priorytecie większym lub równym *TriggerMsgPriority* była poprzednio, w zależności od wartości *TriggerType*:
   - Zero (dla wyzwalacza typu MQTT_FIRST)
   - Dowolna liczba (dla wyzwalacza typu MQTT_EVERY)
   - *TriggerDepth* minus 1 (dla wyzwalacza typu MQTT_DEPTH)

**Uwaga:**

a. W przypadku niewspółużytkowanych kolejek lokalnych menedżer kolejek zlicza zarówno zatwierdzone, jak i niezatwierdzone komunikaty podczas oceny, czy istnieją warunki dla zdarzenia wyzwalającego. W związku z tym aplikacja może zostać uruchomiona, gdy nie ma żadnych komunikatów do pobrania, ponieważ komunikaty w kolejce nie zostały zatwierdzone. W takiej sytuacji należy rozważyć użycie opcji wait z odpowiednią wartością *WaitInterval*, aby aplikacja czekała na nadejście komunikatów.

b. W przypadku lokalnych kolejek współużytkowanych menedżer kolejek zlicza tylko zatwierdzone komunikaty.

4. W przypadku wyzwalania typu FIRST lub DEPTH żaden program nie ma otwartej kolejki aplikacji do usuwania komunikatów (atrybut kolejki lokalnej **OpenInputCount** ma wartość zero).

**Uwaga:**

a. W przypadku kolejek współużytkowanych mają zastosowanie specjalne warunki, gdy wiele menedżerów kolejek ma uruchomione monitory wyzwalacza dla kolejki. W takiej sytuacji, jeśli co najmniej jeden menedżer kolejek ma otwartą kolejkę na potrzeby współużytkowania danych wejściowych, kryteria wyzwalacza w innych menedżerach kolejek są traktowane jako *TriggerType* MQTT_FIRST i *TriggerMsgPriority* zero. Gdy wszystkie menedżery kolejek zamkną kolejkę dla wejścia, warunki wyzwalacza zostaną przywrócone do warunków określonych w definicji kolejki.

Przykładowym scenariuszem, którego dotyczy ten warunek, jest wiele menedżerów kolejek QM1, QM2i QM3 z uruchomionym monitorem wyzwalacza dla kolejki aplikacji A. Komunikat dociera do A spełniającego warunki wyzwalania, a w kolejce inicjującej generowany jest komunikat wyzwalacza. Monitor wyzwalacza QM1 pobiera komunikat wyzwalacza i wyzwala aplikację. Wyzwolona aplikacja otwiera kolejkę aplikacji dla wejścia współużytkowanego. Od tego punktu warunki wyzwalacza dla kolejki aplikacji A są wartościowane jako *TriggerType* MQTT_FIRST i *TriggerMsgPriority* zero w menedżerach kolejek QM2 i QM3, dopóki QM1 nie zamknie kolejki aplikacji.

b. W przypadku kolejek współużytkowanych warunek ten jest stosowany dla każdego menedżera kolejek. Oznacza to, że wartość parametru *OpenInputCount* menedżera kolejek dla kolejki musi wynosić zero, aby komunikat wyzwalacza został wygenerowany dla kolejki przez ten menedżer kolejek. Jeśli jednak dowolny menedżer kolejek w grupie współużytkowania kolejek ma otwartą kolejkę przy użyciu opcji MQOO_INPUT_EXCLUSIVE, żaden z menedżerów kolejek w grupie współużytkowania kolejek nie wygeneruje dla tej kolejki komunikatu wyzwalacza.

Zmiana sposobu wartościowania warunków wyzwalacza występuje, gdy wyzwalana aplikacja otwiera kolejkę dla wejścia. W scenariuszach, w których działa tylko jeden monitor wyzwalacza, inne aplikacje mogą mieć taki sam efekt, ponieważ podobnie otwierają kolejkę aplikacji dla danych wejściowych. Nie ma znaczenia, czy kolejka aplikacji została otwarta przez aplikację uruchomioną przez monitor wyzwalacza, czy przez inną aplikację. Jest to fakt, że kolejka jest otwarta do wprowadzania w innym menedżerze kolejek, co powoduje zmianę kryteriów wyzwalacza.

5. W systemie IBM MQ for z/OS, jeśli kolejka aplikacji jest kolejką z atrybutem **Usage** o wartości MQUS_NORMAL, żądania pobrania dla tej kolejki nie są zablokowane (to znaczy, że atrybut kolejki **InhibitGet** ma wartość MQQA_GET_ALLOWED). Ponadto, jeśli wyzwalana kolejka aplikacji ma atrybut **Usage** o wartości MQUS_XMITQ, żądania pobrania dla niej nie są zablokowane.

6. Albo:

- Atrybut kolejki lokalnej **ProcessName** nie jest pusty, a obiekt definicji procesu identyfikowany przez ten atrybut został utworzony lub
- Atrybut kolejki lokalnej **ProcessName** dla kolejki jest pusty, ale kolejka jest kolejką transmisji. Ponieważ definicja procesu jest opcjonalna, atrybut **TriggerData** może również zawierać nazwę kanału, który ma zostać uruchomiony. W takim przypadku komunikat wyzwalacza zawiera atrybuty o następujących wartościach:
  - **QName**: nazwa kolejki
  - **ProcessName**: odstępy

- **TriggerData**: dane wyzwalacza
- **ApplType**: MQAT_UNKNOWN
- **ApplId**: odstępy
- **EnvData**: odstępy
- **UserData**: odstępy

7. Utworzono kolejkę inicjującą i określono ją w atrybucie kolejki lokalnej **InitiationQName** . Ponadto:
   - Żądania pobierania nie są zablokowane dla kolejki inicjującej (wartość atrybutu kolejki **InhibitGet** to MQQA_GET_ALLOWED).
   - Żądania umieszczania nie mogą być zablokowane dla kolejki inicjującej (tzn. wartością atrybutu kolejki **InhibitPut** musi być MQQA_PUT_ALLOWED).
   - Wartością atrybutu **Usage** kolejki inicjującej musi być MQUS_NORMAL.
   - W środowiskach, w których obsługiwane są kolejki dynamiczne, kolejka inicjująca nie może być kolejką dynamiczną, która została oznaczona jako usunięta logicznie.
8. Monitor wyzwalacza ma obecnie otwartą kolejkę inicjującą w celu usunięcia komunikatów (czyli atrybut kolejki lokalnej **OpenInputCount** jest większy od zera).
9. Element sterujący wyzwalacza (atrybut kolejki lokalnej**TriggerControl** ) dla kolejki aplikacji jest ustawiony na wartość MQTC_ON. W tym celu podczas definiowania kolejki należy ustawić atrybut **trigger** lub użyć komendy ALTER QLOCAL.
10. Typem wyzwalacza (atrybut kolejki lokalnej**TriggerType** ) nie jest MQTT_NONE.

    Jeśli wszystkie wymagane warunki są spełnione, a komunikat, który spowodował warunek wyzwalacza, jest umieszczany jako część jednostki pracy, komunikat wyzwalacza nie staje się dostępny do pobrania przez aplikację monitora wyzwalacza do czasu zakończenia jednostki pracy, bez względu na to, czy jednostka pracy została zatwierdzona, czy też dla wyzwalacza typu MQTT_FIRST lub MQTT_DEPTH została wycofana.
11. Odpowiedni komunikat jest umieszczany w kolejce dla parametru **TriggerType** o wartości MQTT_FIRST lub MQTT_DEPTH oraz w kolejce:
    - Poprzednio nie było puste (MQTT_FIRST) lub
    - Wystąpiły komunikaty ( **TriggerDepth** lub więcej) (MQTT_DEPTH)

    i spełnione są warunki od “2” na stronie 898 do “10” na stronie 900 (z wyjątkiem “3” na stronie 898), jeśli w przypadku MQTT_FIRST upłynął wystarczający odstęp czasu (atrybut menedżera kolejek**TriggerInterval** ) od ostatniego zapisania komunikatu wyzwalacza dla tej kolejki.

    Ma to na celu umożliwienie zakończenia działania serwera kolejek przed przetworzeniem wszystkich komunikatów w kolejce. Celem interwału wyzwalacza jest zmniejszenie liczby zduplikowanych komunikatów wyzwalacza, które są generowane.

    **Uwaga:** Jeśli menedżer kolejek zostanie zatrzymany i zrestartowany, licznik czasu **TriggerInterval** zostanie zresetowany. Istnieje małe okno, w którym można wygenerować dwa komunikaty wyzwalacza. Okno istnieje, gdy atrybut wyzwalacza kolejki jest włączony w tym samym czasie, co komunikat, a kolejka nie była wcześniej pusta (MQTT_FIRST) lub zawierała **TriggerDepth** lub więcej komunikatów (MQTT_DEPTH).
12. Jedyna aplikacja obsługująca kolejkę wysyła wywołanie MQCLOSE dla parametru **TriggerType** o wartości MQTT_FIRST lub MQTT_DEPTH, przy czym dostępne są co najmniej następujące wartości:
    - Jeden (MQTT_FIRST) lub
    - **TriggerDepth** (MQTT_DEPTH)

    Spełnione są również komunikaty w kolejce o odpowiednim priorytecie (warunek “2” na stronie 898 ) i warunki od “6” na stronie 899 do “10” na stronie 900 .

    Ma to na celu umożliwienie serwerowi kolejek, który wykonuje wywołanie MQGET, znalezienie kolejki pustej i tak dalej, ale w odstępie czasu między wywołaniami MQGET i MQCLOSE odbierany jest co najmniej jeden komunikat.

**Uwaga:**

a. Jeśli program obsługujący kolejkę aplikacji nie pobiera wszystkich komunikatów, może to spowodować pętlę zamkniętą. Za każdym razem, gdy program zamyka kolejkę, menedżer kolejek tworzy kolejny komunikat wyzwalacza, który powoduje ponowne uruchomienie programu serwera przez monitor wyzwalacza.

b. Jeśli program obsługujący kolejkę aplikacji wycofa żądanie pobrania (lub jeśli program zostanie zakończony awaryjnie) przed zamknięciem kolejki, nastąpi to samo. Jeśli jednak program zamknie kolejkę przed wycofaniem żądania pobrania, a kolejka jest pusta, komunikat wyzwalacza nie zostanie utworzony.

c. Aby zapobiec występowaniu takiej pętli, należy użyć pola *BackoutCount* deskryptora MQMD w celu wykrycia komunikatów, które są wielokrotnie wycofywane. Więcej informacji na ten temat zawiera sekcja "Wycofane komunikaty" na stronie 47.

13. Przy użyciu komendy MQSET lub komendy spełnione są następujące warunki:

a. • Wartość **TriggerControl** została zmieniona na MQTC_ON lub

• **TriggerControl** ma już wartość MQTC_ON, a wartość **TriggerType**, **TriggerMsgPriority**lub **TriggerDepth** (jeśli ma zastosowanie) została zmieniona,

i jest co najmniej:

• Jeden (MQTT_FIRST lub MQTT_EVERY), lub

• **TriggerDepth** (MQTT_DEPTH)

komunikaty w kolejce o odpowiednim priorytecie (warunek "2" na stronie 898 ) i warunki od "4" na stronie 899 do "10" na stronie 900 (z wyjątkiem "8" na stronie 900 ) są również spełnione.

Ma to na celu umożliwienie aplikacji lub operatorowi zmiany kryteriów wyzwalania, gdy warunki wyzwalacza są już spełnione.

b. Wartość atrybutu kolejki **InhibitPut** kolejki inicjującej zmienia się z MQQA_PUT_INHIBITED na MQQA_PUT_ALLOWED i jest co najmniej:

• Jeden (MQTT_FIRST lub MQTT_EVERY), lub

• **TriggerDepth** (MQTT_DEPTH)

komunikaty o odpowiednim priorytecie (warunek "2" na stronie 898 ) w dowolnej z kolejek, dla których jest to kolejka inicjująca, a także spełnione są warunki od "4" na stronie 899 do "10" na stronie 900 . Dla każdej kolejki spełniającej warunki generowany jest jeden komunikat wyzwalacza.

Ma to na celu umożliwienie generowania komunikatów wyzwalacza z powodu warunku MQQA_PUT_INHIBITED w kolejce inicjującej, ale ten warunek został zmieniony.

c. Wartość atrybutu kolejki **InhibitGet** kolejki aplikacji zmienia się z MQQA_GET_INHIBITED na MQQA_GET_ALLOWED i jest co najmniej:

• Jeden (MQTT_FIRST lub MQTT_EVERY), lub

• **TriggerDepth** (MQTT_DEPTH)

komunikaty o odpowiednim priorytecie (warunek "2" na stronie 898 ) w kolejce i spełnione są również warunki od "4" na stronie 899 do "10" na stronie 900, z wyjątkiem "5" na stronie 899.

Dzięki temu aplikacje mogą być wyzwalane tylko wtedy, gdy mogą pobierać komunikaty z kolejki aplikacji.

d. Aplikacja monitora wyzwalacza wysyła wywołanie MQOPEN dla danych wejściowych z kolejki inicjującej i istnieją co najmniej następujące elementy:

• Jeden (MQTT_FIRST lub MQTT_EVERY), lub

• **TriggerDepth** (MQTT_DEPTH)

komunikaty o odpowiednim priorytecie (warunek "2" na stronie 898 ) w dowolnej kolejce aplikacji, dla której jest to kolejka inicjująca, oraz warunki od "4" na stronie 899 do "10" na stronie 900 (z

wyjątkiem “8” na stronie 900 ) są również spełnione, a żadna inna aplikacja nie ma otwartej kolejki inicjującej dla wejścia (dla każdej takiej kolejki jest generowany jeden komunikat wyzwalacza spełniający warunki).

Pozwala to na umieszczanie komunikatów w kolejkach, gdy monitor wyzwalacza nie jest uruchomiony, oraz na restartowanie menedżera kolejek i utratę komunikatów wyzwalacza (które są nietrwałe).

14. Parametr MSGDLVSQ jest ustawiony poprawnie. Jeśli zostanie ustawiona wartość MSGDLVSQ=FIFO, komunikaty będą dostarczane do kolejki w oparciu o pierwszy przyszedł-pierwszy wyszedł. Priorytet komunikatu jest ignorowany, a do komunikatu przypisywany jest domyślny priorytet kolejki. Jeśli parametr **TriggerMsgPriority** ma wartość wyższą niż domyślny priorytet kolejki, nie są wyzwalane żadne komunikaty. Jeśli parametr **TriggerMsgPriority** jest ustawiony na wartość równą domyślnemu priorytetowi kolejki lub od niego niższą, wyzwalanie jest wykonywane dla typów FIRST, EVERY i DEPTH. Informacje na temat tych typów zawiera opis pola **TriggerType** w sekcji “Kontrolowanie zdarzeń wyzwalających” na stronie 902.

    Jeśli zostanie ustawiona wartość MSGDLVSQ=PRIORITY, a priorytet komunikatu jest równy lub większy niż wartość w polu *TriggerMsgPriority* , komunikaty będą uwzględniane tylko w zdarzeniu wyzwalającym. W tym przypadku wyzwalanie występuje dla typów FIRST, EVERY i DEPTH. Na przykład, jeśli zostanie umieszczonych 100 komunikatów o niższym priorytecie niż **TriggerMsgPriority**, efektywna głębokość kolejki dla celów wyzwalania będzie nadal wynosić zero. Jeśli w kolejce zostanie umieszczony kolejny komunikat, ale tym razem priorytet będzie większy lub równy wartości **TriggerMsgPriority**, efektywne zapełnienie kolejki zostanie zwiększone z zera do jednego i spełniony zostanie warunek **TriggerType** FIRST.

**Uwagi:**

1. Począwszy od kroku “12” na stronie 900 (w którym komunikaty wyzwalacza są generowane w wyniku zdarzenia innego niż komunikat nadchodzący do kolejki aplikacji), komunikat wyzwalacza nie jest umieszczany jako część jednostki pracy. Ponadto, jeśli parametr **TriggerType** ma wartość MQTT_EVERY i jeśli w kolejce aplikacji znajduje się co najmniej jeden komunikat, generowany jest tylko jeden komunikat wyzwalacza.
2. Jeśli program IBM MQ segmentuje komunikat podczas operacji MQPUT, zdarzenie wyzwalacza nie zostanie przetworzone, dopóki wszystkie segmenty nie zostaną pomyślnie umieszczone w kolejce. Jednak po umieszczeniu segmentów komunikatów w kolejce program IBM MQ traktuje je jako pojedyncze komunikaty na potrzeby wyzwalania. Na przykład pojedynczy komunikat logiczny podzielony na trzy części powoduje, że tylko jedno zdarzenie wyzwalające jest przetwarzane podczas pierwszego wywołania MQPUT i segmentowane. Jednak każdy z tych trzech segmentów powoduje, że ich własne zdarzenia wyzwalające są przetwarzane podczas ich przenoszenia przez sieć IBM MQ .
3. W przypadku systemu IBM MQ for z/OS, jeśli kolejka współużytkowana jest skonfigurowana do wyzwalania i nawiązywania połączenia z narzędziem CF udostępniającym tę kolejkę współużytkowaną, może zostać wygenerowane zdarzenie wyzwalające i umieszczony komunikat w kolejce inicjującej. Taka sytuacja może wystąpić nawet wtedy, gdy żaden komunikat nie został umieszczony w oryginalnej konfiguracji kolejki współużytkowanej w celu wyzwolenia. Jest to spowodowane nadmiernym wskazaniem bitów przez makro IXLVECTR, zgodnie z opisem w sekcji The List Notification Vector.

## *Kontrolowanie zdarzeń wyzwalających*

Za pomocą niektórych atrybutów definiujących kolejkę aplikacji można sterować zdarzeniami wyzwalaczami. Poniższe informacje zawierają również przykłady użycia typów wyzwalaczy: KAŻDY, PIERWSZY i GŁĘBOKOŚĆ.

Można włączyć lub wyłączyć wyzwalanie oraz wybrać liczbę lub priorytet komunikatów, które mają być uwzględniane w zdarzeniu wyzwalającym. Pełny opis tych atrybutów znajduje się w sekcji Atrybuty obiektów.

Odpowiednie atrybuty to:

***TriggerControl***
: Atrybut ten służy do włączania i wyłączania wyzwalania dla kolejki aplikacji.

***TriggerMsgPriority***
Minimalny priorytet, jaki musi mieć komunikat, aby mógł zostać zaliczony do zdarzenia wyzwalającego. Jeśli do kolejki aplikacji zostanie odebrany komunikat o priorytecie mniejszym niż *TriggerMsgPriority* , menedżer kolejek ignoruje ten komunikat podczas określania, czy ma zostać utworzony komunikat wyzwalacza. Jeśli parametr *TriggerMsgPriority* ma wartość zero, wszystkie komunikaty są uwzględniane w zdarzeniu wyzwalającym.

***TriggerType***
Oprócz wyzwalacza typu NONE (który wyłącza wyzwalanie w taki sam sposób, jak ustawienie parametru *TriggerControl* na OFF), można użyć następujących typów wyzwalaczy, aby ustawić czułość kolejki w celu wyzwalania zdarzeń:

**Każdy**
Zdarzenie wyzwalające występuje za każdym razem, gdy komunikat pojawia się w kolejce aplikacji. Tego typu wyzwalacza należy użyć, aby uruchomić wiele instancji aplikacji.

**PIERWSZE**
Zdarzenie wyzwalające występuje tylko wtedy, gdy liczba komunikatów w kolejce aplikacji zmienia się z zera na jeden. Tego typu wyzwalacza należy użyć, jeśli program udostępniający ma zostać uruchomiony po nadejściu pierwszego komunikatu do kolejki, kontynuować, aż nie będzie więcej komunikatów do przetworzenia, a następnie zakończyć działanie. Kolejkę należy zawsze przetwarzać, dopóki nie będzie pusta. Patrz także "Szczególny przypadek wyzwalacza typu FIRST" na stronie 904.

**Głębokość**
Zdarzenie wyzwalające występuje tylko wtedy, gdy liczba komunikatów w kolejce aplikacji osiągnie wartość atrybutu **TriggerDepth** . Typowym zastosowaniem tego typu wyzwalania jest uruchomienie programu po odebraniu wszystkich odpowiedzi na zestaw żądań.

**Wyzwalanie według głębokości:** Po wyzwoleniu przez głębokość menedżer kolejek wyłącza wyzwalanie (za pomocą atrybutu *TriggerControl* ) po utworzeniu komunikatu wyzwalacza. Po wykonaniu tej operacji aplikacja musi ponownie włączyć wyzwalanie samego siebie (za pomocą wywołania MQSET).

Działanie wyłączania wyzwalania nie jest sterowane za pomocą punktu synchronizacji, dlatego nie można ponownie włączyć wyzwalania przez wycofanie jednostki pracy. Jeśli program wycofa żądanie umieszczenia, które spowodowało zdarzenie wyzwalające, lub jeśli program zostanie zakończony awaryjnie, należy ponownie włączyć wyzwalanie za pomocą wywołania MQSET lub komendy ALTER QLOCAL.

***TriggerDepth***
Liczba komunikatów w kolejce, które powodują wyzwalanie zdarzenia, gdy używane jest wyzwalanie przez zapełnienie.

Warunki, które muszą zostać spełnione, aby menedżer kolejek utworzył komunikat wyzwalacza, są opisane w sekcji "Warunki dla zdarzenia wyzwalającego" na stronie 898.

## Przykład użycia wyzwalacza typu EVERY

Rozważmy aplikację, która generuje żądania dla ubezpieczeń komunikacyjnych. Aplikacja może wysyłać komunikaty żądań do pewnej liczby firm ubezpieczeniowych, określając za każdym razem tę samą kolejkę odpowiedzi. W tej kolejce odpowiedzi może zostać ustawiony wyzwalacz typu EVERY, tak aby po każdym nadejściu odpowiedzi odpowiedź mogła wyzwolić instancję serwera w celu przetworzenia odpowiedzi.

## Przykład użycia wyzwalacza typu FIRST

Należy rozważyć organizację z wieloma oddziałami, z których każda przekazuje szczegóły dotyczące dni roboczych do centrali. Wszyscy robią to w tym samym czasie, pod koniec dnia roboczego, a w centrali znajduje się aplikacja, która przetwarza szczegóły ze wszystkich oddziałów. Pierwszy komunikat, który zostanie odebrany do centrali, może spowodować wystąpienie zdarzenia wyzwalającego, które uruchamia tę aplikację. Ta aplikacja będzie kontynuować przetwarzanie do momentu, gdy w jej kolejce nie będzie więcej komunikatów.

## Przykład użycia wyzwalacza typu DEPTH

Rozważmy aplikację biura podróży, która tworzy pojedyncze żądanie potwierdzenia rezerwacji lotu, potwierdzenia rezerwacji pokoju hotelowego, wypożyczenia samochodu i zamówienia czeków podróżnych. Aplikacja może rozdzielić te elementy na cztery komunikaty żądań, wysyłając każdy z nich do osobnego miejsca docelowego. Może on ustawić wyzwalacz typu DEPTH w swojej kolejce odpowiedzi (z głębokością ustawioną na wartość 4), tak aby był restartowany dopiero po nadejściu wszystkich czterech odpowiedzi.

Jeśli inny komunikat (prawdopodobnie z innego żądania) pojawi się w kolejce odpowiedzi przed ostatnimi z czterech odpowiedzi, aplikacja żądająca zostanie uruchomiona wcześniej. Aby tego uniknąć, podczas używania wyzwalania DEPTH w celu gromadzenia wielu odpowiedzi na żądanie należy zawsze używać nowej kolejki odpowiedzi dla każdego żądania.

## Szczególny przypadek wyzwalacza typu FIRST

W przypadku wyzwalacza typu FIRST, jeśli w kolejce aplikacji znajduje się już komunikat po nadejściu innego komunikatu, menedżer kolejek zwykle nie tworzy innego komunikatu wyzwalacza.

Jednak aplikacja obsługująca kolejkę może nie otworzyć kolejki (na przykład aplikacja może zostać zakończona, prawdopodobnie z powodu problemu z systemem). Jeśli w obiekcie definicji procesu zostanie umieszczona niepoprawna nazwa aplikacji, aplikacja obsługująca kolejkę nie odbierze żadnego komunikatu. W takich sytuacjach, jeśli w kolejce aplikacji pojawi się inny komunikat, oznacza to, że nie jest uruchomiony żaden serwer, który przetworzył ten komunikat (i wszystkie inne komunikaty w kolejce).

W tym celu menedżer kolejek tworzy kolejne komunikaty wyzwalacza w następujących okolicznościach:

- Jeśli do kolejki aplikacji zostanie odebrany inny komunikat, ale tylko wtedy, gdy upłynął predefiniowany przedział czasu od momentu utworzenia przez menedżera kolejek ostatniego komunikatu wyzwalacza dla tej kolejki. Ten przedział czasu jest zdefiniowany w atrybucie menedżera kolejek *TriggerInterval*. Wartością domyślną jest 999 999 999 milisekund.
- W systemie IBM MQ for z/OSkolejki aplikacji o nazwie otwartej kolejki inicjującej są okresowo skanowane. Jeśli upłynęło *TRIGINT* milisekund od wysłania ostatniego komunikatu wyzwalacza, a kolejka spełnia warunki dla zdarzenia wyzwalacza i wartość CURDEPTH jest większa od zera, generowany jest komunikat wyzwalacza. Ten proces jest nazywany wyzwalaniem backstop.

Decydując o wartości odstępu czasu wyzwalacza, która ma być używana w aplikacji, należy wziąć pod uwagę następujące kwestie:

- Jeśli parametr *TriggerInterval* zostanie ustawiony na niską wartość, a nie ma aplikacji obsługującej kolejkę aplikacji, wyzwalacz typu FIRST może zachowywać się jak wyzwalacz typu EVERY. Zależy to od częstotliwości umieszczania komunikatów w kolejce aplikacji, która z kolei może zależeć od innych działań systemu. Dzieje się tak dlatego, że jeśli odstęp czasu wyzwalacza jest bardzo mały, za każdym razem, gdy komunikat jest umieszczany w kolejce aplikacji, generowany jest kolejny komunikat wyzwalacza, nawet jeśli typem wyzwalacza jest FIRST, a nie EVERY. (Typ wyzwalacza FIRST z zerowym odstępem czasu wyzwalacza jest równoważny typowi wyzwalacza EVERY).
- W systemie IBM MQ for z/OS , jeśli parametr *TRIGINT* zostanie ustawiony na niską wartość, a żadna aplikacja nie obsługuje kolejki aplikacji typu FIRST, wyzwalanie backstop wygeneruje komunikat wyzwalacza za każdym razem, gdy wykonywane jest okresowe skanowanie kolejek aplikacji o nazwach otwartych kolejek inicjujących.
- Jeśli jednostka pracy została wycofana (patrz sekcja Wyzwalanie komunikatów i jednostek pracy ). a odstęp czasu wyzwalacza został ustawiony na wysoką wartość (lub wartość domyślną), po wycofaniu jednostki pracy generowany jest jeden komunikat wyzwalacza. Jeśli jednak odstęp czasu wyzwalacza został ustawiony na niską wartość lub na zero (co powoduje, że wyzwalacz typu FIRST zachowuje się jak wyzwalacz typu EVERY), może zostać wygenerowanych wiele komunikatów wyzwalacza. Jeśli jednostka pracy zostanie wycofana, wszystkie komunikaty wyzwalacza będą nadal dostępne. Liczba generowanych komunikatów wyzwalacza zależy od odstępu czasu wyzwalacza. Jeśli odstęp czasu wyzwalacza jest ustawiony na zero, generowana jest maksymalna liczba komunikatów.

### *Projektowanie aplikacji korzystającej z kolejek wyzwalanych*

Przedstawiono sposób konfigurowania i sterowania wyzwalaniem aplikacji. Poniżej przedstawiono kilka wskazówek, które należy wziąć pod uwagę podczas projektowania aplikacji.

## Wyzwalanie komunikatów i jednostek pracy

Komunikaty wyzwalacza utworzone z powodu zdarzeń wyzwalacza, które nie są częścią jednostki pracy, są umieszczane w kolejce inicjującej, poza dowolną jednostką pracy, bez zależności od innych komunikatów i są natychmiast dostępne do pobrania przez monitor wyzwalacza.

Komunikaty wyzwalacza utworzone z powodu zdarzeń wyzwalacza, które są częścią jednostki pracy, są udostępniane w kolejce inicjującej po rozstrzygnięciu jednostki pracy, niezależnie od tego, czy jednostka pracy została zatwierdzona, czy wycofana.

Jeśli menedżer kolejek nie umieści komunikatu wyzwalacza w kolejce inicjującej, zostanie on umieszczony w kolejce niedostarczonych komunikatów.

**Uwaga:**

1. Menedżer kolejek zlicza zarówno zatwierdzone, jak i niezatwierdzone komunikaty podczas oceny, czy istnieją warunki dla zdarzenia wyzwalającego.

   W przypadku wyzwalania typu FIRST lub DEPTH, komunikaty wyzwalacza są udostępniane nawet wtedy, gdy jednostka pracy jest wycofana, tak aby komunikat wyzwalacza był zawsze dostępny, gdy spełnione są wymagane warunki. Na przykład można rozważyć żądanie umieszczenia w obrębie jednostki pracy dla kolejki, która jest wyzwalana z wyzwalaczem typu FIRST. Powoduje to, że menedżer kolejek tworzy komunikat wyzwalacza. Jeśli wystąpi kolejne żądanie umieszczenia z innej jednostki pracy, nie spowoduje to kolejnego zdarzenia wyzwalającego, ponieważ liczba komunikatów w kolejce aplikacji zmieniła się z jednego na dwa, co nie spełnia warunków dla zdarzenia wyzwalającego. Jeśli pierwsza jednostka pracy została wycofana, ale druga została zatwierdzona, komunikat wyzwalacza jest nadal tworzony.

   Oznacza to jednak, że komunikaty wyzwalacza są czasami tworzone, gdy warunki zdarzenia wyzwalacza nie są spełnione. Aplikacje, które używają wyzwalania, muszą być zawsze przygotowane do obsługi tej sytuacji. Zaleca się użycie opcji wait z wywołaniem MQGET, ustawiając dla parametru *WaitInterval* odpowiednią wartość.

   Utworzone komunikaty wyzwalacza są zawsze udostępniane, niezależnie od tego, czy jednostka pracy została wycofana, czy zatwierdzona.
2. W przypadku lokalnych kolejek współużytkowanych (czyli kolejek współużytkowanych w grupie współużytkowania kolejek) menedżer kolejek zlicza tylko zatwierdzone komunikaty.

## Pobieranie komunikatów z wyzwalanej kolejki

Podczas projektowania aplikacji korzystających z wyzwalania należy pamiętać, że może wystąpić opóźnienie między uruchomieniem programu przez monitor wyzwalacza a udostępnieniem innych komunikatów w kolejce aplikacji. Taka sytuacja może wystąpić, gdy komunikat, który powoduje zdarzenie wyzwalające, zostanie zatwierdzony przed innymi.

Aby zapewnić czas na nadejście komunikatów, należy zawsze używać opcji oczekiwania w przypadku użycia wywołania MQGET w celu usunięcia komunikatów z kolejki, dla której są ustawione warunki wyzwalacza. Parametr *WaitInterval* musi być wystarczający, aby zapewnić najdłuższy rozsądny czas między umieszczaniem komunikatu a zatwierdzaniem wywołania umieszczania. Jeśli komunikat jest odbierany ze zdalnego menedżera kolejek, na ten czas mają wpływ:

- Liczba komunikatów umieszczonych przed zatwierdzeniem
- Szybkość i dostępność łącza komunikacyjnego
- Wielkości komunikatów

Przykład sytuacji, w której należy użyć wywołania MQGET z opcją wait, należy wziąć pod uwagę ten sam przykład, który był używany podczas opisywania jednostek pracy. Było to żądanie umieszczenia w obrębie

jednostki pracy dla kolejki wyzwalanej z wyzwalaczem typu FIRST. To zdarzenie powoduje, że menedżer kolejek tworzy komunikat wyzwalacza. Jeśli wystąpi kolejne żądanie umieszczenia z innej jednostki pracy, nie spowoduje to kolejnego zdarzenia wyzwalającego, ponieważ liczba komunikatów w kolejce aplikacji nie zmieniła się z zera na jeden. Jeśli pierwsza jednostka pracy została wycofana, ale druga została zatwierdzona, komunikat wyzwalacza jest nadal tworzony. Dlatego komunikat wyzwalacza jest tworzony w momencie, gdy wycofywana jest pierwsza jednostka pracy. Jeśli przed zatwierdzeniem drugiego komunikatu wystąpi znaczne opóźnienie, wyzwolona aplikacja może oczekiwać na ten komunikat.

W przypadku wyzwalania typu DEPTH może wystąpić opóźnienie, nawet jeśli wszystkie odpowiednie komunikaty zostaną ostatecznie zatwierdzone. Załóżmy, że atrybut kolejki **TriggerDepth** ma wartość 2. Po nadejściu dwóch komunikatów do kolejki drugi powoduje utworzenie komunikatu wyzwalacza. Jeśli jednak drugi komunikat jest pierwszym, który zostanie zatwierdzony, wówczas komunikat wyzwalacza stanie się dostępny. Monitor wyzwalacza uruchamia program serwera, ale program może wczytać tylko drugi komunikat do momentu zatwierdzenia pierwszego. Dlatego program może wymagać oczekiwania na udostępnienie pierwszego komunikatu.

Zaprojektuj aplikację w taki sposób, aby kończona była w przypadku braku dostępnych komunikatów do pobrania po upływie okresu oczekiwania. Jeśli co najmniej jeden komunikat pojawi się później, należy polegać na ponownym wyzwoleniu aplikacji w celu przetworzenia tych komunikatów. Ta metoda zapobiega bezczynności aplikacji i niepotrzebnemu wykorzystywaniu zasobów.

## *Przetwarzanie kolejki inicjującej przez monitory wyzwalacza*

Dla menedżera kolejek monitor wyzwalacza jest podobny do każdej innej aplikacji obsługującej kolejkę. Jednak monitor wyzwalacza obsługuje kolejki inicjowania.

Monitor wyzwalacza jest zwykle programem działającym w trybie ciągłym. Gdy komunikat wyzwalacza pojawia się w kolejce inicjującej, monitor wyzwalacza pobiera ten komunikat. Na podstawie informacji zawartych w komunikacie uruchamia komendę uruchamiającą aplikację, która przetwarza komunikaty w kolejce aplikacji.

Monitor wyzwalacza musi przekazać wystarczającą ilość informacji do programu, który jest uruchamiany, aby program mógł wykonać poprawne działania w odpowiedniej kolejce aplikacji.

Inicjator kanału jest przykładem specjalnego typu monitora wyzwalacza dla agentów kanału komunikatów. Jednak w takiej sytuacji należy użyć wyzwalacza typu FIRST lub DEPTH.

ALW *Monitory wyzwalacza w systemach AIX, Linux, and Windows*

Ta sekcja zawiera informacje na temat monitorów wyzwalaczy udostępnianych w systemach AIX, Linux, and Windows .

Dla środowiska serwera udostępniono następujące monitory wyzwalacza:

**amqstrg0**
: Jest to przykładowy monitor wyzwalacza, który udostępnia podzbiór funkcji udostępnianych przez program **runmqtrm**. Więcej informacji na temat komendy amqstrg0zawiera sekcja “Korzystanie z przykładowych programów w systemie Multiplatforms” na stronie 1090 .

**runmqtrm,**
: Składnia tej komendy jest następująca: **runmqtrm** [ *-m QMgrName* ] [ *-q InitQ* ], gdzie QMgrName jest menedżerem kolejek, a InitQ jest kolejką inicjującą. Domyślna nazwa kolejki to SYSTEM.DEFAULT.INITIATION.QUEUE w domyślnym menedżerze kolejek. Wywołuje on programy dla odpowiednich komunikatów wyzwalacza. Ten monitor wyzwalacza obsługuje domyślny typ aplikacji.

  Łańcuch komendy przekazywany przez monitor wyzwalacza do systemu operacyjnego jest budowany w następujący sposób:

  1. *ApplId* z odpowiedniej definicji PROCESS (jeśli została utworzona)
  2. Struktura MQTMC2 ujęta w cudzysłów
  3. *EnvData* z odpowiedniej definicji PROCESS (jeśli została utworzona)

gdzie *ApplId* jest nazwą programu, który ma zostać uruchomiony w taki sposób, w jakim zostałby wprowadzony w wierszu komend.

Przekazany parametr jest strukturą znaków MQTMC2 . Wywoływany jest łańcuch komendy, który zawiera ten łańcuch, dokładnie tak, jak został podany, w cudzysłowie, w celu zaakceptowania go przez komendę systemową jako jednego parametru.

Monitor wyzwalacza nie sprawdza, czy w kolejce inicjującej znajduje się inny komunikat do momentu zakończenia aplikacji, która właśnie została uruchomiona. Jeśli aplikacja ma do wykonania wiele operacji przetwarzania, monitor wyzwalacza może nie być w stanie nadążyć za liczbą nadchodzących komunikatów wyzwalacza. Dostępne są dwie opcje:

- Uruchom więcej monitorów wyzwalaczy
- Uruchamianie uruchomionych aplikacji w tle

Jeśli uruchomionych jest więcej monitorów wyzwalaczy, można kontrolować maksymalną liczbę aplikacji, które mogą być uruchomione jednocześnie. Jeśli aplikacje są uruchamiane w tle, program IBM MQ nie nakłada ograniczeń na liczbę aplikacji, które mogą być uruchamiane.

Linux AIX Aby uruchomić uruchomioną aplikację w tle w systemie AIX and Linux, umieść & na końcu *EnvData* definicji PROCESS.

Aby uruchomić uruchomioną aplikację w tle w systemach Windows , w polu *ApplId* należy poprzedzić nazwę aplikacji komendą START. Na przykład:

```
START ?B AMQSECHA
```

**Uwaga:** Windows Jeśli ścieżka Windows zawiera spacje w nazwie ścieżki, należy je ująć w cudzysłów ("). aby upewnić się, że jest on obsługiwany jako pojedynczy argument. Na przykład "`C:\Program Files\Application Directory\Application.exe`".

Poniżej przedstawiono przykład łańcucha APPLICID, w którym nazwa pliku zawiera spacje w ścieżce:

```
START "" /B "C:\Program Files\Application Directory\Application.exe"
```

Składnia komendy Windows START w przykładzie zawiera pusty łańcuch ujęty w cudzysłów. START określa, że pierwszy argument w cudzysłowie będzie traktowany jako tytuł nowej komendy. Aby upewnić się, że Windows nie pomyli ścieżki aplikacji z argumentem 'title', dodaj łańcuch tytułu ujęty w cudzysłów przed nazwą aplikacji.

Dla klienta IBM MQ dostępne są następujące monitory wyzwalacza:

**Komenda runmqtmc**
Jest on taki sam, jak komenda runmqtrm, z tą różnicą, że łączy się z bibliotekami IBM MQ MQI client .

ALW *Monitor wyzwalacza dla CICS*
Monitor wyzwalacza amqltmc0 jest udostępniany dla systemu CICS. Działa on w taki sam sposób, jak standardowy monitor wyzwalacza, runmqtrm, ale użytkownik uruchamia go w inny sposób i wyzwala transakcje CICS .

Ten temat dotyczy tylko systemów Windows, AIX and Linux x86-64 .

Monitor wyzwalacza jest dostarczany jako program CICS ; zdefiniuj go za pomocą 4-znakowej nazwy transakcji. Wprowadź 4-znakową nazwę, aby uruchomić monitor wyzwalacza. Używa on domyślnego menedżera kolejek (zgodnie z nazwą w pliku qm.ini lub, w systemie IBM MQ for Windows, rejestru) oraz systemu SYSTEM.CICS.INITIATION.QUEUE.

Aby użyć innego menedżera kolejek lub innej kolejki, należy zbudować strukturę monitora wyzwalacza MQTMC2 : wymaga to napisania programu przy użyciu wywołania EXEC CICS START, ponieważ struktura

jest zbyt długa, aby można ją było dodać jako parametr. Następnie przekaż strukturę MQTMC2 jako dane do żądania START dla monitora wyzwalacza.

Jeśli używana jest struktura MQTMC2 , należy do monitora wyzwalacza podać tylko parametry *StrucId*, *Version*, *QName*i **QMgrName** , ponieważ nie odwołują się one do żadnych innych pól.

Komunikaty są odczytywane z kolejki inicjującej i używane do uruchamiania transakcji CICS przy użyciu komendy EXEC CICS START, przy założeniu, że wartością APPL_TYPE w komunikacie wyzwalacza jest MQAT_CICS. Odczyt komunikatów z kolejki inicjującej jest wykonywany pod kontrolą punktu synchronizacji CICS .

Komunikaty są generowane podczas uruchamiania i zatrzymywania monitora oraz w przypadku wystąpienia błędu. Te komunikaty są wysyłane do kolejki danych przejściowych CSMT.

*Tabela 129. Dostępne wersje monitora wyzwalacza.*

Tabela z dwiema kolumnami. Pierwsza kolumna zawiera listę dostępnych wersji monitora wyzwalacza, a druga kolumna pokazuje platformy, dla których używana jest każda wersja.

| **Wersja** | **Użycie** |
|---|---|
| amqltmc0 | TXSeries Przez:<br>• AIX AIX<br>• Linux Systemy Linux x86-64 |
| amqltmc4 | Windows TXSeries dla Windows 5.1 |
| amqltmcc | Powiązana z klientem wersja monitora wyzwalacza CICS |

Jeśli potrzebny jest monitor wyzwalacza dla innych środowisk, należy napisać program, który może przetwarzać komunikaty wyzwalacza umieszczane przez menedżer kolejek w kolejkach inicjujących. Taki program powinien wykonywać następujące czynności:

1. Wywołanie MQGET służy do oczekiwania na przybycie komunikatu do kolejki inicjującej.
2. Sprawdź pola struktury MQTM komunikatu wyzwalacza, aby znaleźć nazwę aplikacji do uruchomienia oraz środowisko, w którym jest ona uruchamiana.
3. Wydaj komendę uruchamiania specyficzną dla środowiska.

   z/OS Na przykład w przypadku zadania wsadowego z/OS należy wprowadzić zadanie do wewnętrznego programu czytającego.
4. W razie potrzeby przekształć strukturę MQTM w strukturę MQTMC2 .
5. Przekaż strukturę MQTMC2 lub MQTM do uruchomionej aplikacji. Może zawierać dane użytkownika.
6. Powiąż z kolejką aplikacji aplikację, która ma obsługiwać tę kolejkę. W tym celu należy nadać nazwę obiektowi definicji procesu (jeśli został utworzony) w atrybucie **ProcessName** kolejki. Aby nazwać obiekt definicji procesu, można użyć komendy **DEFINE QLOCAL** lub **ALTER QLOCAL** .

   IBM i W systemie IBM imożna również użyć komendy CRTMQMQ lub CHGMQMQ.

Więcej informacji na temat interfejsu monitora wyzwalacza zawiera sekcja MQTMC2.

IBM i *Monitory wyzwalacza w systemie IBM i*

W systemie IBM izamiast komendy sterującej **runmqtrm** należy użyć IBM MQ for IBM i komendy CL **STRMQMTRM**.

Użyj komendy STRMQMTRM w następujący sposób:

```
STRMQMTRM INITQNAME(InitQ) MQMNAME(QMgrName)
```

Szczegóły są takie same jak w przypadku komendy runmqtrm.

Dostępne są również następujące programy przykładowe, których można używać jako modeli do pisania własnych monitorów wyzwalaczy:

**AMQSTRG4**
Jest to monitor wyzwalacza, który wprowadza zadanie IBM i dla procesu, który ma zostać uruchomiony, ale oznacza to, że z każdym komunikatem wyzwalacza powiązane jest dodatkowe przetwarzanie.

**AMQSERV4**
Jest to serwer wyzwalacza. Dla każdego komunikatu wyzwalacza ten serwer uruchamia komendę dla procesu w swoim własnym zadaniu i może wywoływać transakcje CICS .

Zarówno monitor wyzwalacza, jak i serwer wyzwalacza przekazują strukturę MQTMC2 do uruchamianych przez nie programów. Opis tej struktury zawiera sekcja MQTMC2. Oba te przykłady są dostarczane zarówno w postaci kodu źródłowego, jak i kodu wykonywalnego.

Ponieważ te monitory wyzwalacza mogą wywoływać tylko rodzime programy IBM i , nie mogą bezpośrednio wyzwalać programów Java , ponieważ klasy Java znajdują się w systemie plików IFS. Jednak programy Java mogą być wyzwalane pośrednio przez wywołanie programu CL, który następnie wywołuje program Java i przekazuje dane przez strukturę TMC2 . Minimalna wielkość struktury TMC2 wynosi 732 bajty.

Poniżej przedstawiono źródło przykładowego procesora CLP:

```
PGM PARM(&TMC2)
  DCL &TMC2 *CHAR LEN(800)
    ADDENVVAR ENVVAR(TM) VALUE(&TMC2)
    QSH  CMD('java_pgmname $TM')
    RMVENVVAR ENVVAR(TM)
ENDPGM
```

Dla produktu IBM MQ MQI clientudostępniono następujący program monitora wyzwalacza: RUNMQTMC

Wywołaj komendę RUNMQTMC w następujący sposób:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QMgrName '-q' InitQ)
```

## *Właściwości komunikatów wyzwalacza*

W poniższych tematach opisano niektóre inne właściwości komunikatów wyzwalacza.

- “Trwałość i priorytet komunikatów wyzwalacza” na stronie 909
- “Komunikaty dotyczące restartowania i wyzwalania menedżera kolejek” na stronie 910
- “Wyzwalanie komunikatów i zmiany atrybutów obiektu” na stronie 910
- “Format komunikatów wyzwalacza” na stronie 910

## Trwałość i priorytet komunikatów wyzwalacza

Komunikaty wyzwalacza nie są trwałe, ponieważ nie jest to wymagane.

Jednak warunki generowania zdarzeń wyzwalających są zachowywane, dlatego komunikaty wyzwalacza są generowane zawsze wtedy, gdy te warunki są spełnione. Jeśli komunikat wyzwalacza zostanie utracony, dalsze istnienie komunikatu aplikacji w kolejce aplikacji gwarantuje, że menedżer kolejek wygeneruje komunikat wyzwalacza natychmiast po spełnieniu wszystkich warunków.

Jeśli jednostka pracy zostanie wycofana, wszystkie wygenerowane przez nią komunikaty wyzwalacza są zawsze dostarczane.

Komunikaty wyzwalacza mają domyślny priorytet kolejki inicjującej.

## Komunikaty dotyczące restartowania i wyzwalania menedżera kolejek

Po restarcie menedżera kolejek, gdy kolejka inicjująca jest otwierana dla wejścia, komunikat wyzwalacza może zostać umieszczony w tej kolejce inicjującej, jeśli powiązana z nią kolejka aplikacji zawiera komunikaty i jest zdefiniowana do wyzwalania.

## Wyzwalanie komunikatów i zmiany atrybutów obiektu

Komunikaty wyzwalacza są tworzone zgodnie z wartościami atrybutów wyzwalacza obowiązującymi w momencie zdarzenia wyzwalacza.

Jeśli komunikat wyzwalacza nie będzie dostępny dla monitora wyzwalacza do momentu późniejszego (ponieważ komunikat, który spowodował jego wygenerowanie, został umieszczony w jednostce pracy), wszelkie zmiany atrybutów wyzwalacza w międzyczasie nie będą miały wpływu na komunikat wyzwalacza. W szczególności wyłączenie wyzwalania nie zapobiega udostępnieniu komunikatu wyzwalacza po jego utworzeniu. Ponadto kolejka aplikacji może już nie istnieć w momencie udostępnienia komunikatu wyzwalacza.

## Format komunikatów wyzwalacza

Format komunikatu wyzwalacza jest definiowany przez strukturę MQTM.

W tym miejscu znajdują się następujące pola, które menedżer kolejek wypełnia podczas tworzenia komunikatu wyzwalacza przy użyciu informacji w definicjach obiektów kolejki aplikacji i procesu powiązanego z tą kolejką:

***StrucId***
Identyfikator struktury.

***Version***
Wersja struktury.

***QName***
Nazwa kolejki aplikacji, w której wystąpiło zdarzenie wyzwalające. Gdy menedżer kolejek tworzy komunikat wyzwalacza, wypełnia to pole przy użyciu atrybutu **QName** kolejki aplikacji.

***ProcessName***
Nazwa obiektu definicji procesu powiązanego z kolejką aplikacji. Gdy menedżer kolejek tworzy komunikat wyzwalacza, wypełnia to pole przy użyciu atrybutu **ProcessName** kolejki aplikacji.

***TriggerData***
Pole w formacie swobodnym używane przez monitor wyzwalacza. Gdy menedżer kolejek tworzy komunikat wyzwalacza, wypełnia to pole przy użyciu atrybutu **TriggerData** kolejki aplikacji. W przypadku każdego IBM MQ produktu z wyjątkiem IBM MQ for z/OStego pola można użyć do określenia nazwy kanału, który ma zostać wyzwolony.

***ApplType***
Typ aplikacji, którą ma uruchomić monitor wyzwalacza. Gdy menedżer kolejek tworzy komunikat wyzwalacza, wypełnia to pole za pomocą atrybutu **ApplType** obiektu definicji procesu określonego w pliku *ProcessName*.

***ApplId***
Łańcuch znaków identyfikujący aplikację, którą ma uruchomić monitor wyzwalacza. Gdy menedżer kolejek tworzy komunikat wyzwalacza, wypełnia to pole za pomocą atrybutu **ApplId** obiektu definicji procesu określonego w pliku *ProcessName*.

Jeśli używany jest monitor wyzwalacza CKTI, dostarczony przez CICS, atrybut **ApplId** obiektu definicji procesu jest identyfikatorem transakcji CICS .

Jeśli używana jest komenda CSQQTRMN dostarczana z systemem IBM MQ for z/OS, atrybut **ApplId** obiektu definicji procesu jest identyfikatorem transakcji IMS .

***EnvData***
Pole znakowe zawierające dane związane ze środowiskiem przeznaczone do użycia przez monitor wyzwalacza. Gdy menedżer kolejek tworzy komunikat wyzwalacza, wypełnia to pole za

pomocą atrybutu **EnvData** obiektu definicji procesu określonego w pliku *ProcessName*. Monitor wyzwalacza dostarczony przez CICS(CKTI) lub monitor wyzwalacza dostarczony przez IBM MQ for z/OS(CSQQTRMN) nie używają tego pola, ale inne monitory wyzwalacza mogą go użyć.

***UserData***
Pole znakowe zawierające dane użytkownika używane przez monitor wyzwalacza. Gdy menedżer kolejek tworzy komunikat wyzwalacza, wypełnia to pole za pomocą atrybutu **UserData** obiektu definicji procesu określonego w pliku *ProcessName*. Tego pola można użyć do określenia nazwy kanału, który ma zostać wyzwolony.

Istnieje pełny opis struktury komunikatu wyzwalacza w programie MQTM.

### *Gdy wyzwalanie nie działa*

Program nie jest wyzwalany, jeśli monitor wyzwalacza nie może uruchomić programu lub menedżer kolejek nie może dostarczyć komunikatu wyzwalacza. Na przykład zmienna applid w obiekcie procesu musi określać, że program ma być uruchamiany w tle; w przeciwnym razie monitor wyzwalacza nie może uruchomić programu.

Jeśli komunikat wyzwalacza został utworzony, ale nie można go umieścić w kolejce inicjującej (na przykład dlatego, że kolejka jest pełna lub długość komunikatu wyzwalacza jest większa niż maksymalna długość komunikatu określona dla kolejki inicjującej), komunikat wyzwalacza jest umieszczany w kolejce niedostarczonych komunikatów.

Jeśli operacja umieszczania w kolejce niedostarczonych komunikatów nie może zostać zakończona pomyślnie, komunikat wyzwalacza jest usuwany, a do operatora systemu wysyłany jest komunikat ostrzegawczy z/OS do konsoli z/OS lub lub dziennik błędów.

Umieszczenie komunikatu wyzwalacza w kolejce niedostarczonych komunikatów może spowodować wygenerowanie komunikatu wyzwalacza dla tej kolejki. Ten drugi komunikat wyzwalacza jest odrzucany, jeśli dodaje komunikat do kolejki niedostarczonych komunikatów.

Jeśli program został wyzwolony pomyślnie, ale został zakończony awaryjnie przed odebraniem komunikatu z kolejki, należy użyć programu narzędziowego do śledzenia (na przykład CICS AUXTRACE, jeśli program jest uruchomiony w systemie CICS ). aby znaleźć przyczynę niepowodzenia.

## Praca z funkcją MQI i klastrami

Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do łączenia w klastry.

Użyj następujących odsyłaczy, aby uzyskać więcej informacji na temat opcji dostępnych w wywołaniach i kodach powrotu dla klastrów:

- “MQOPEN i klastry” na stronie 912
- “MQPUT, MQPUT1 i klastry” na stronie 913
- “MQINQ i klastry” na stronie 914
- “MQSET i klastry” na stronie 914
- “Kody powrotu” na stronie 915

**Pojęcia pokrewne**

“Przegląd interfejsu kolejki komunikatów” na stronie 745
Sekcja zawiera informacje na temat komponentów interfejsu kolejki komunikatów (Message Queue Interface-MQI).

“Nawiązywanie i rozłączanie połączenia z menedżerem kolejek” na stronie 758
Aby można było używać usług programistycznych IBM MQ , program musi mieć połączenie z menedżerem kolejek. Ten temat zawiera informacje o nawiązywaniu połączenia z menedżerem kolejek i rozłączaniu się z nim.

“Otwieranie i zamykanie obiektów” na stronie 766
Informacje te udostępniają wgląd w otwieranie i zamykanie obiektów IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 777

Ta sekcja zawiera informacje na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 792
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Uzyskiwanie informacji o atrybutach obiektu i ustawianie ich” na stronie 876
Atrybuty są właściwościami definiującymi charakterystykę obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 879
W tej sekcji opisano sposób zatwierdzania i wycofywania wszelkich odtwarzalnych operacji pobierania i umieszczania, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy” na stronie 891
Informacje o wyzwalaczach i sposobie uruchamiania aplikacji IBM MQ za pomocą wyzwalaczy.

“Używanie i pisanie aplikacji w systemie IBM MQ for z/OS” na stronie 916
Aplikacje IBM MQ for z/OS mogą być tworzone z programów działających w wielu różnych środowiskach. Oznacza to, że mogą korzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje mostu IMS i IMS w systemie IBM MQ for z/OS” na stronie 71
Informacje te są pomocne podczas pisania aplikacji IMS przy użyciu produktu IBM MQ.

### *MQOPEN i klastry*

Kolejka, do której komunikat jest umieszczany lub z której jest odczytywany po otwarciu kolejki klastra, zależy od wywołania funkcji MQOPEN .

## Wybieranie kolejki docelowej

Jeśli nazwa menedżera kolejek nie zostanie podana w deskryptorze obiektu MQOD, menedżer kolejek wybierze menedżer kolejek, do którego ma zostać wysłany komunikat. Jeśli w deskryptorze obiektu zostanie podana nazwa menedżera kolejek, komunikaty będą zawsze wysyłane do wybranego menedżera kolejek.

Jeśli menedżer kolejek wybiera docelowy menedżer kolejek, wybór zależy od opcji powiązania MQOO_BIND_* oraz od tego, czy istnieje kolejka lokalna. Jeśli istnieje lokalna instancja kolejki, jest ona zawsze otwierana zamiast instancji zdalnej, chyba że atrybut CLWLUSEQ ma wartość ANY. W przeciwnym razie wybór zależy od opcji powiązania. W przypadku korzystania z grup komunikatów z klastrami należy określić parametr MQOO_BIND_ON_OPEN lub MQOO_BIND_ON_GROUP , aby zapewnić, że wszystkie komunikaty w grupie będą przetwarzane w tym samym miejscu docelowym.

Jeśli menedżer kolejek wybiera docelowy menedżer kolejek, robi to w sposób karuzelowy, używając algorytmu zarządzania obciążeniem. Więcej informacji na ten temat zawiera sekcja Równoważenie obciążenia w klastrach.

Jeśli używany jest algorytm równoważenia obciążenia, zależy to od sposobu otwierania kolejki klastra:

- MQOO_BIND_ON_OPEN -algorytm jest używany raz w momencie otwarcia kolejki przez aplikację.
- MQOO_BIND_NOT_FIXED -algorytm jest używany dla każdego komunikatu umieszczanego w kolejce.
- MQOO_BIND_ON_GROUP -algorytm jest używany raz na początku każdej grupy komunikatów.

**MQOO_BIND_ON_OPEN**
Opcja MQOO_BIND_ON_OPEN w wywołaniu MQOPEN określa, że docelowy menedżer kolejek ma zostać naprawiony. Opcji MQOO_BIND_ON_OPEN należy użyć, jeśli w klastrze istnieje wiele instancji tej samej kolejki. Wszystkie komunikaty umieszczane w kolejce z określeniem uchwytu obiektu zwróconego przez wywołanie funkcji MQOPEN są kierowane do tego samego menedżera kolejek.

- Opcji MQOO_BIND_ON_OPEN należy użyć, jeśli komunikaty mają powinowactwa. Jeśli na przykład zadanie wsadowe komunikatów ma być przetwarzane przez ten sam menedżer kolejek, podczas otwierania kolejki należy określić wartość MQOO_BIND_ON_OPEN . Program IBM MQ poprawia menedżer kolejek i trasę, która ma być kierowana przez wszystkie komunikaty umieszczane w tej kolejce.
- Jeśli podano opcję MQOO_BIND_ON_OPEN , należy ponownie otworzyć kolejkę, aby można było wybrać nową instancję kolejki.

**MQOO_BIND_NOT_FIXED**
Opcja MQOO_BIND_NOT_FIXED w wywołaniu MQOPEN określa, że docelowy menedżer kolejek nie jest ustalony. Komunikaty zapisane w kolejce, określające uchwyt obiektu zwrócony z wywołania MQOPEN , są kierowane do menedżera kolejek w czasie MQPUT (komunikat po komunikacie). Opcji MQOO_BIND_NOT_FIXED należy użyć, aby nie wymuszać zapisywania wszystkich komunikatów w tym samym miejscu docelowym.

- Nie należy jednocześnie podawać wartości MQOO_BIND_NOT_FIXED i MQMF_SEGMENTATION_ALLOWED . W przeciwnym razie segmenty komunikatu mogą być dostarczane do różnych menedżerów kolejek, rozrzuconych po całym klastrze.

**MQOO_BIND_ON_GROUP**
Umożliwia aplikacji żądanie przydzielenia grupy komunikatów do tej samej instancji docelowej. Ta opcja jest poprawna tylko dla kolejek i dotyczy tylko kolejek klastra. Jeśli ta opcja zostanie określona dla kolejki, która nie jest kolejką klastra, opcja zostanie zignorowana.

- Grupy są kierowane tylko do jednego miejsca docelowego, jeśli w parametrze MQPUT określono parametr MQPMO_LOGICAL_ORDER. Jeśli określono parametr MQOO_BIND_ON_GROUP, ale komunikat nie jest częścią grupy logicznej, używane jest zachowanie BIND_NOT_FIXED.

**MQOO_BIND_AS_Q_DEF**
Jeśli nie zostanie podana opcja MQOO_BIND_ON_OPEN, MQOO_BIND_NOT_FIXED lub MQOO_BIND_ON_GROUP, wartością domyślną jest MQOO_BIND_AS_Q_DEF. Użycie wartości MQOO_BIND_AS_Q_DEF powoduje, że powiązanie używane dla uchwytu kolejki jest pobierane z atrybutu kolejki DefBind .

## Istotność opcji MQOPEN

MQOPEN Opcje MQOO_BROWSE, MQOO_INPUT_*lub MQOO_SET wymagają lokalnej instancji kolejki klastra, aby komenda MQOPEN powiodła się.

MQOPEN Opcje MQOO_OUTPUT, MQOO_BIND_*lub MQOO_INQUIRE nie wymagają, aby lokalna instancja kolejki klastra zakończyła się pomyślnie.

## Nazwa rozstrzygniętego menedżera kolejek

Gdy nazwa menedżera kolejek jest tłumaczona w czasie MQOPEN , rozstrzygnięta nazwa jest zwracana do aplikacji. Jeśli aplikacja próbuje użyć tej nazwy w kolejnym wywołaniu MQOPEN , może stwierdzić, że nie ma uprawnień dostępu do tej nazwy.

### *MQPUT, MQPUT1 i klastry*

Jeśli określono wartość MQOO_BIND_NOT_FIXED dla MQOPEN , procedury zarządzania obciążeniem wybierają miejsce docelowe MQPUT lub MQPUT1 .

Jeśli w wywołaniu funkcji MQOPEN określono parametr MQOO_BIND_NOT_FIXED , każde kolejne wywołanie funkcji MQPUT wywołuje procedurę zarządzania obciążeniem w celu określenia, do którego menedżera kolejek ma zostać wysłany komunikat. Miejsce docelowe i trasa, które mają zostać wybrane, są wybierane osobno dla każdego komunikatu. Miejsce docelowe i trasa mogą ulec zmianie po umieszczeniu komunikatu, jeśli warunki w sieci ulegną zmianie. Wywołanie MQPUT1 zawsze działa tak, jakby było aktywne MQOO_BIND_NOT_FIXED , to znaczy zawsze wywołuje procedurę zarządzania obciążeniem.

Po wybraniu menedżera kolejek przez procedurę zarządzania obciążeniem lokalny menedżer kolejek kończy operację umieszczania (put). Komunikat można umieścić w różnych kolejkach:

1. Jeśli miejscem docelowym jest lokalna instancja kolejki, komunikat jest umieszczany w kolejce lokalnej.
2. Jeśli miejsce docelowe jest menedżerem kolejek w klastrze, komunikat jest umieszczany w kolejce transmisji klastra.
3. Jeśli miejsce docelowe jest menedżerem kolejek poza klastrem, komunikat jest umieszczany w kolejce transmisji o takiej samej nazwie jak nazwa docelowego menedżera kolejek.

Jeśli w wywołaniu MQOPEN określono parametr MQOO_BIND_ON_OPEN , wywołania MQPUT nie wywołują procedury zarządzania obciążeniem, ponieważ miejsce docelowe i trasa zostały już wybrane.

### *MQINQ i klastry*

To, do której kolejki klastra zostanie wykonane zapytanie, zależy od opcji łączących się z produktem MQOO_INQUIRE.

Zanim możliwe będzie uzyskanie informacji o kolejce, należy ją otworzyć za pomocą wywołania MQOPEN i podać wartość MQOO_INQUIRE.

Aby uzyskać informacje o kolejce klastra, należy użyć wywołania MQOPEN i połączyć inne opcje z opcją MQOO_INQUIRE. Atrybuty, które można sprawdzić, zależą od tego, czy istnieje lokalna instancja kolejki klastra i od sposobu jej otwarcia:

- Połączenie MQOO_BROWSE, MQOO_INPUT_*lub MQOO_SET z MQOO_INQUIRE wymaga lokalnej instancji kolejki klastra, aby otwarcie powiodło się. W takim przypadku można uzyskać informacje o wszystkich atrybutach, które są poprawne dla kolejek lokalnych.
- Łącząc MQOO_OUTPUT z MQOO_INQUIREi nie określając żadnej z powyższych opcji, otwarta instancja ma jedną z następujących wartości:
  - Instancja w lokalnym menedżerze kolejek, jeśli istnieje. W takim przypadku można uzyskać informacje o wszystkich atrybutach, które są poprawne dla kolejek lokalnych.
  - Instancja w innym miejscu w klastrze, jeśli nie istnieje instancja lokalnego menedżera kolejek. W takim przypadku można uzyskać tylko następujące atrybuty. W tym przypadku atrybut QType ma wartość MQQT_CLUSTER .

    - DefBind
    - DefPersistence
    - DefPriority
    - InhibitPut
    - QDesc (Opis kolejek)
    - Nazwa QName
    - QTYPE

Aby uzyskać informacje o atrybucie DefBind kolejki klastra, należy użyć wywołania MQINQ z selektorem MQIA_DEF_BIND. Zwracana wartość to MQBND_BIND_ON_OPEN , MQBND_BIND_NOT_FIXEDlub MQBND_BIND_ON_GROUP. W przypadku korzystania z grup z klastrami należy określić parametr MQBND_BIND_ON_OPEN lub MQBND_BIND_ON_GROUP .

Aby uzyskać informacje na temat atrybutów CLUSTER i CLUSNL lokalnej instancji kolejki, należy użyć wywołania MQINQ z selektorem MQCA_CLUSTER_NAME lub selektorem MQCA_CLUSTER_NAMELIST.

**Uwaga:** Jeśli kolejka klastra zostanie otwarta bez naprawiania kolejki, z którą powiązana jest komenda MQOPEN , kolejne wywołania komendy MQINQ mogą wykonywać zapytania dotyczące różnych instancji kolejki klastra.

**Pojęcia pokrewne**

“Opcja MQOPEN dla kolejki klastra” na stronie 772

Powiązanie używane dla uchwytu kolejki jest pobierane z atrybutu kolejki **DefBind** , który może mieć wartość MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXEDlub MQBND_BIND_ON_GROUP.

### *MQSET i klastry*

Opcja MQOPEN MQOO_SET wymaga lokalnej instancji kolejki klastra, aby program MQSET mógł pomyślnie zakończyć działanie.

Nie można użyć wywołania MQSET do ustawienia atrybutów kolejki w innym miejscu w klastrze.

Można otworzyć lokalny alias lub zdalną kolejkę zdefiniowaną za pomocą atrybutu cluster i użyć wywołania MQSET . Można ustawić atrybuty lokalnego aliasu lub kolejki zdalnej. Nie ma znaczenia, czy kolejka docelowa jest kolejką klastra zdefiniowaną w innym menedżerze kolejek.

### *Kody powrotu*

Kody powrotu specyficzne dla klastrów

**MQRC_CLUSTER_EXIT_ERROR ( 2266 X'8DA')**

Wywołanie MQOPEN, MQPUTlub MQPUT1 jest wysyłane w celu otwarcia kolejki klastra lub umieszczenia w niej komunikatu. Wyjście obciążenia klastra zdefiniowane przez atrybut ClusterWorkloadExit menedżera kolejek nieoczekiwanie kończy się niepowodzeniem lub nie odpowiada w czasie.

W dzienniku systemowym IBM MQ for z/OS zapisywany jest komunikat zawierający więcej informacji o tym błędzie.

Kolejne wywołania MQOPEN, MQPUTi MQPUT1 dla tego uchwytu kolejki są przetwarzane tak, jakby atrybut ClusterWorkloadExit był pusty.

**MQRC_CLUSTER_EXIT_LOAD_ERROR ( 2267 X'8DB')**

W systemie z/OSnie można załadować wyjścia obciążenia klastra.

W dzienniku systemowym zapisywany jest komunikat, a przetwarzanie jest kontynuowane tak, jakby atrybut ClusterWorkloadExit był pusty.

Multi W systemie Wiele platformjest wysyłane wywołanie MQCONN lub MQCONNX w celu nawiązania połączenia z menedżerem kolejek. Wywołanie nie powiodło się, ponieważ nie można załadować wyjścia obciążenia klastra zdefiniowanego przez atrybut ClusterWorkloadExit menedżera kolejek.

**MQRC_CLUSTER_PUT_INHIBITED ( 2268 X'8DC')**

Dla kolejki klastra jest wysyłane wywołanie MQOPEN z obowiązującą opcją MQOO_OUTPUT i MQOO_BIND_ON_OPEN . Wszystkie instancje kolejki w klastrze są obecnie zablokowane przed umieszczaniem, jeśli atrybut InhibitPut ma wartość MQQA_PUT_INHIBITED. Ponieważ nie ma dostępnych instancji kolejki do odbierania komunikatów, wywołanie MQOPEN kończy się niepowodzeniem.

Ten kod przyczyny występuje tylko wtedy, gdy spełnione są oba poniższe warunki:

- Brak lokalnej instancji kolejki. Jeśli istnieje instancja lokalna, wywołanie MQOPEN powiedzie się, nawet jeśli instancja lokalna jest zablokowana.
- Nie ma wyjścia obciążenia klastra dla kolejki lub istnieje wyjście obciążenia klastra, ale nie wybrano instancji kolejki. (Jeśli wyjście obciążenia klastra wybiera instancję kolejki, wywołanie MQOPEN powiedzie się, nawet jeśli instancja ta jest zablokowana.)

Jeśli w wywołaniu MQOPEN podano opcję MQOO_BIND_NOT_FIXED , wywołanie może zakończyć się pomyślnie, nawet jeśli wszystkie kolejki w klastrze są zablokowane. Jednak kolejne wywołanie funkcji MQPUT może zakończyć się niepowodzeniem, jeśli wszystkie kolejki są nadal zablokowane w momencie wywołania.

**MQRC_CLUSTER_RESOLUTION_ERROR ( 2189 X'88D')**

1. Wywołanie MQOPEN, MQPUTlub MQPUT1 jest wysyłane w celu otwarcia kolejki klastra lub umieszczenia w niej komunikatu. Nie można poprawnie rozstrzygnąć definicji kolejki, ponieważ wymagana jest odpowiedź z menedżera kolejek pełnego repozytorium, ale żadna z nich nie jest dostępna.
2. Dla obiektu tematu wywołano metodę MQOPEN, MQPUT, MQPUT1 lub MQSUB , podając parametr PUBSCOPE ( ALL) lub SUBSCOPE ( ALL). Nie można poprawnie rozstrzygnąć definicji tematu klastra, ponieważ wymagana jest odpowiedź z menedżera kolejek pełnego repozytorium, ale żadna odpowiedź nie jest dostępna.

**MQRC_CLUSTER_RESOURCE_ERROR ( 2269 X'8DD')**

Dla kolejki klastra wysyłane jest wywołanie MQOPEN, MQPUTlub MQPUT1 . Wystąpił błąd podczas próby użycia zasobu wymaganego do łączenia w klastry.

**MQRC_NO_DESTINATIONS_AVAILABLE ( 2270 X'8DE')**

W celu umieszczenia komunikatu w kolejce klastra wysyłane jest wywołanie MQPUT lub MQPUT1 . W momencie wywołania w klastrze nie ma już żadnych instancji kolejki. Wykonanie komendy MQPUT nie powiodło się i komunikat nie został wysłany.

Błąd może wystąpić, jeśli w wywołaniu programu MQOPEN , który otwiera kolejkę, podano parametr MQOO_BIND_NOT_FIXED lub do umieszczenia komunikatu użyto parametru MQPUT1 .

**MQRC_STOPPED_BY_CLUSTER_EXIT ( 2188 X'88C')**

Wywołanie metody MQOPEN, MQPUTlub MQPUT1 jest wysyłane w celu otwarcia lub umieszczenia komunikatu w kolejce klastra. Wyjście obciążenia klastra odrzuca wywołanie.

## z/OS Używanie i pisanie aplikacji w systemie IBM MQ for z/OS

Aplikacje IBM MQ for z/OS mogą być tworzone z programów działających w wielu różnych środowiskach. Oznacza to, że mogą korzystać z udogodnień dostępnych w więcej niż jednym środowisku.

W tej sekcji opisano narzędzia IBM MQ dostępne dla programów działających w każdym z obsługiwanych środowisk. Dodatkowo,

- Informacje na temat korzystania z opcji IBM MQ-CICS bridgezawiera sekcja Używanie języka IBM MQ z produktem CICS .
- Informacje na temat używania produktu IMS i mostu IMS zawiera sekcja “Aplikacje mostu IMS i IMS w systemie IBM MQ for z/OS” na stronie 71.

Aby uzyskać więcej informacji na temat używania i pisania aplikacji w systemie IBM MQ for z/OS, należy skorzystać z następujących odsyłaczy:

- “Funkcje IBM MQ for z/OS zależne od środowiska” na stronie 917
- “Narzędzia do debugowania, obsługa punktów synchronizacji i obsługa odtwarzania” na stronie 917
- “Interfejs IBM MQ for z/OS ze środowiskiem aplikacji” na stronie 918
- “Pisanie aplikacji z/OS UNIX System Services” na stronie 920
- “Programowanie aplikacji z kolejkami współużytkowanymi” na stronie 923

**Pojęcia pokrewne**

“Przegląd interfejsu kolejki komunikatów” na stronie 745
Sekcja zawiera informacje na temat komponentów interfejsu kolejki komunikatów (Message Queue Interface-MQI).

“Nawiązywanie i rozłączanie połączenia z menedżerem kolejek” na stronie 758
Aby można było używać usług programistycznych IBM MQ , program musi mieć połączenie z menedżerem kolejek. Ten temat zawiera informacje o nawiązywaniu połączenia z menedżerem kolejek i rozłączaniu się z nim.

“Otwieranie i zamykanie obiektów” na stronie 766
Informacje te udostępniają wgląd w otwieranie i zamykanie obiektów IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 777
Ta sekcja zawiera informacje na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 792
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Uzyskiwanie informacji o atrybutach obiektu i ustawianie ich” na stronie 876
Atrybuty są właściwościami definiującymi charakterystykę obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 879
W tej sekcji opisano sposób zatwierdzania i wycofywania wszelkich odtwarzalnych operacji pobierania i umieszczania, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy” na stronie 891
Informacje o wyzwalaczach i sposobie uruchamiania aplikacji IBM MQ za pomocą wyzwalaczy.

“Praca z funkcją MQI i klastrami” na stronie 911
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do łączenia w klastry.

“Aplikacje mostu IMS i IMS w systemie IBM MQ for z/OS” na stronie 71
Informacje te są pomocne podczas pisania aplikacji IMS przy użyciu produktu IBM MQ.

## z/OS *Funkcje IBM MQ for z/OS zależne od środowiska*

Tych informacji należy użyć podczas rozważania funkcji IBM MQ for z/OS .

Główne różnice, które należy wziąć pod uwagę między funkcjami IBM MQ w środowiskach, w których działa program IBM MQ for z/OS , to:

- Program IBM MQ for z/OS udostępnia następujące monitory wyzwalacza:
  - CKTI do użycia w środowisku CICS
  - CSQQTRMN do użytku w środowisku IMS

  Aby uruchamiać aplikacje w innych środowiskach, należy napisać własny moduł.
- Synchronizowanie przy użyciu zatwierdzania dwufazowego jest obsługiwane w środowiskach CICS i IMS . Jest on również obsługiwany w środowisku wsadowym z/OS przy użyciu usług zarządzania transakcjami i odtwarzalnego menedżera zasobów (RRS). Zatwierdzanie jednofazowe jest obsługiwane w środowisku z/OS przez samą IBM MQ .
- W środowiskach wsadowych i IMS MQI udostępnia wywołania w celu łączenia programów z menedżerem kolejek i rozłączania ich z menedżerem kolejek. Programy mogą łączyć się z więcej niż jednym menedżerem kolejek.
- System CICS może nawiązać połączenie tylko z jednym menedżerem kolejek. Może się tak zdarzyć, gdy inicjowana jest operacja CICS , jeśli nazwa podsystemu jest zdefiniowana w zadaniu uruchamiania systemu CICS . Wywołania MQI połączenia i rozłączenia są tolerowane, ale nie mają wpływu na środowisko CICS .
- Wyjście krzyżowe API umożliwia programowi ingerencję w przetwarzanie wszystkich wywołań MQI. To wyjście jest dostępne tylko w środowisku CICS .
- W systemie CICS w systemach wieloprocesorowych istnieje pewna korzyść w zakresie wydajności, ponieważ wywołania MQI mogą być wykonywane w wielu z/OS TCB. Więcej informacji na ten temat zawiera publikacja Planowanie w systemie z/OS *IBM MQ for z/OS Concepts and Planning Guide*.

Te funkcje zostały podsumowane w sekcji Tabela 130 na stronie 917.

*Tabela 130. Cechy środowiskowe z/OS*

| | **CICS** | **IMS** | **Zadania wsadowe/TSO** |
|---|---|---|---|
| Podano monitor wyzwalacza | Tak | Tak | Nie |
| zatwierdzania dwufazowe. | Tak | Tak | Tak |
| zatwierdzanie jednofazowe | Tak | Nie | Tak |
| Połącz/rozłącz wywołania MQI | Tolerowane | Tak | Tak |
| zewnętrzny program obsługi wywołań API | Tak | Nie | Nie |

**Uwaga:** Zatwierdzanie dwufazowe jest obsługiwane w środowisku Batch/TSO przy użyciu usług RRS.

## z/OS *Narzędzia do debugowania, obsługa punktów synchronizacji i obsługa odtwarzania*

Te informacje umożliwiają poznanie funkcji debugowania programów, obsługi punktów synchronizacji i obsługi odtwarzania.

## Narzędzia do debugowania programów

IBM MQ for z/OS udostępnia narzędzie śledzenia, którego można użyć do debugowania programów we wszystkich środowiskach.

Dodatkowo w środowisku CICS można użyć:

- Narzędzie CICS Execution Diagnostic Facility (CEDF)
- Transakcja sterowania śledzeniem CICS (CETR)
- Wyjście funkcji API języka IBM MQ for z/OS

Na platformie z/OS można użyć dowolnego dostępnego interaktywnego narzędzia do debugowania, które jest obsługiwane przez używany język programowania.

## Obsługa punktu synchronizacji

Synchronizowanie początku i końca jednostek pracy jest konieczne w środowisku przetwarzania transakcji, aby można było bezpiecznie korzystać z przetwarzania transakcji.

Jest to w pełni obsługiwane przez produkt IBM MQ for z/OS w środowiskach CICS i IMS . Pełna obsługa oznacza współpracę między menedżerami zasobów, dzięki czemu jednostki pracy mogą być zatwierdzane lub wycofane razem, pod kontrolą systemu CICS lub IMS. Przykładami menedżerów zasobów są Db2, CICS File Control, IMSi IBM MQ for z/OS.

Aplikacje wsadowe z/OS mogą używać wywołań IBM MQ for z/OS w celu nadania funkcji zatwierdzania jednofazowego. Oznacza to, że zdefiniowany przez aplikację zestaw operacji kolejki może zostać zatwierdzony lub wycofany bez odwoływania się do innych menedżerów zasobów.

Zatwierdzanie dwufazowe jest również obsługiwane w środowisku wsadowym systemu z/OS przy użyciu usług zarządzania transakcjami i odtwarzalnego menedżera zasobów (RRS). Więcej informacji na ten temat zawiera sekcja Punkty synchronizacji w z/OS aplikacjach wsadowych.

## Obsługa odtwarzania

Jeśli połączenie między menedżerem kolejek i systemem CICS lub IMS zostanie zerwane podczas transakcji, niektóre jednostki pracy mogą nie zostać pomyślnie wycofane.

Jednak te jednostki pracy są rozstrzygane przez menedżer kolejek (pod kontrolą menedżera punktów synchronizacji) podczas ponownego nawiązywania połączenia z systemem CICS lub IMS .

## z/OS *Interfejs IBM MQ for z/OS ze środowiskiem aplikacji*

Aby umożliwić aplikacjom działającym w różnych środowiskach wysyłanie i odbieranie komunikatów za pośrednictwem sieci kolejkowania komunikatów, produkt IBM MQ for z/OS udostępnia *adapter* dla każdego obsługiwanego środowiska.

Te adaptery są interfejsem między aplikacjami i podsystemami IBM MQ for z/OS . Umożliwiają one programom korzystanie z interfejsu MQI.

### z/OS *Adapter zadania wsadowego*

Te informacje umożliwiają zapoznanie się z adapterem zadania wsadowego i obsługiwanym przez niego protokołem zatwierdzania.

*Adapter wsadowy* zapewnia dostęp do zasobów systemu IBM MQ for z/OS dla programów działających w:

- Tryb zadania (TCB)
- Stan problemu lub nadzorcy
- Tryb sterowania podstawową przestrzenią adresową

Programy nie mogą być w trybie międzypamięciowym.

Połączenia między aplikacjami i programem IBM MQ for z/OS są nawiązywane na poziomie zadania. Adapter udostępnia pojedynczy wątek połączenia z bloku kontrolnego zadania aplikacji (TCB) do bazy danych IBM MQ for z/OS.

Adapter obsługuje protokół zatwierdzania jednofazowego dla zmian w zasobach, których właścicielem jest IBM MQ for z/OS ; nie obsługuje protokołów zatwierdzania wielofazowego.

z/OS *Adapter zadania wsadowego RRS*

Ten temat zawiera informacje o adapterze zadań wsadowych RRS i dwóch adapterach zadań wsadowych RRS udostępnianych przez produkt IBM MQ.

Adapter usług RRS (Transaction Management and recoverable resource manager services):

- Używa z/OS RRS do kontroli zatwierdzania.
- Obsługuje jednoczesne połączenia z wieloma podsystemami IBM MQ działającymi w jednej instancji z/OS z jednego zadania.
- Provides z/OS-wide coordinated commitment control (using z/OS RRS) for recoverable resources accessed through z/OS RRS-compliant recoverable managers for:
  - Aplikacje, które łączą się z produktem IBM MQ przy użyciu adaptera RRS Batch.
  - Db2-procedury składowane wykonywane w przestrzeni adresowej Db2-stored procedures, która jest zarządzana przez menedżer obciążenia (WLM) w systemie z/OS.
- Obsługuje możliwość przełączania wątku wsadowego IBM MQ między rekordami wykonania instrukcji testowania.

Produkt IBM MQ for z/OS udostępnia dwa adaptery zadań wsadowych RRS:

**CSQBRSTB**
Ten adapter wymaga zmiany dowolnej instrukcji MQCMIT na SRRCMIT i każdej instrukcji MQBACK na SRRBACK w aplikacji IBM MQ . (Jeśli w aplikacji połączonej z CSQBRSTB zostanie zakodowana komenda MQCMIT lub MQBACK, wystąpi błąd MQRC_ENVIRONMENT_ERROR).

**CSQBRRSI**
Ten adapter umożliwia aplikacji IBM MQ korzystanie z opcji MQCMIT i MQBACK lub SRRCMIT i SRRBACK.

**Uwaga:** CSQBRSTB i CSQBRRSI są dostarczane z atrybutami łączącymi AMODE (31) RMODE (ANY). Jeśli aplikacja załaduje kod pośredniczący poniżej wiersza 16 MB, najpierw należy ponownie dowiązanie kodu pośredniczącego do RMODE (24).

## Migracja

Istniejące aplikacje Batch/TSO IBM MQ można zmigrować w celu użycia koordynacji RRS z niewielką liczbą zmian lub bez nich.

W przypadku łączenia aplikacji IBM MQ z adapterem CSQBRRSI, za pomocą punktu synchronizacji MQCMIT i MQBACK jednostka pracy w produkcie IBM MQ i we wszystkich innych menedżerach zasobów obsługujących usługi RRS. Po połączeniu aplikacji IBM MQ z adapterem CSQBRSTB należy zmienić wartość MQCMIT na SRRCMIT i MQBACK na SRRBACK. To drugie podejście jest preferowane; wyraźnie wskazuje, że punkt synchronizacji nie jest ograniczony tylko do zasobów IBM MQ .

z/OS *Adapter IMS*

Jeśli używany jest adapter IMS z systemu IBM MQ for z/OS , należy upewnić się, że IMS może uzyskać wystarczającą ilość pamięci, aby pomieścić komunikaty o długości do 100 MB.

## Uwaga dla użytkowników

*Adapter IMS* zapewnia dostęp do zasobów IBM MQ for z/OS dla:

- Programy przetwarzania komunikatów w trybie z połączeniem (MPP)

- Programy krótkiej ścieżki interaktywnej (IFP)
- Wsadowe programy przetwarzania komunikatów (BMP)

Aby można było korzystać z tych zasobów, programy muszą działać w trybie zadania (TCB) i w stanie problemu; nie mogą działać w trybie międzypamięciowym ani w trybie rejestracji dostępu.

Adapter udostępnia wątek połączenia z bloku kontrolnego zadania aplikacji (TCB) do bazy danych IBM MQ. Adapter obsługuje protokół zatwierdzania dwufazowego w przypadku zmian w zasobach należących do produktu IBM MQ for z/OS, gdzie IMS działa jako koordynator punktu synchronizacji.

Adapter udostępnia również program monitora wyzwalacza, który może uruchamiać programy automatycznie po spełnieniu określonych warunków wyzwalania w kolejce. Więcej informacji na ten temat zawiera “Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy” na stronie 891.

Jeśli piszesz programy wsadowe DL/I, postępuj zgodnie ze wskazówkami podanymi w tym temacie dla programów wsadowych systemu z/OS .

### z/OS *Pisanie aplikacji z/OS UNIX System Services*

Adapter zadania wsadowego obsługuje połączenia menedżera kolejek z przestrzeni adresowych zadań wsadowych i TSO.

W przypadku wsadowej przestrzeni adresowej adapter obsługuje połączenia z wielu obiektów TCB w tej przestrzeni adresowej w następujący sposób:

- Każda baza TCB może łączyć się z wieloma menedżerami kolejek za pomocą wywołania MQCONN lub MQCONNX (ale baza TCB może mieć tylko jedną instancję połączenia z określonym menedżerem kolejek w danym momencie).
- Wiele obiektów TCB może nawiązać połączenie z tym samym menedżerem kolejek (ale uchwyt menedżera kolejek zwrócony w dowolnym wywołaniu MQCONN lub MQCONNX jest powiązany z wydającym TCB i nie może być używany przez żadną inną bazę TCB).

z/OS UNIX System Services obsługuje dwa typy wywołań pthread_create:

1. Wątki o dużej wadze należy uruchomić po jednym dla każdej bazy TCB, która jest przyłączona i odcięta na początku i na końcu wątku przez z/OS.
2. Wątki o średniej wadze, uruchamiane po jednym dla każdego TCB, ale TCB może być jednym z puli długotrwałych TCB. Aplikacja musi wykonać wszystkie niezbędne procedury czyszczące, ponieważ jeśli jest połączona z serwerem, domyślne zakończenie wątku, które może być udostępniane przez serwer w momencie zakończenia zadania (TCB), **nie** jest zawsze sterowane.

Wątki lekkie nie są obsługiwane. (Jeśli aplikacja tworzy wątki trwałe, które rozsyłają własne żądania pracy, ***aplikacja*** jest odpowiedzialna za czyszczenie zasobów przed rozpoczęciem następnego żądania pracy).

Produkt IBM MQ for z/OS obsługuje wątki z/OS UNIX System Services przy użyciu adaptera wsadowego w następujący sposób:

1. Wątki o dużej wadze są w pełni obsługiwane jako połączenia wsadowe. Każdy wątek działa w swojej własnej bazie TCB, która jest dołączana i odłączana podczas uruchamiania i kończenia wątku. Jeśli wątek zostanie zakończony przed wywołaniem MQDISC, program IBM MQ for z/OS wykonuje standardowe czyszczenie zadania, które obejmuje zatwierdzanie wszystkich zaległych jednostek pracy, jeśli wątek został zakończony normalnie, lub wycofywanie go, jeśli wątek został zakończony nieprawidłowo.
2. Wątki o średniej wadze są w pełni obsługiwane, ale jeśli baza TCB ma być ponownie wykorzystywana przez inny wątek, aplikacja musi upewnić się, że wywołanie MQDISC poprzedzone przez MQCMIT lub MQBACK zostało wysłane przed następnym uruchomieniem wątku. Oznacza to, że jeśli aplikacja ustanowiła procedurę obsługi przerwania programu, a następnie aplikacja została zakończona awaryjnie, procedura obsługi przerwania musi wywołać wywołania MQCMIT i MQDISC przed ponownym wykorzystaniem bazy TCB dla innego wątku.

**Uwaga:** Modele wielowątkowości **nie** obsługują dostępu do wspólnych zasobów IBM MQ z wielu wątków.

## z/OS *Wyjście przecięcia interfejsu API dla systemu z/OS*

Ten temat zawiera informacje dotyczące interfejsu programistycznego, które są zależne od produktu.

Wyjście to punkt w kodzie dostarczonym przez IBM, w którym można uruchomić własny kod. Produkt IBM MQ for z/OS udostępnia *wyjście przekraczające interfejs API* , którego można użyć do przechwytywania wywołań interfejsu MQI i monitorowania lub modyfikowania funkcji wywołań interfejsu MQI. W tej sekcji opisano sposób korzystania z wyjścia przekraczania granicy interfejsu API oraz przykładowy program obsługi wyjścia dostarczany z produktem IBM MQ for z/OS.

Ta sekcja ma zastosowanie tylko do użytkowników systemu CICS TS V3.1 i wcześniejszych. Użytkownicy systemu CICS TS V3.2 i nowszych powinni zapoznać się z sekcją CICS Integracja z produktem IBM MQ w dokumentacji produktu CICS .

### Uwaga

Wyjście przejścia przez interfejs API jest wywoływane tylko przez adapter CICS produktu IBM MQ for z/OS. Program obsługi wyjścia działa w przestrzeni adresowej CICS .

### z/OS *Pisanie własnego programu obsługi wyjścia*

Można użyć przykładowego programu obsługi wyjścia przecięcia interfejsu API (CSQCAPX), który jest dostarczany z produktem IBM MQ for z/OS jako środowisko dla własnego programu.

Zostało to opisane w sekcji “Przykładowy program obsługi wyjścia dla komunikacji między interfejsami API, CSQCAPX” na stronie 922.

Podczas pisania programu obsługi wyjścia, aby znaleźć nazwę wywołania MQI wydanego przez aplikację, należy sprawdzić pole *ExitCommand* w strukturze MQXP. Aby znaleźć liczbę parametrów w wywołaniu, należy sprawdzić pole *ExitParmCount* . Można użyć 16-bajtowego pola *ExitUserArea* , aby zapisać adres dowolnej dynamicznej pamięci masowej uzyskanej przez aplikację. To pole jest zachowywane między wywołaniami wyjścia i ma taki sam czas życia, jak zadanie CICS .

Jeśli używany jest serwer CICS Transaction Server V3.2, należy napisać program obsługi wyjścia jako wątkowo bezpieczny i zadeklarować program obsługi wyjścia jako wątkowo bezpieczny. Jeśli używane są wcześniejsze wersje systemu CICS , zaleca się również napisanie i zadeklarowanie programów obsługi wyjścia jako wątkowo bezpiecznych, aby były gotowe do migracji do produktu CICS Transaction Server V3.2.

Program obsługi wyjścia może zablokować wykonanie wywołania MQI przez zwrócenie wartości MQXCC_SUPPRESS_FUNCTION lub MQXCC_SKIP_FUNCTION w polu *ExitResponse* . Aby umożliwić wykonanie wywołania (i ponowne wywołanie programu obsługi wyjścia po zakończeniu wywołania), program obsługi wyjścia musi zwrócić kod MQXCC_OK.

Program obsługi wyjścia wywołany po wywołaniu MQI może sprawdzać i modyfikować kody zakończenia i przyczyny ustawione przez wywołanie.

### Użycie notatek

Poniżej przedstawiono kilka ogólnych punktów, które należy wziąć pod uwagę podczas pisania programu obsługi wyjścia:

- Ze względu na wydajność, należy napisać program w języku asembler. Jeśli jest on zapisywany w dowolnym innym języku obsługiwanym przez program IBM MQ for z/OS, należy udostępnić własny plik definicji danych.
- Link-edytuj swój program jako AMODE (31) i RMODE (ANY).
- Aby zdefiniować blok parametrów wyjścia dla programu, należy użyć makra w języku asemblera, CMQXPA.
- Parametr CONCURRENCY (THREADSAFE) należy określić podczas definiowania programu obsługi wyjścia i wszystkich programów wywoływanych przez program obsługi wyjścia.

- Jeśli używana jest funkcja ochrony pamięci masowej produktu CICS Transaction Server for z/OS , program musi działać z kluczem wykonywania CICS . Oznacza to, że należy określić parametr EXECKEY ( CICS ) podczas definiowania zarówno programu obsługi wyjścia, jak i programów, do których przekazuje sterowanie. Informacje na temat programów obsługi wyjścia systemu CICS i funkcji ochrony pamięci masowej systemu CICS zawiera podręcznik *CICS Customization Guide*.
- Program może korzystać ze wszystkich funkcji API (na przykład IMS, Db2i CICS ). który może być używany przez program obsługi wyjścia użytkownika powiązany z zadaniem CICS . Może również używać dowolnych wywołań MQI z wyjątkiem MQCONN, MQCONNX i MQDISC. Jednak wywołania MQI w programie obsługi wyjścia nie wywołują programu obsługi wyjścia po raz drugi.
- Program może wydawać komendy EXEC CICS SYNCPOINT lub EXEC CICS SYNCPOINT ROLLBACK. Jednak te komendy zatwierdzają lub wycofują ***wszystkie*** aktualizacje wykonane przez zadanie do momentu użycia wyjścia, dlatego ich użycie nie jest zalecane.
- Program musi zostać zakończony za pomocą komendy EXEC CICS RETURN. Nie może on przekazywać sterowania za pomocą komendy XCTL.
- Wyjścia są zapisywane jako rozszerzenia kodu IBM MQ for z/OS . Upewnij się, że wyjście nie zakłóca żadnych programów lub transakcji IBM MQ for z/OS używających interfejsu MQI. Zazwyczaj są one oznaczone przedrostkiem CSQ lub CK.
- Jeśli baza danych CSQCAPX jest zdefiniowana w systemie CICS, system CICS próbuje załadować program obsługi wyjścia, gdy CICS łączy się z IBM MQ for z/OS. Jeśli ta próba się powiedzie, do panelu CKQC lub do konsoli systemowej zostanie wysłany komunikat CSQC301I . Jeśli ładowanie nie powiedzie się (na przykład, jeśli moduł ładowania nie istnieje w żadnej z bibliotek w konkatenacji DFHRPL), do panelu CKQC lub konsoli systemowej zostanie wysłany komunikat CSQC315 .
- Ponieważ parametry w obszarze komunikacyjnym są adresami, program obsługi wyjścia musi być zdefiniowany jako lokalny w systemie CICS (tj. nie jako program zdalny).

z/OS *Przykładowy program obsługi wyjścia dla komunikacji między interfejsami API, CSQCAPX*

Przykładowy program obsługi wyjścia jest dostarczany jako program w języku asembler. Plik źródłowy (CSQCAPX) jest dostarczany w bibliotece **thlqual**.SCSQASMS (gdzie **thlqual** jest kwalifikatorem wysokiego poziomu używanym przez instalację). Ten plik źródłowy zawiera pseudokod opisujący logikę programu.

Program przykładowy zawiera kod inicjowania i układ, którego można użyć podczas pisania własnych programów obsługi wyjścia.

Przykład pokazuje, w jaki sposób:

- Skonfiguruj blok parametrów wyjścia
- Adresowanie bloków parametrów wywołania i wyjścia
- Określ, dla którego wywołania MQI jest wywoływane wyjście
- Określ, czy wyjście jest wywoływane przed, czy po przetworzeniu wywołania MQI
- Umieszczenie komunikatu w kolejce pamięci tymczasowej systemu CICS
- Użyj makra DFHEIENT do dynamicznego uzyskiwania pamięci masowej, aby zachować wielobieżność
- Użyj DFHEIBLK dla bloku kontrolnego interfejsu exec systemu CICS
- Warunki błędu pułapki
- Zwróć sterowanie do programu wywołującego

## Projekt przykładowego programu obsługi wyjścia

Przykładowy program obsługi wyjścia zapisuje komunikaty w kolejce pamięci tymczasowej CICS (CSQ1EXIT) w celu wyświetlenia operacji wyjścia.

Komunikaty wskazują, czy wyjście jest wywoływane przed, czy po wywołaniu MQI. Jeśli wyjście jest wywoływane po wywołaniu, komunikat zawiera kod zakończenia i kod przyczyny zwrócony przez

wywołanie. W tym przykładzie używane są stałe nazwane z makra CMQXPA w celu sprawdzenia typu pozycji (czyli przed lub po wywołaniu).

Przykład nie wykonuje żadnej funkcji monitorowania, ale po prostu umieszcza komunikaty ze znacznikiem czasu w kolejce CICS , wskazując typ przetwarzanego wywołania. Wskazuje to wydajność interfejsu MQI, a także poprawne działanie programu obsługi wyjścia.

**Uwaga:** Przykładowy program obsługi wyjścia wywołuje sześć wywołań EXEC CICS dla każdego wywołania MQI wykonanego podczas działania programu. Jeśli używany jest ten program obsługi wyjścia, wydajność systemu IBM MQ for z/OS jest obniżona.

z/OS *Przygotowywanie i używanie wyjścia przecięcia interfejsu API*

Przykładowe wyjście jest dostarczane tylko w formie źródłowej.

Aby użyć przykładowego programu obsługi wyjścia lub napisanego programu obsługi wyjścia, należy utworzyć bibliotekę ładowania, tak jak w przypadku każdego innego programu CICS , zgodnie z opisem w sekcji "Budowanie aplikacji CICS w produkcie z/OS" na stronie 1057.

- W przypadku produktów CICS Transaction Server for z/OS i CICS for MVS/ESApodczas aktualizowania zestawu danych CSD ( CICS system definition) wymagane definicje znajdują się w elemencie **thlqual**.SCSQPROC (CSQ4B100).

  **Uwaga:** W definicjach używany jest przyrostek MQ. Jeśli ten przyrostek jest już używany w przedsiębiorstwie, należy go zmienić przed etapem składania.

Jeśli używane są domyślne definicje programu CICS , program obsługi wyjścia CSQCAPX jest instalowany w stanie ***wyłączonym*** . Wynika to z faktu, że użycie programu obsługi wyjścia może spowodować znaczne zmniejszenie wydajności.

Aby tymczasowo aktywować wyjście przekraczania granicy interfejsu API:

1. Wydaj komendę **CEMT S PROGRAM(CSQCAPX) ENABLED** z głównego terminalu systemu CICS .
2. Uruchom transakcję CKQC i użyj opcji 3 w menu rozwijanym Połączenie, aby zmienić status wyjścia przekraczania API na **Włączone**.

Aby uruchomić program IBM MQ for z/OS z trwale włączonym wyjściem przejścia między interfejsami API, a serwerami CICS Transaction Server for z/OS i CICS for MVS/ESA, wykonaj jedną z następujących czynności:

- Zmień definicję CSQCAPX w elemencie CSQ4B100, zmieniając STATUS (DISABLED) na STATUS (ENABLED). Definicję CSD CICS można zaktualizować za pomocą programu wsadowego DFHCSDUP dostarczonego przez CICS.
- Zmień definicję CSQCAPX w grupie CSQCAT1 , zmieniając status z DISABLED na ENABLED.

W obu przypadkach należy ponownie zainstalować grupę. Można to zrobić, uruchamiając na zimno system CICS lub używając transakcji CICS CEDA do reinstalacji grupy w czasie, gdy działa program CICS .

**Uwaga:** Użycie programu CEDA może spowodować błąd, jeśli dowolna z pozycji w grupie jest obecnie używana.

Koniec informacji o interfejsie programistycznym objętych szczególną ochroną.

## z/OS *Programowanie aplikacji z kolejkami współużytkowanymi*

Ten temat zawiera informacje o niektórych czynnikach, które należy wziąć pod uwagę podczas projektowania nowych aplikacji korzystających z kolejek współużytkowanych oraz podczas migrowania istniejących aplikacji do środowiska kolejek współużytkowanych.

z/OS *Przekształcanie aplikacji do postaci szeregowej*

Niektóre typy aplikacji mogą wymagać, aby komunikaty były pobierane z kolejki w dokładnie tej samej kolejności, w jakiej zostały odebrane w kolejce.

Na przykład, jeśli program IBM MQ jest używany do śledzenia aktualizacji bazy danych w systemie zdalnym, komunikat opisujący aktualizację rekordu musi zostać przetworzony po komunikacie opisującym

wstawienie tego rekordu. W lokalnym środowisku kolejkowania jest to często osiągane przez aplikację, która otrzymuje komunikaty otwierające kolejkę za pomocą opcji MQOO_INPUT_EXCLUSIVE, co uniemożliwia innym aplikacjom przetwarzanie kolejki w tym samym czasie.

Produkt IBM MQ umożliwia aplikacjom otwieranie kolejek współużytkowanych wyłącznie w ten sam sposób. Jeśli jednak aplikacja działa z partycji kolejki (na przykład wszystkie aktualizacje bazy danych znajdują się w tej samej kolejce, ale te dla tabeli A mają identyfikator korelacji A, a dla tabeli B identyfikator korelacji B), a aplikacje chcą współbieżnie otrzymywać komunikaty dla aktualizacji tabeli A i aktualizacji tabeli B, prosty mechanizm wyłącznego otwierania kolejki nie jest możliwy.

Jeśli ten typ aplikacji ma korzystać z wysokiej dostępności kolejek współużytkowanych, można zdecydować, że inna instancja aplikacji, która uzyskuje dostęp do tych samych kolejek współużytkowanych, uruchomiona w dodatkowym menedżerze kolejek, powinna przejąć kontrolę w przypadku awarii podstawowej aplikacji pobierającej lub menedżera kolejek.

Jeśli podstawowy menedżer kolejek ulegnie awarii, zostaną wykonane dwie czynności:

- Odtwarzanie węzła sieci kolejki współużytkowanej zapewnia, że wszystkie niekompletne aktualizacje z aplikacji podstawowej zostaną zakończone lub wycofane.
- Kolejka jest przetwarzana przez aplikację dodatkową.

Aplikacja dodatkowa może zostać uruchomiona przed wykonaniem wszystkich niekompletnych jednostek pracy, co może prowadzić do pobrania komunikatów poza kolejnością przez aplikację dodatkową. Aby rozwiązać ten problem, aplikacja może być *aplikacją przekształconą do postaci szeregowej*.

Aplikacja przekształcona do postaci szeregowej używa wywołania MQCONNX w celu nawiązania połączenia z menedżerem kolejek i określenia znacznika połączenia podczas nawiązywania połączenia, który jest unikalny dla tej aplikacji. Wszystkie jednostki pracy wykonywane przez aplikację są oznaczone znacznikiem połączenia. IBM MQ zapewnia, że jednostki pracy w grupie współużytkowania kolejek z tym samym znacznikiem połączenia są przekształcane do postaci szeregowej (zgodnie z opcjami przekształcania do postaci szeregowej w wywołaniu MQCONNX ).

Oznacza to, że jeśli aplikacja podstawowa używa wywołania MQCONNX ze znacznikiem połączenia `Database shadow retriever`, a dodatkowa aplikacja przejęcia próbuje użyć wywołania MQCONNX z identycznym znacznikiem połączenia, aplikacja dodatkowa nie może połączyć się z drugim IBM MQ , dopóki nie zostaną zakończone wszystkie zaległe podstawowe jednostki pracy, w tym przypadku przez odtwarzanie równorzędne.

Należy rozważyć użycie techniki aplikacji przekształconej do postaci szeregowej w przypadku aplikacji, które zależą od dokładnej kolejności komunikatów w kolejce. W szczególności:

- Aplikacje, które nie mogą być restartowane po awarii aplikacji lub menedżera kolejek, dopóki nie zostaną zakończone wszystkie operacje zatwierdzania i wycofywania dla poprzedniego wykonania aplikacji.

  W tym przypadku technika aplikacji przekształconej do postaci szeregowej ma zastosowanie tylko wtedy, gdy aplikacja działa w punkcie synchronizacji.
- Aplikacje, które nie mogą być uruchamiane, gdy jest już uruchomiona inna instancja tej samej aplikacji.

  W tym przypadku technika aplikacji przekształconej do postaci szeregowej jest wymagana tylko wtedy, gdy aplikacja nie może otworzyć kolejki na wyłączne wejście.

**Uwaga:** Produkt IBM MQ gwarantuje zachowanie sekwencji komunikatów tylko wtedy, gdy spełnione są określone kryteria. Są one opisane w opisie komendy MQGET.

z/OS *Aplikacje, które nie są odpowiednie do użycia z kolejkami współużytkowanymi*

Niektóre funkcje produktu IBM MQ nie są obsługiwane w przypadku korzystania z kolejek współużytkowanych, dlatego aplikacje korzystające z tych funkcji nie są odpowiednie dla środowiska kolejek współużytkowanych.

Podczas projektowania aplikacji z kolejką współużytkowaną należy wziąć pod uwagę następujące kwestie:

- Indeksowanie kolejek jest ograniczone w przypadku kolejek współużytkowanych. Jeśli do wybrania komunikatu, który ma zostać otrzymany z kolejki, ma zostać użyty identyfikator komunikatu lub

identyfikator korelacji, kolejka powinna być poindeksowana z poprawną wartością. Jeśli komunikaty są wybierane tylko na podstawie identyfikatora komunikatu, kolejka wymaga indeksu typu MQIT_MSG_ID (chociaż można również użyć atrybutu MQIT_NONE). Jeśli komunikaty są wybierane tylko na podstawie identyfikatora korelacji, kolejka musi mieć indeks typu MQIT_CORREL_ID.

- Tymczasowych kolejek dynamicznych nie można używać jako kolejek współużytkowanych. Można jednak użyć trwałych kolejek dynamicznych. Modele współużytkowanych kolejek dynamicznych mają parametr DEFTYPE o wartości SHAREDYN (współużytkowany dynamiczny), chociaż są one tworzone i niszczone w taki sam sposób, jak trwałe kolejki dynamiczne (PERMDYN).

z/OS *Podejmowanie decyzji o współużytowaniu kolejek innych niż kolejki aplikacji*

Informacje te należy wykorzystać podczas rozważania współużytkowania kolejek innych niż kolejki aplikacji.

Istnieją kolejki inne niż kolejki aplikacji, które można rozważyć do współużytkowania:

**Kolejki inicjująca**

Jeśli zdefiniowano współużytkowaną kolejkę inicjującą, monitor wyzwalacza nie musi być uruchomiony w każdym menedżerze kolejek w grupie współużytkowania kolejek, o ile działa co najmniej jeden monitor wyzwalacza. (Można również użyć współużytkowanej kolejki inicjującej, nawet jeśli w każdym menedżerze kolejek w grupie współużytkowania kolejek jest uruchomiony monitor wyzwalacza).

Jeśli używana jest współużytkowana kolejka aplikacji i używany jest wyzwalacz typu EVERY (lub wyzwalacz typu FIRST z małym odstępem czasu wyzwalacza, który zachowuje się jak wyzwalacz typu EVERY), kolejka inicjująca musi zawsze być kolejką współużytkowaną. Więcej informacji na temat używania współużytkowanej kolejki inicjującej zawiera sekcja Tabela 131 na stronie 926.

**SYSTEM.* kolejki**

Można zdefiniować SYSTEM.ADMIN.* Kolejki używane do przechowywania komunikatów zdarzeń jako kolejki współużytkowane. Może to być przydatne do sprawdzania równoważenia obciążenia w przypadku wystąpienia wyjątku. Każdy komunikat zdarzenia utworzony przez program IBM MQ zawiera identyfikator korelacji wskazujący, który menedżer kolejek go wygenerował.

Należy zdefiniować SYSTEM SYSTEM.QSG.* Kolejki używane na potrzeby kolejek współużytkowanych i kolejkowania wewnątrz grupy jako kolejki współużytkowane.

Można również zmienić definicje systemu SYSTEM.DEFAULT.LOCAL.QUEUE do współużytkowania lub zdefiniuj własną domyślną definicję kolejki współużytkowanej. Więcej informacji na ten temat zawiera sekcja Definiowanie obiektów systemowych dla systemu IBM MQ for z/OS .

Nie można zdefiniować żadnego innego SYSTEM.* kolejki jako kolejki współużytkowane.

z/OS *Migrowanie istniejących aplikacji w celu użycia kolejek współużytkowanych*

Kody przyczyny, wyzwalanie i wywołanie funkcji API MQINQ mogą działać inaczej w środowisku kolejki współużytkowanej.

Informacje na temat migrowania istniejących kolejek do kolejek współużytkowanych zawiera sekcja Migracja kolejek niewspółużytkowanych do kolejek współużytkowanych .

Podczas migrowania istniejących aplikacji należy wziąć pod uwagę następujące kwestie, które mogą działać w inny sposób w środowisku kolejek współużytkowanych:

**Kody przyczyny**

Podczas migrowania istniejących aplikacji w celu korzystania z kolejek współużytkowanych należy sprawdzić, czy nie pojawiły się nowe kody przyczyny.

**Wyzwalanie**

Jeśli używana jest współużytkowana kolejka aplikacji, wyzwalanie działa tylko w przypadku zatwierdzonych komunikatów (w przypadku niewspółużytkowanej kolejki aplikacji wyzwalanie działa w przypadku wszystkich komunikatów).

Jeśli do uruchamiania aplikacji używane jest wyzwalanie, można użyć współużytkowanej kolejki inicjującej. Tabela 131 na stronie 926 opisuje, co należy wziąć pod uwagę przy podejmowaniu decyzji o typie kolejki inicjującej, która ma być używana.

*Tabela 131. Kiedy używać współużytkowanej kolejki inicjującej*

| | **Niewspółużytkowana kolejka aplikacji** | **Współużytkowana kolejka aplikacji** |
|---|---|---|
| **Niewspółużytkowana kolejka inicjowania** | Podobnie jak w poprzednich wersjach. | Jeśli używany jest wyzwalacz typu FIRST lub DEPTH, można użyć niewspółużytkowanej kolejki inicjującej ze współużytkowaną kolejką aplikacji. Mogą być generowane dodatkowe komunikaty wyzwalacza, ale ta konfiguracja jest odpowiednia do wyzwalania długotrwałych aplikacji (takich jak CICS bridge) i zapewnia wysoką dostępność.<br><br>W przypadku wyzwalacza typu FIRST lub DEPTH komunikat wyzwalacza wyzwala instancję aplikacji w każdym menedżerze kolejek, w którym działa monitor wyzwalacza i w którym kolejka aplikacji nie jest jeszcze otwarta do wprowadzania. Dla każdego menedżera kolejek generowany jest jeden komunikat wyzwalacza. Jeśli dla niewspółużytkowanej lokalnej kolejki inicjującej działa więcej niż jeden monitor wyzwalacza, w konkretnym menedżerze kolejek będą one konkurować o przetwarzanie komunikatu. |

| *Tabela 131. Kiedy używać współużytkowanej kolejki inicjującej (kontynuacja)* | | |
|---|---|---|
| | **Niewspółużytkowana kolejka aplikacji** | **Współużytkowana kolejka aplikacji** |
| **Współużytkowana kolejka inicjowania** | Nie należy używać współużytkowanej kolejki inicjującej z niewspółużytkowaną kolejką aplikacji. | W przypadku wyzwalacza typu EVERY, gdy aplikacja umieszcza komunikat we współużytkowanej kolejce aplikacji, menedżer kolejek umieszczający określa, które menedżery kolejek są interesujące w wyzwalaczu-każde zdarzenie i wysyła powiadomienie do jednego z tych menedżerów kolejek. W przypadku powiadamianego menedżera kolejek działaniem wynikowym jest wygenerowanie komunikatu wyzwalacza do kolejki inicjującej.<br><br>**Uwaga:** W przypadku współużytkowanej kolejki aplikacji, która ma typ wyzwalacza EVERY, należy użyć współużytkowanej kolejki inicjującej. W przeciwnym razie w pewnych okolicznościach mogą zostać utracone komunikaty wyzwalacza, na przykład w przypadku niepowodzenia menedżera kolejek.<br><br>W przypadku wyzwalacza typu FIRST lub DEPTH jeden komunikat wyzwalacza jest generowany przez każdy menedżer kolejek, który ma otwartą nazwaną kolejkę inicjującą dla wejścia.<br><br>**Uwaga:** W przypadku wyzwalacza typu FIRST lub DEPTH, jeśli jedna instancja monitora wyzwalacza jest zajęta, może to spowodować, że mniej zajęte monitory wyzwalacza będą przetwarzać więcej niż jeden komunikat wyzwalacza ze współużytkowanej kolejki inicjującej. Z tego powodu dla danego menedżera kolejek można uruchomić wiele instancji aplikacji serwera. Należy zauważyć, że te wiele instancji jest uruchamianych w wyniku przetwarzania wielu komunikatów wyzwalacza. Zwykle w przypadku wyzwalacza typu FIRST lub DEPTH, jeśli instancja aplikacji już obsługuje kolejkę aplikacji, inny komunikat wyzwalacza nie zostanie wygenerowany przez menedżer kolejek, z którym aplikacja jest połączona. |

**MQINQ**

Jeśli wywołanie MQINQ jest używane do wyświetlania informacji o kolejce współużytkowanej, wartości liczby wywołań MQOPEN , które mają otwartą kolejkę dla wejścia i wyjścia, odnoszą się tylko do menedżera kolejek, który wywołał to wywołanie. Nie są generowane żadne informacje o innych menedżerach kolejek w grupie współużytkowania kolejek, które mają otwartą kolejkę.

## z/OS Aplikacje mostu IMS i IMS w systemie IBM MQ for z/OS

Informacje te są pomocne podczas pisania aplikacji IMS przy użyciu produktu IBM MQ.

- Informacje na temat używania punktów synchronizacji i wywołań MQI w aplikacjach IMS zawiera sekcja “Pisanie aplikacji IMS przy użyciu programu IBM MQ” na stronie 72.
- Informacje na temat pisania aplikacji używających mostu IBM MQ - IMS zawiera sekcja “Pisanie aplikacji mostu IMS” na stronie 76.

Aby uzyskać więcej informacji na temat aplikacji mostu IMS i IMS w systemie IBM MQ for z/OS, należy skorzystać z następujących odsyłaczy:

- “Pisanie aplikacji IMS przy użyciu programu IBM MQ” na stronie 72
- “Pisanie aplikacji mostu IMS” na stronie 76

**Pojęcia pokrewne**

“Przegląd interfejsu kolejki komunikatów” na stronie 745
Sekcja zawiera informacje na temat komponentów interfejsu kolejki komunikatów (Message Queue Interface-MQI).

“Nawiązywanie i rozłączanie połączenia z menedżerem kolejek” na stronie 758
Aby można było używać usług programistycznych IBM MQ , program musi mieć połączenie z menedżerem kolejek. Ten temat zawiera informacje o nawiązywaniu połączenia z menedżerem kolejek i rozłączaniu się z nim.

“Otwieranie i zamykanie obiektów” na stronie 766
Informacje te udostępniają wgląd w otwieranie i zamykanie obiektów IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 777
Ta sekcja zawiera informacje na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 792
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Uzyskiwanie informacji o atrybutach obiektu i ustawianie ich” na stronie 876
Atrybuty są właściwościami definiującymi charakterystykę obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 879
W tej sekcji opisano sposób zatwierdzania i wycofywania wszelkich odtwarzalnych operacji pobierania i umieszczania, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy” na stronie 891
Informacje o wyzwalaczach i sposobie uruchamiania aplikacji IBM MQ za pomocą wyzwalaczy.

“Praca z funkcją MQI i klastrami” na stronie 911
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do łączenia w klastry.

“Używanie i pisanie aplikacji w systemie IBM MQ for z/OS” na stronie 916
Aplikacje IBM MQ for z/OS mogą być tworzone z programów działających w wielu różnych środowiskach. Oznacza to, że mogą korzystać z udogodnień dostępnych w więcej niż jednym środowisku.

## z/OS *Pisanie aplikacji IMS przy użyciu programu IBM MQ*

Podczas korzystania z produktu IBM MQ w aplikacjach systemu IMS należy wziąć pod uwagę, które wywołania interfejsu API produktu MQ mogą być używane, a które mechanizm jest używany na potrzeby punktu synchronizacji.

Aby uzyskać więcej informacji na temat pisania aplikacji IMS w systemie IBM MQ for z/OS, należy skorzystać z następujących odsyłaczy:

- “Punkty synchronizacji w aplikacjach IMS” na stronie 72
- “Wywołania MQI w aplikacjach IMS” na stronie 73

### Ograniczenia

Istnieją ograniczenia dotyczące wywołań funkcji API języka IBM MQ , które mogą być używane przez aplikację używającą adaptera IMS .

Następujące wywołania API IBM MQ nie są obsługiwane w aplikacji używającej adaptera IMS :

- Baza MQCB
- MQCB_FUNKCJA
- Komenda MQCTL

**Pojęcia pokrewne**

“Pisanie aplikacji mostu IMS” na stronie 76
Ten temat zawiera informacje dotyczące pisania aplikacji korzystających z mostu IBM MQ - IMS .

z/OS *Punkty synchronizacji w aplikacjach IMS*

W aplikacji IMS punkt synchronizacji jest ustanawiany za pomocą wywołań IMS , takich jak GU (get unique) do IOPCB i CHKP (checkpoint).

Aby wycofać wszystkie zmiany od poprzedniego punktu kontrolnego, można użyć wywołania IMS ROLB (wycofanie zmian). Więcej informacji na ten temat zawiera sekcja Wywołanie ROLB w dokumentacji systemu IMS .

Menedżer kolejek jest uczestnikiem protokołu zatwierdzania dwufazowego, a menedżer punktów synchronizacji systemu IMS jest koordynatorem.

Wszystkie otwarte uchwyty są zamykane przez adapter IMS w punkcie synchronizacji (z wyjątkiem środowiska BMP sterowanego wsadowo lub niesterowanego komunikatami). Jest to spowodowane tym, że inny użytkownik może zainicjować następną jednostkę pracy, a sprawdzanie zabezpieczeń produktu IBM MQ jest wykonywane podczas wykonywania wywołań MQCONN, MQCONNX i MQOPEN, a nie podczas wykonywania wywołań MQPUT lub MQGET.

Jednak w środowisku typu Wait-for-Input (WFI) lub pseudo-Wait-for-Input (PWFI) IMS nie powiadamia programu IBM MQ o zamknięciu uchwytów do momentu pojawienia się następnego komunikatu lub zwrócenia do aplikacji kodu statusu QC. Jeśli aplikacja oczekuje w regionie IMS i dowolny z tych uchwytów należy do kolejek wyzwalanych, wyzwalanie nie nastąpi, ponieważ kolejki są otwarte. Z tego powodu aplikacje działające w środowisku WFI lub PWFI powinny jawnie MQCLOSE uchwyty kolejki przed wykonaniem operacji GU na IOPCB dla następnego komunikatu.

Jeśli aplikacja IMS (BMP lub MPP) wysyła wywołanie MQDISC, otwarte kolejki są zamykane, ale nie jest pobierany niejawny punkt synchronizacji. Jeśli aplikacja zakończy się normalnie, wszystkie otwarte kolejki zostaną zamknięte i nastąpi niejawne zatwierdzenie. Jeśli aplikacja zakończy działanie nieprawidłowo, wszystkie otwarte kolejki zostaną zamknięte i nastąpi niejawne wycofanie.

z/OS *Wywołania MQI w aplikacjach IMS*

Te informacje umożliwiają zapoznanie się z użyciem wywołań MQI w aplikacjach serwera i aplikacjach Enquiry.

W tej sekcji opisano użycie wywołań MQI w następujących typach aplikacji IMS :

- “Aplikacje serwera” na stronie 929
- “Aplikacje do uzyskiwania informacji” na stronie 932

## Aplikacje serwera

Poniżej przedstawiono schemat modelu aplikacji serwera MQI:

```
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
```

Przykładowy program CSQ4ICB3 przedstawia implementację, w systemie C/370, BMP wykorzystującego ten model. Program najpierw nawiązuje komunikację z programem IMS , a następnie z programem IBM MQ:

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

Return
```

Proces inicjowania produktu IMS określa, czy program został wywołany jako BMP sterowany komunikatami, czy jako BMP zorientowany na zadanie wsadowe, oraz steruje odpowiednio połączeniami i uchwytami kolejek menedżera kolejek produktu IBM MQ :

```
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
```

Proces inicjowania programu IBM MQ łączy się z menedżerem kolejek i otwiera kolejki. W komponencie BMP sterowanym komunikatami jest on wywoływany po wykonaniu każdego punktu synchronizacji systemu IMS . W komponencie BMP zorientowanym na zadanie wsadowe jest on wywoływany tylko podczas uruchamiania programu:

```
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
```

Na implementację modelu serwera w MPP ma wpływ fakt, że MPP przetwarza pojedynczą jednostkę pracy na wywołanie. Jest to spowodowane tym, że po wykonaniu punktu synchronizacji (GU) uchwyty połączenia i kolejki są zamykane i dostarczany jest następny komunikat IMS . Ograniczenie to może zostać częściowo usunięte przez jedną z następujących sytuacji:

- **Przetwarzanie wielu komunikatów w pojedynczej jednostce pracy**

  Obejmuje to:

  – Odczytywanie komunikatu
  – Przetwarzanie wymaganych aktualizacji
  – Umieszczanie odpowiedzi

  w pętli do momentu przetworzenia wszystkich komunikatów lub do momentu przetworzenia ustawionej maksymalnej liczby komunikatów. W tym momencie pobierany jest punkt synchronizacji.

  W ten sposób można podejść do tylko niektórych typów aplikacji (na przykład prostej aktualizacji bazy danych lub zapytania). Mimo że komunikaty odpowiedzi MQI mogą być umieszczane z uprawnieniami inicjatora obsługiwanego komunikatu MQI, należy uważnie uwzględnić wpływ aktualizacji zasobów IMS na bezpieczeństwo.
- **Przetwarzanie jednego komunikatu na wywołanie MPP i zapewnienie wielu harmonogramów MPP w celu przetworzenia wszystkich dostępnych komunikatów.**

  Użyj programu monitora wyzwalacza IBM MQ IMS (CSQQTRMN), aby zaplanować transakcję MPP, gdy w kolejce IBM MQ znajdują się komunikaty i nie są obsługiwane żadne aplikacje.

  Jeśli monitor wyzwalacza uruchamia proces MPP, nazwa menedżera kolejek i nazwa kolejki są przekazywane do programu, jak pokazano w następującym wyodrębnieniu kodu COBOL:

```
* 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 CMQTMC2L.
*
* 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 '='.
```

Model serwera, który powinien być długotrwałym zadaniem, jest lepiej obsługiwany w regionie przetwarzania wsadowego, chociaż BMP nie może być wyzwalany przy użyciu CSQQTRMN.

## Aplikacje do uzyskiwania informacji

Typowa aplikacja IBM MQ inicjująca zapytanie lub aktualizację działa w następujący sposób:

- Zgromadź dane od użytkownika
- Umieść jeden lub więcej komunikatów IBM MQ
- Pobierz komunikaty odpowiedzi (może być konieczne oczekiwanie na nie)
- Podaj odpowiedź dla użytkownika

Ponieważ komunikaty umieszczane w kolejkach IBM MQ nie są dostępne dla innych aplikacji IBM MQ , dopóki nie zostaną zatwierdzone, muszą być umieszczane poza punktem synchronizacji lub aplikacja IMS musi być podzielona na dwie transakcje.

Jeśli zapytanie obejmuje umieszczenie pojedynczego komunikatu, można użyć opcji *no syncpoint* . Jeśli jednak zapytanie jest bardziej złożone lub dotyczy aktualizacji zasobów, mogą wystąpić problemy ze spójnością w przypadku wystąpienia awarii i braku użycia punktu synchronizacji.

Aby rozwiązać ten problem, można podzielić transakcje IMS MPP za pomocą wywołań MQI za pomocą przełącznika komunikatów typu program-program (program-to-program message switch). Więcej informacji na ten temat zawiera sekcja *IMS Komunikacja międzysystemowa (ISC)* . Pozwala to na zaimplementowanie programu zapytania w MPP:

```
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 *Pisanie aplikacji mostu IMS*

Ten temat zawiera informacje dotyczące pisania aplikacji korzystających z mostu IBM MQ - IMS .

Informacje na temat mostu IBM MQ - IMS zawiera sekcja Most IMS.

Aby uzyskać więcej informacji na temat pisania aplikacji mostu IMS w systemie IBM MQ for z/OS, należy skorzystać z następujących odsyłaczy:

- “Sposób postępowania mostu IMS z komunikatami” na stronie 76
- “Pisanie programów transakcyjnych IMS za pośrednictwem programu IBM MQ” na stronie 939

**Pojęcia pokrewne**

“Pisanie aplikacji IMS przy użyciu programu IBM MQ” na stronie 72
Podczas korzystania z produktu IBM MQ w aplikacjach systemu IMS należy wziąć pod uwagę, które wywołania interfejsu API produktu MQ mogą być używane, a które mechanizm jest używany na potrzeby punktu synchronizacji.

### z/OS *Sposób postępowania mostu IMS z komunikatami*

Jeśli do wysyłania komunikatów do aplikacji IMS używany jest most IBM MQ - IMS , należy utworzyć komunikaty w specjalnym formacie.

Należy również umieścić komunikaty w kolejkach systemu IBM MQ , które zostały zdefiniowane za pomocą klasy pamięci określającej grupę XCF i nazwę elementu docelowego systemu IMS . Są one nazywane kolejkami mostów MQ-IMS lub po prostu kolejkami **mostów** .

Most IBM MQ-IMS wymaga wyłącznego dostępu wejścia (MQOO_INPUT_EXCLUSIVE) do kolejki mostu, jeśli jest zdefiniowany z użyciem QSGDISP (QMGR) lub jeśli jest zdefiniowany z użyciem QSGDISP (SHARED) razem z opcją NOSHARE.

Użytkownik nie musi wpisywać się do IMS przed wysłaniem komunikatów do aplikacji IMS . Identyfikator użytkownika w polu *UserIdentifier* struktury MQMD jest używany do sprawdzania zabezpieczeń. Poziom sprawdzania jest określany, gdy program IBM MQ łączy się z systemem IMSi jest opisany w sekcji Kontrola dostępu do aplikacji dla mostu IMS. Umożliwia to zaimplementowanie pseudo-logowania.

Most IBM MQ - IMS akceptuje następujące typy komunikatów:

- Komunikaty zawierające dane transakcji IMS i strukturę MQIIH (opisane w sekcji MQIIH ):

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

  **Uwaga:**

  1. Nawiasy kwadratowe [] reprezentują opcjonalne wiele segmentów.
  2. Ustaw pole *Format* struktury MQMD na wartość MQFMT_IMS, aby użyć struktury MQIIH.

- Komunikaty zawierające dane transakcji IMS , ale bez struktury MQIIH:

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

Program IBM MQ sprawdza poprawność danych komunikatu, aby upewnić się, że suma bajtów LL i długości komunikatu MQIIH (jeśli istnieje) jest równa długości komunikatu.

Gdy most IBM MQ - IMS pobiera komunikaty z kolejek mostu, przetwarza je w następujący sposób:

- Jeśli komunikat zawiera strukturę MQIIH, most sprawdza MQIIH (patrz sekcja MQIIH ), buduje nagłówki OTMA i wysyła komunikat do produktu IMS. Kod transakcji jest określony w komunikacie wejściowym. Jeśli jest to LTERM, IMS odpowiada komunikatem DFS1288E . Jeśli kod transakcji reprezentuje komendę, program IMS wykonuje komendę; w przeciwnym razie komunikat jest umieszczany w kolejce IMS dla transakcji.
- Jeśli komunikat zawiera dane transakcji IMS , ale nie zawiera struktury MQIIH, most IMS przyjmuje następujące założenia:
  - Kod transakcji jest w bajtach od 5 do 12 danych użytkownika
  - Transakcja jest w trybie niekonwersacyjnym
  - Transakcja jest w trybie kontroli transakcji 0 (commit-then-send)
  - Zmienna *Format* w strukturze MQMD jest używana jako zmienna *MFSMapName* (na wejściu)
  - Tryb zabezpieczeń to MQISS_CHECK

  Komunikat odpowiedzi jest również budowany bez struktury MQIIH, pobierając wartość *Format* dla deskryptora MQMD z pliku *MFSMapName* wyjścia IMS .

Most IBM MQ - IMS używa jednego lub dwóch potoków dla każdej kolejki produktu IBM MQ :

- Zsynchronizowany potok Tpipe jest używany dla wszystkich komunikatów korzystających z trybu kontroli transakcji 0 (COMMIT_THEN_SEND) (wyświetlane z wartością SYN w polu statusu komendy TPIPE xxxx klienta IMS /DIS TMEMBER).
- Niezsynchronizowany potok Tpipe jest używany dla wszystkich komunikatów korzystających z trybu kontroli transakcji 1 (SEND_THEN_COMMIT)

Potoki Tpotokowe są tworzone przez produkt IBM MQ , gdy są używane po raz pierwszy. Niezsynchronizowany potok Tpipe istnieje do momentu zrestartowania systemu IMS . Zsynchronizowane

potoki istnieją do momentu zimnego uruchomienia serwera IMS . Nie można samodzielnie usunąć tych potoków.

Więcej informacji na temat obsługi komunikatów przez most IBM MQ - IMS zawierają następujące tematy:

- “Odwzorowanie komunikatów IBM MQ na typy transakcji IMS” na stronie 78
- “Jeśli nie można umieścić komunikatu w kolejce IMS” na stronie 78
- “Kody sprzężenia zwrotnego mostu IMS” na stronie 79
- “Pola MQMD w komunikatach z mostu IMS” na stronie 79
- “Pola MQIIH w komunikatach z mostu IMS” na stronie 80
- “Komunikaty odpowiedzi od IMS” na stronie 81
- “Używanie alternatywnych bloków PCB odpowiedzi w transakcjach IMS” na stronie 81
- “Wysyłanie niezamówionych komunikatów z serwisu IMS” na stronie 82
- “Segmentacja komunikatów” na stronie 82
- “Konwersja danych dla komunikatów do i z mostu IMS” na stronie 82

**Pojęcia pokrewne**

“Pisanie programów transakcyjnych IMS za pośrednictwem programu IBM MQ” na stronie 939
Kod wymagany do obsługi transakcji IMS za pośrednictwem produktu IBM MQ zależy od formatu komunikatu wymaganego przez transakcję IMS oraz zakresu zwracanych przez nią odpowiedzi. Istnieje jednak kilka punktów, które należy wziąć pod uwagę, gdy aplikacja obsługuje informacje o formatowaniu ekranu w systemie IMS .

z/OS *Odwzorowanie komunikatów IBM MQ na typy transakcji IMS*

Tabela opisująca odwzorowanie komunikatów IBM MQ na typy transakcji IMS .

*Tabela 132. W jaki sposób komunikaty IBM MQ są odwzorowywane na typy transakcji IMS*

| **IBM MQ typ komunikatu** | **Commit-then-send (tryb 0)-używa zsynchronizowanych potoków IMS** | **Send-then-commit (tryb 1)-używa niezsynchronizowanych potoków IMS** |
|---|---|---|
| Trwałe komunikaty IBM MQ | • Odtwarzalne transakcje o pełnej funkcjonalności<br>• Transakcje nienaprawialne są odrzucane przez IMS | • Transakcje krótkiej ścieżki<br>• Transakcje konwersacyjne<br>• W pełni funkcjonalne transakcje |
| Nietrwałe komunikaty IBM MQ | • Nienaprawialne transakcje z pełnymi funkcjami<br>• Transakcje odtwarzalne są dozwolone w poprawkach IMS V8 i APAR PQ61404 oraz we wszystkich późniejszych wersjach systemu IMS . | • Transakcje krótkiej ścieżki<br>• Transakcje konwersacyjne<br>• W pełni funkcjonalne transakcje |

**Uwaga:** Komendy IMS nie mogą używać trwałych komunikatów IBM MQ w trybie kontroli transakcji 0. Więcej informacji na ten temat zawiera sekcja Tryb zatwierdzania (commitMode) .

z/OS *Jeśli nie można umieścić komunikatu w kolejce IMS*

Sekcja zawiera informacje na temat działań, które należy wykonać, jeśli nie można umieścić komunikatu w kolejce IMS .

Jeśli komunikat nie może zostać umieszczony w kolejce IMS , program IBM MQpodejmuje następujące działanie:

- Jeśli komunikat nie może zostać umieszczony w katalogu IMS z powodu niepoprawnego komunikatu, jest on umieszczany w kolejce niedostarczonych komunikatów, a komunikat jest wysyłany do konsoli systemowej.
- Jeśli komunikat jest poprawny, ale został odrzucony przez program IMS, program IBM MQ wysyła komunikat o błędzie do konsoli systemowej, komunikat zawiera kod rozpoznania IMS , a komunikat IBM MQ jest umieszczany w kolejce niedostarczonych komunikatów. Jeśli kod rozpoznania IMS to 001A, program IMS wysyła do kolejki odpowiedzi komunikat IBM MQ zawierający przyczynę niepowodzenia.

  **Uwaga:** W wymienionych wcześniej okolicznościach, jeśli program IBM MQ nie może umieścić komunikatu w kolejce niedostarczonych komunikatów z jakiegokolwiek powodu, komunikat jest zwracany do źródłowej kolejki IBM MQ . Do konsoli systemowej wysyłany jest komunikat o błędzie, a z tej kolejki nie są wysyłane żadne dalsze komunikaty.

  Aby ponownie wysłać komunikaty, wykonaj ***jedną*** z następujących czynności:

  – Zatrzymaj i zrestartuj potoki Tpotoków w programie IMS odpowiadające kolejce.
  – Zmień kolejkę na GET (DISABLED) i ponownie na GET (ENABLED)
  – Zatrzymaj i zrestartuj produkt IMS lub komponent OTMA.
  – Zatrzymaj i zrestartuj podsystem IBM MQ .
- Jeśli komunikat został odrzucony przez program IMS z powodu błędu innego niż komunikat, komunikat IBM MQ jest zwracany do kolejki źródłowej, program IBM MQ zatrzymuje przetwarzanie kolejki, a do konsoli systemowej wysyłany jest komunikat o błędzie.

  Jeśli wymagany jest komunikat raportu o wyjątku, most umieszcza go w kolejce odpowiedzi z uprawnieniami nadawcy. Jeśli nie można umieścić komunikatu w kolejce, komunikat raportu jest umieszczany w kolejce niedostarczonych komunikatów z uprawnieniami mostu. Jeśli nie można go umieścić w DLQ, jest on odrzucany.

z/OS *Kody sprzężenia zwrotnego mostu IMS*

Kody rozpoznania IMS są zwykle wyprowadzane w formacie szesnastkowym w komunikatach konsoli IBM MQ , takich jak CSQ2001I (na przykład kod rozpoznania 0x001F). Kody informacji zwrotnych IBM MQ widoczne w nagłówku niedostarczonych komunikatów w kolejce niedostarczonych komunikatów są liczbami dziesiętnymi.

Kody sprzężenia zwrotnego mostu IMS należą do zakresu od 301 do 399 lub od 600 do 855 dla kodu rozpoznania NACK 0x001A. Są one odwzorowywane z kodów rozpoznania IMS-OTMA w następujący sposób:

1. Kod rozpoznania IMS-OTMA jest przekształcany z liczby szesnastkowej na liczbę dziesiętną.
2. 300 jest dodawana do liczby uzyskanej w wyniku obliczenia w 1, co daje kod IBM MQ *Feedback* .
3. Specjalny przypadek to IMS-OTMA sense code 0x001A, dziesiętne 26. Generowany jest kod *Feedback* w zakresie od 600 do 855.
   a. Kod przyczyny IMS-OTMA jest przekształcany z liczby szesnastkowej na liczbę dziesiętną.
   b. Wartość 600 jest dodawana do liczby uzyskanej w wyniku obliczenia w a, co daje kod IBM MQ *Feedback* (Opinia).

Informacje na temat kodów rozpoznania IMS-OTMA zawiera sekcja Kody rozpoznania OTMA dla komunikatów NAK.

z/OS *Pola MQMD w komunikatach z mostu IMS*

Informacje o polach MQMD w komunikatach z mostu IMS .

Deskryptor MQMD komunikatu źródłowego jest przenoszony przez IMS w sekcji User Data (Dane użytkownika) nagłówków OTMA. Jeśli komunikat pochodzi z pliku IMS, jest on budowany przez wyjście rozstrzygania miejsca docelowego produktu IMS . Struktura MQMD komunikatu odebranego z IMS jest zbudowana w następujący sposób:

**StrucID**
"MD"

**Wersja**
MQMD_VERSION_1

**Raport**
MQRO_BRAK

**MsgType**
MQMT_REPLY

**Utrata ważności**
Jeśli opcja MQIIH_PASS_EXPIRATION jest ustawiona w polu Flags nagłówka MQIIH, to pole zawiera pozostały czas utraty ważności, w przeciwnym razie jest ustawione na wartość MQEI_UNLIMITED.

**Opinie**
MQFB_BRAK

**Kodowanie**
MQENC.Native (kodowanie systemu z/OS )

**CodedCharSetId**
MQCCSI_Q_MGR ( CodedCharSetID systemu z/OS )

**Format**
MQFMT_IMS, jeśli MQMD.Format komunikatu wejściowego to MQFMT_IMS, w przeciwnym razie IOPCB.MODNAME

**Priorytet**
MQMD.Priority komunikatu wejściowego

**Trwałość**
Zależy od trybu zatwierdzania: MQMD.Persistence komunikatu wejściowego, jeśli trwałość CM-1; jest zgodna z odtwarzalnością komunikatu IMS , jeśli CM-0

**MsgId**
MQMD.MsgId jeśli MQRO_PASS_MSG_ID, w przeciwnym razie New MsgId (wartość domyślna)

**CorrelId**
MQMD.CorrelId z komunikatu wejściowego, jeśli MQRO_PASS_CORREL_ID, w przeciwnym razie MQMD.MsgId z komunikatu wejściowego (wartość domyślna)

**BackoutCount**
0

**ReplyToQ**
Puste

**ReplyToQMgr**
Odstępy (ustawione na nazwę lokalnego menedżera kolejek przez menedżer kolejek podczas operacji MQPUT)

**UserIdentifier**
MQMD.UserIdentifier komunikatu wejściowego

**AccountingToken**
MQMD.AccountingToken komunikatu wejściowego

**Dane_tożsamości_aplikacji**
MQMD.ApplIdentityData komunikatu wejściowego

**Typ_aplikacji_wstawiającej**
MQAT_XCF, jeśli nie ma błędu, w przeciwnym razie MQAT_BRIDGE

**Nazwa_aplikacji_wstawiającej**
<XCFgroupName> <XCFmemberName>, jeśli nie wystąpił błąd, w przeciwnym razie nazwa QMGR

**PutDate**
Data umieszczenia komunikatu

**PutTime**
Czas umieszczenia komunikatu

**Dane_pochodzenia_aplikacji**
Puste

z/OS *Pola MQIIH w komunikatach z mostu IMS*

Informacje o polach MQIIH w komunikatach z mostu IMS .

Komunikat MQIIH odebrany z produktu IMS jest budowany w następujący sposób:

**StrucId**
"IIH"

**Wersja**
1

**StrucLength**
84

**Kodowanie**
RODZIMA MQENC

**CodedCharSetId**
MQCCSI_Q_MGR (menedżer kolejek MQ)

**Format**
MQIIH.ReplyToFormat komunikatu wejściowego, jeśli MQIIH.ReplyToFormat nie jest pusty, w przeciwnym razie IOPCB.MODNAME

**Flagi**
0

**LTermOverride**
Nazwa LTERM (Tpipe) z nagłówka OTMA

**MFSMapName**
Nazwa mapy z nagłówka OTMA

**Format ReplyTo**
Puste

**Authenticator**
MQIIH.Authenticator komunikatu wejściowego, jeśli komunikat odpowiedzi jest umieszczany w kolejce mostu MQ-IMS , w przeciwnym razie jest pusty.

**Identyfikator TranInstance**
Identyfikator konwersacji/znacznik serwera z nagłówka OTMA, jeśli w konwersacji. W wersjach systemu IMS wcześniejszych niż V14to pole ma zawsze wartość null, jeśli nie jest używane w konwersacji. Począwszy od wersji IMS V14 , to pole może być ustawiane przez system IMS , nawet jeśli nie jest używane w konwersacji.

**TranState**
"C", jeśli w konwersacji, w przeciwnym razie puste

**CommitMode**
Tryb kontroli transakcji z nagłówka OTMA ("0" lub "1")

**SecurityScope**
Wartość pusta

**Zarezerwowane**
Wartość pusta

z/OS *Komunikaty odpowiedzi od IMS*

Gdy transakcja IMS kieruje dane ISRT do swojego IOPCB, komunikat jest kierowany z powrotem do źródłowego LTERM lub TPIPE.

Są one wyświetlane w pliku IBM MQ jako komunikaty odpowiedzi. Komunikaty odpowiedzi z produktu IMS są umieszczane w kolejce odpowiedzi określonej w pierwotnym komunikacie. Jeśli komunikatu nie można umieścić w kolejce odpowiedzi, jest on umieszczany w kolejce niedostarczonych komunikatów przy użyciu uprawnień mostu. Jeśli komunikat nie może zostać umieszczony w kolejce niedostarczonych komunikatów, do produktu IMS wysyłane jest potwierdzenie negatywne informujące, że komunikat nie może zostać odebrany. Odpowiedzialność za komunikat jest następnie zwracana do IMS. Jeśli używany jest tryb kontroli transakcji 0, komunikaty z tego potoku Tpipe nie są wysyłane do mostu i pozostają w kolejce IMS . Oznacza to, że do czasu restartu nie są wysyłane żadne dalsze komunikaty. Jeśli używany jest tryb kontroli transakcji 1, inne prace mogą być kontynuowane.

Jeśli odpowiedź ma strukturę MQIIH, jej typ formatu to MQFMT_IMS; jeśli nie, typ formatu jest określany przez nazwę IMS MOD używaną podczas wstawiania komunikatu.

z/OS *Używanie alternatywnych bloków PCB odpowiedzi w transakcjach IMS*

Gdy transakcja systemu IMS używa alternatywnych bloków PCB odpowiedzi (ISRTs do ALTPCB lub wysyła wywołanie CHNG do modyfikowalnego bloku PCB), wywoływane jest wyjście przed routingiem (DFSYPRX0) w celu określenia, czy komunikat powinien zostać przekierowany.

Jeśli komunikat ma zostać przekierowany, wywoływane jest wyjście rozstrzygania miejsca docelowego (DFSYDRU0) w celu potwierdzenia miejsca docelowego i przygotowania informacji nagłówkowych. Informacje na temat tych programów obsługi wyjścia zawiera sekcja Używanie wyjść OTMA w IMS oraz sekcja Wyjście routingu wstępnego DFSYPRX0 .

Jeśli nie zostanie wykonane działanie w wyjściach, wszystkie dane wyjściowe z transakcji IMS zainicjowanych z menedżera kolejek IBM MQ , zarówno do IOPCB, jak i do ALTPCB, zostaną zwrócone do tego samego menedżera kolejek.

z/OS *Wysyłanie niezamówionych komunikatów z serwisu IMS*

Aby wysłać komunikaty z produktu IMS do kolejki IBM MQ , należy wywołać transakcję IMS , która ISRTs do ALTPCB.

Konieczne jest napisanie programów zewnętrznych routingu wstępnego i rozstrzygania miejsc docelowych w celu kierowania niezamówionych komunikatów z produktu IMS i budowania danych użytkownika OTMA, aby możliwe było poprawne zbudowanie deskryptora MQMD komunikatu. Informacje na temat tych programów obsługi wyjścia znajdują się w sekcji Program obsługi wyjścia routingu wstępnego DFSYPRX0 i Program obsługi wyjścia o docelowej rozdzielczości .

**Uwaga:** Most IBM MQ - IMS nie wie, czy odbierany komunikat jest odpowiedzią, czy niezamówionym komunikatem. Obsługuje on komunikat w taki sam sposób w każdym przypadku, budując deskryptorze MQMD i MQIIH odpowiedzi na podstawie UserData z OTMA, który został odebrany wraz z komunikatem.

Niezamówione komunikaty mogą tworzyć nowe potoki Tpotoków. Na przykład, jeśli istniejąca transakcja IMS została przełączona na nowy terminal LTERM (na przykład PRINT01), ale implementacja wymaga, aby dane wyjściowe były dostarczane za pośrednictwem OTMA, zostanie utworzony nowy potok Tpipe (w tym przykładzie nazywany PRINT01 ). Domyślnie jest to niezsynchronizowany Tpipe. Jeśli implementacja wymaga, aby komunikat był odtwarzalny, należy ustawić flagę wyjścia wyjścia rozstrzygnięcia miejsca docelowego. Więcej informacji na ten temat zawiera publikacja *IMS Customization Guide* .

z/OS *Segmentacja komunikatów*

Transakcje IMS można zdefiniować jako oczekiwane dane wejściowe jedno-lub wielosegmentowe.

Źródłowa aplikacja IBM MQ musi tworzyć dane wejściowe użytkownika następujące po strukturze MQIIH jako jeden lub więcej segmentów danych LLZZ. Wszystkie segmenty komunikatu IMS muszą być zawarte w pojedynczym komunikacie IBM MQ wysłanym za pomocą pojedynczego wywołania MQPUT.

Maksymalna długość segmentu danych LLZZ jest definiowana przez IMS/OTMA (32767 bajtów). Łączna długość komunikatu IBM MQ jest sumą bajtów LL plus długość struktury MQIIH.

Wszystkie segmenty odpowiedzi są zawarte w pojedynczym komunikacie IBM MQ .

Istnieje dodatkowe ograniczenie 32 kB dotyczące komunikatów w formacie MQFMT_IMS_VAR_STRING. Gdy dane w komunikacie ASCII-mixed CCSID są konwertowane na komunikat EBCDIC-mixed CCSID, bajt

shift-in lub bajt shift-out jest dodawany za każdym razem, gdy występuje przejście między znakami SBCS i DBCS. Ograniczenie 32 kB dotyczy maksymalnej wielkości komunikatu. Oznacza to, że ponieważ pole LL w komunikacie nie może być większe niż 32 kB, komunikat nie może być dłuższy niż 32 kB, w tym wszystkie znaki shift-in i shift-out. Aplikacja budująca komunikat musi na to zezwolić.

z/OS *Konwersja danych dla komunikatów do i z mostu IMS*

Konwersja danych jest wykonywana przez rozproszoną funkcję kolejkowania (która może wywoływać wszystkie niezbędne wyjścia) lub przez wewnątrzgrupowy agent kolejkowania (który nie obsługuje użycia wyjść), gdy umieszcza komunikat w kolejce docelowej, która ma informacje XCF zdefiniowane dla swojej klasy pamięci masowej. Konwersja danych nie jest wykonywana, gdy komunikat jest dostarczany do kolejki przez publikowanie/subskrypcję.

Wszystkie wymagane wyjścia muszą być dostępne dla rozproszonego narzędzia kolejkowania w zestawie danych, do którego odwołuje się instrukcja CSQXLIB DD. Oznacza to, że można wysyłać komunikaty do aplikacji IMS przy użyciu mostu IBM MQ - IMS z dowolnej platformy IBM MQ .

W przypadku wystąpienia błędów konwersji komunikat jest umieszczany w kolejce bez konwersji. W rezultacie most IBM MQ - IMS traktuje go jako błąd, ponieważ most nie rozpoznaje formatu nagłówka. Jeśli wystąpi błąd konwersji, do konsoli z/OS zostanie wysłany komunikat o błędzie.

Szczegółowe informacje na temat konwersji danych znajdują się w sekcji “Zapisywanie wyjść konwersji danych” na stronie 1013 .

## Wysyłanie komunikatów do mostu IBM MQ - IMS

Aby zapewnić poprawne wykonanie konwersji, należy poinformować menedżer kolejek o formacie komunikatu.

Jeśli komunikat ma strukturę MQIIH, parametr *Format* w strukturze MQMD musi być ustawiony na wbudowany format MQFMT_IMS, a parametr *Format* w strukturze MQIIH musi być ustawiony na nazwę formatu opisującego dane komunikatu. Jeśli nie ma MQIIH, ustaw wartość *Format* w deskryptorze MQMD na nazwę formatu.

Jeśli wszystkie dane (inne niż LLZZs) są danymi znakowymi (MQCHAR), należy użyć jako nazwy formatu (odpowiednio w MQIIH lub MQMD) wbudowanego formatu MQFMT_IMS_VAR_STRING. W przeciwnym razie należy użyć własnej nazwy formatu. W takim przypadku należy również podać wyjście konwersji danych dla formatu. Wyjście musi obsługiwać konwersję LLZZs w komunikacie, oprócz samych danych (ale nie musi obsługiwać żadnych MQIIH na początku komunikatu).

Jeśli aplikacja używa produktu *MFSMapName*, można użyć komunikatów z systemem MQFMT_IMS i zdefiniować nazwę odwzorowania przekazaną do transakcji IMS w polu MFSMapName MQIIH.

## Odbieranie komunikatów z mostu IBM MQ - IMS

Jeśli w oryginalnym komunikacie wysyłanym do produktu IMSznajduje się struktura MQIIH, taka struktura jest również obecna w komunikacie odpowiedzi.

Aby upewnić się, że odpowiedź została poprawnie przekształcona:

- Jeśli w oryginalnym komunikacie istnieje struktura MQIIH, określ format komunikatu odpowiedzi w polu MQIIH *ReplytoFormat* komunikatu oryginalnego. Ta wartość jest umieszczana w polu MQIIH *Format* komunikatu odpowiedzi. Jest to szczególnie przydatne, gdy wszystkie dane wyjściowe mają postać LLZZ < dane znakowe >.
- Jeśli w oryginalnym komunikacie nie ma struktury MQIIH, należy określić format komunikatu odpowiedzi jako nazwę MFS MOD w narzędziu ISRT aplikacji IMS do IOPCB.

z/OS *Pisanie programów transakcyjnych IMS za pośrednictwem programu IBM MQ*

Kod wymagany do obsługi transakcji IMS za pośrednictwem produktu IBM MQ zależy od formatu komunikatu wymaganego przez transakcję IMS oraz zakresu zwracanych przez nią odpowiedzi. Istnieje jednak kilka punktów, które należy wziąć pod uwagę, gdy aplikacja obsługuje informacje o formatowaniu ekranu w systemie IMS .

Gdy transakcja IMS jest uruchamiana z ekranu 3270, komunikat jest przekazywany przez usługi IMS Message Format Services. Może to spowodować usunięcie wszystkich zależności terminalu ze strumienia danych widzianego przez transakcję. Gdy transakcja jest uruchamiana za pośrednictwem OTMA, MFS nie jest zaangażowany. Jeśli logika aplikacji jest zaimplementowana w systemie plików MFS, należy ją ponownie utworzyć w nowej aplikacji.

W niektórych transakcjach IMS aplikacja użytkownika końcowego może zmodyfikować pewne zachowanie ekranu 3270, na przykład podświetlając pole, w którym wprowadzono niepoprawne dane. Ten typ informacji jest przekazywany przez dodanie dwubajtowego pola atrybutu do komunikatu IMS dla każdego pola ekranu, które ma być modyfikowane przez program.

Dlatego w przypadku kodowania aplikacji w celu naśladowania terminalu 3270 należy uwzględnić te pola podczas budowania lub odbierania komunikatów.

Może być konieczne zakodowanie informacji w programie w celu przetworzenia:

- Który klawisz jest naciśnięty (na przykład Enter i PF1)
- Miejsce, w którym znajduje się kursor, gdy komunikat jest przekazywany do aplikacji
- Określa, czy pola atrybutów zostały ustawione przez aplikację IMS .
  - Wysoka, normalna lub zerowa intensywność
  - Kolor
  - Określa, czy program IMS ma oczekiwać tego pola po następnym naciśnięciu klawisza Enter.
- Określa, czy aplikacja IMS używała znaków o kodzie zero (X'3F') w dowolnych polach.

Jeśli komunikat IMS zawiera tylko dane znakowe (oprócz segmentu danych LLZZ), a używana jest struktura MQIIH, należy ustawić format MQMD na MQFMT_IMS, a format MQIIH na MQFMT_IMS_VAR_STRING.

Jeśli komunikat IMS zawiera tylko dane znakowe (oprócz segmentu danych LLZZ), a użytkownik **nie** używa struktury MQIIH, należy ustawić format MQMD na wartość MQFMT_IMS_VAR_STRING i upewnić się, że aplikacja IMS określa nazwę MODname MQFMT_IMS_VAR_STRING podczas odpowiadania. Jeśli wystąpi problem (na przykład użytkownik nie ma uprawnień do użycia transakcji) i program IMS wyśle komunikat o błędzie, będzie miał nazwę MODname w postaci DFSMOx, gdzie x jest liczbą z zakresu od 1 do 5. Jest on umieszczany w strukturze MQMD.Format.

Jeśli komunikat IMS zawiera dane binarne, spakowane lub zmiennopozycyjne (z wyjątkiem segmentu danych LLZZ), należy zakodować własne procedury konwersji danych. Informacje na temat formatowania ekranu w programie IMS zawiera publikacja *IMS/ESA Application Programming: Transaction Manager* .

Podczas pisania kodu do obsługi transakcji IMS za pośrednictwem produktu IBM MQnależy wziąć pod uwagę następujące tematy.

- “Pisanie aplikacji IBM MQ do wywoływania transakcji konwersacyjnych systemu IMS” na stronie 940
- “Pisanie programów zawierających komendy systemu IMS” na stronie 941
- “Wyzwalanie” na stronie 941

## Pisanie aplikacji IBM MQ do wywoływania transakcji konwersacyjnych systemu IMS

Te informacje są pomocne podczas pisania IBM MQ aplikacji wywołującej IMS transakcje konwersacyjne.

Pisząc aplikację, która wywołuje konwersację IMS , należy wziąć pod uwagę następujące kwestie:

- Dołącz strukturę MQIIH do komunikatu aplikacji.
- Ustaw parametr *CommitMode* w MQIIH na wartość MQICM_SEND_THEN_COMMIT.
- Aby wywołać nową konwersację, należy ustawić parametr *TranState* w MQIIH na wartość MQITS_NOT_IN_CONVERSATION.
- Aby wywołać drugi i kolejne kroki konwersacji, należy ustawić parametr *TranState* na wartość MQITS_IN_CONVERSATION i ustawić parametr *TranInstanceId* na wartość tego pola zwróconą w poprzednim kroku konwersacji.

- W programie IMS nie ma łatwego sposobu na znalezienie wartości parametru *TranInstanceId*, jeśli oryginalny komunikat wysłany z programu IMSzostanie utracony.
- Aplikacja musi sprawdzić *TranState* komunikatów z IMS , aby sprawdzić, czy transakcja IMS zakończyła konwersację.
- Możesz użyć /EXIT, aby zakończyć konwersację. Należy również ująć w cudzysłów parametr *TranInstanceId*, ustawić parametr *TranState* na wartość MQITS_IN_CONVERSATION i użyć kolejki IBM MQ , w której prowadzona jest konwersacja.
- Nie można użyć opcji /HOLD ani /REL do wstrzymania lub zwolnienia konwersacji.
- Konwersacje wywoływane przez most IBM MQ - IMS są przerywane po zrestartowaniu serwera IMS .

### Pisanie programów zawierających komendy systemu IMS

Aplikacja może utworzyć IBM MQ komunikat w postaci LLZZ*Komenda*, zamiast transakcji, gdzie *Komenda* jest w postaci /DIS TRAN PART lub /DIS POOL ALL.

Większość komend IMS może być wykonywanych w ten sposób. Szczegółowe informacje na ten temat zawiera sekcja *IMS V11 Komunikacja i połączenia* . Dane wyjściowe komendy są odbierane w komunikacie odpowiedzi IBM MQ w postaci tekstowej, tak jak byłyby wysyłane do terminalu 3270 w celu wyświetlenia.

OTMA zaimplementowało specjalną formę komendy wyświetlania transakcji IMS , która zwraca strukturalną postać danych wyjściowych. Dokładny format jest zdefiniowany w sekcji *Komunikacja i połączenia programu IMS V11*. Aby wywołać ten formularz z poziomu komunikatu IBM MQ , należy zbudować dane komunikatu tak jak wcześniej, na przykład /DIS TRAN PART, i ustawić pole TranState w MQIIH na wartość MQITS_ARCHITECTED. IMS przetwarza komendę i zwraca odpowiedź w postaci strukturalnej. Odpowiedź architected zawiera wszystkie informacje, które można znaleźć w postaci tekstowej danych wyjściowych, oraz jeden dodatkowy fragment informacji: informacja, czy transakcja jest zdefiniowana jako odtwarzalna, czy nie.

### Wyzwalanie

Most IBM MQ - IMS nie obsługuje komunikatów wyzwalacza.

W przypadku zdefiniowania kolejki inicjującej, która używa klasy pamięci z parametrami XCF, komunikaty umieszczane w tej kolejce są odrzucane po dotarciu do mostu.

## Pisanie aplikacji proceduralnych klienta

Informacje potrzebne do pisania aplikacji klienckich w systemie IBM MQ przy użyciu języka proceduralnego.

Aplikacje można budować i uruchamiać w środowisku klienta IBM MQ . Aplikacja musi zostać zbudowana i powiązana z używanym IBM MQ MQI client . Sposób, w jaki aplikacje są budowane i łączone, zależy od używanej platformy i języka programowania. Informacje na temat budowania aplikacji klienckich zawiera sekcja “Budowanie aplikacji dla produktu IBM MQ MQI clients” na stronie 947.

Aplikację IBM MQ można uruchomić zarówno w pełnym środowisku IBM MQ , jak i w środowisku IBM MQ MQI client bez zmiany kodu, o ile spełnione są pewne warunki. Więcej informacji na temat uruchamiania aplikacji w środowisku klienta IBM MQ zawiera sekcja “Uruchamianie aplikacji w środowisku IBM MQ MQI client” na stronie 949.

Jeśli do zapisu aplikacji uruchamianych w środowisku IBM MQ MQI client używany jest interfejs kolejki komunikatów (MQI), to podczas wywołania MQI należy nałożyć pewne dodatkowe elementy sterujące, aby zapewnić, że przetwarzanie aplikacji IBM MQ nie jest zakłócone. Więcej informacji na temat tych elementów sterujących zawiera sekcja “Używanie interfejsu MQI w aplikacji klienckiej” na stronie 942.

Poniższe tematy zawierają informacje dotyczące przygotowywania i uruchamiania innych typów aplikacji jako aplikacji klienckich:

- “Przygotowywanie i uruchamianie aplikacji CICS i Tuxedo” na stronie 962
- “Przygotowywanie i uruchamianie aplikacji produktu Microsoft Transaction Server” na stronie 50

- “Przygotowywanie i uruchamianie aplikacji produktu IBM MQ JMS” na stronie 965

**Pojęcia pokrewne**

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć języka proceduralnego lub obiektowego. Przed rozpoczęciem projektowania i pisania aplikacji IBM MQ należy zapoznać się z podstawowymi pojęciami związanymi z produktem IBM MQ .

“Tworzenie aplikacji dla składnika IBM MQ” na stronie 5
Użytkownik może tworzyć aplikacje do wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i zasobami pokrewnymi. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i środowiskach.

“Uwagi dotyczące projektowania dla aplikacji IBM MQ” na stronie 51
Po podjęciu decyzji, w jaki sposób aplikacje mogą korzystać z dostępnych platform i środowisk, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

“Pisanie aplikacji proceduralnej dotyczącej kolejkowania” na stronie 744
Ten temat zawiera informacje o pisaniu aplikacji kolejkowania, nawiązywaniu i rozłączaniu połączeń z menedżerem kolejek, publikowaniu i subskrybowaniu oraz otwieraniu i zamykaniu obiektów.

“Pisanie aplikacji publikowania/subskrybowania” na stronie 834
Rozpocznij pisanie aplikacji IBM MQ publikowania/subskrypcji.

“Budowanie aplikacji proceduralnej” na stronie 1029
Użytkownik może napisać aplikację IBM MQ w jednym z kilku języków proceduralnych i uruchomić ją na kilku różnych platformach.

“Obsługa błędów proceduralnych programu” na stronie 1069
Te informacje wyjaśniają błędy powiązane z wywołaniami MQI aplikacji podczas wykonywania wywołania lub po dostarczeniu komunikatu do miejsca docelowego.

**Zadania pokrewne**

“Korzystanie z przykładowych programów proceduralnych IBM MQ” na stronie 1089
Te przykładowe programy zostały napisane w językach proceduralnych i przedstawiają typowe zastosowania interfejsu kolejek komunikatów (Message Queue Interface-MQI). Programy IBM MQ na różnych platformach.

## Używanie interfejsu MQI w aplikacji klienckiej

W tej kolekcji tematów omówiono różnice między pisaniem aplikacji IBM MQ do uruchomienia w środowisku klienta interfejsu kolejki komunikatów (MQI) a uruchomieniem w pełnym środowisku menedżera kolejek produktu IBM MQ .

Podczas projektowania aplikacji należy wziąć pod uwagę elementy sterujące, które należy narzucić podczas wywołania MQI, aby zapewnić, że przetwarzanie aplikacji IBM MQ nie jest zakłócone.

Przed uruchomieniem aplikacji używających interfejsu MQI należy utworzyć pewne obiekty IBM MQ . Więcej informacji na ten temat zawiera sekcja Aplikacje używające interfejsu MQI.

### *Ograniczanie wielkości komunikatu w aplikacji klienckiej*

Menedżer kolejek ma maksymalną długość komunikatu, ale maksymalna wielkość komunikatu, który można przesłać z aplikacji klienckiej, jest ograniczona przez definicję kanału.

Atrybut maksymalnej długości komunikatu (MaxMsgLength) menedżera kolejek określa maksymalną długość komunikatu, który może obsłużyć dany menedżer kolejek.

Multi W systemie Wiele platformmożna zwiększyć atrybut maksymalnej długości komunikatu menedżera kolejek. Więcej informacji na ten temat zawiera sekcja ALTER QMGR.

Wartość parametru MaxMsgLength dla menedżera kolejek można określić za pomocą wywołania MQINQ.

Jeśli atrybut MaxMsgLength zostanie zmieniony, nie zostanie wykonane żadne sprawdzenie, czy nie ma już kolejek, a nawet komunikatów o długości większej niż nowa wartość. Po zmianie tego atrybutu należy zrestartować aplikacje i kanały, aby zmiana odniosła skutek. W takim przypadku nie jest

możliwe generowanie nowych komunikatów, których długość przekracza wartość parametru MaxMsgdla menedżera kolejek lub kolejki (o ile segmentacja menedżera kolejek nie jest dozwolona).

Maksymalna długość komunikatu w definicji kanału ogranicza wielkość komunikatu, który może być przesyłany przez połączenie klienta. Jeśli aplikacja IBM MQ próbuje użyć wywołania MQPUT lub MQGET z komunikatem większym niż ten, do aplikacji zwracany jest kod błędu. Parametr maksymalnej wielkości komunikatu w definicji kanału nie ma wpływu na maksymalną wielkość komunikatu, która może zostać wykorzystana przy użyciu obiektu MQCB w połączeniu klienckim.

**Pojęcia pokrewne**

"Korzystanie z usługi MQCONNX" na stronie 947
Za pomocą wywołania MQCONNX można określić strukturę definicji kanału (MQCD) w strukturze MQCNO.

**Odsyłacze pokrewne**

Maksymalna długość komunikatu (MAXMSGL)
ZMIEŃ KANAŁ
2010 (07DA) (RC2010): MQRC_DATA_LENGTH_ERROR

## *Wybór identyfikatora CCSID klienta lub serwera*

Dla klienta należy użyć lokalnego identyfikatora kodowanego zestawu znaków (CCSID). Menedżer kolejek wykonuje niezbędną konwersję. Zmiennej środowiskowej **MQCCSID** można użyć do przesłonięcia identyfikatora CCSID. Jeśli aplikacja wykonuje wiele operacji PUTs, pola CCSID i kodowania deskryptora MQMD mogą zostać nadpisane po wykonaniu pierwszej operacji PUT.

Dane przekazywane przez interfejs kolejki komunikatów (MQI) z aplikacji do kodu pośredniczącego klienta muszą mieć lokalny identyfikator CCSID zakodowany dla parametru IBM MQ MQI client. Jeśli połączony menedżer kolejek wymaga konwersji danych, konwersja jest wykonywana przez kod obsługi klienta w menedżerze kolejek.

W produkcie IBM WebSphere MQ 7.0 i jego nowszych wersjach klient Java może przeprowadzić konwersję, jeśli nie jest to możliwe dla menedżera kolejek. Patrz "IBM MQ classes for Java połączenia klientów" na stronie 384.

Kod klienta zakłada, że dane znakowe przesyłane przez MQI w kliencie mają identyfikator CCSID skonfigurowany dla tej stacji roboczej. Jeśli ten identyfikator CCSID nie jest obsługiwany lub nie jest wymagany, można go przesłonić zmienną środowiskową **MQCCSID** za pomocą jednej z następujących komend:

- Windows

```
SET MQCCSID=850
```

- Linux AIX

```
export MQCCSID=850
```

- IBM i

```
ADDENVVAR ENVVAR(MQCCSID) VALUE(37)
```

Jeśli ten parametr jest ustawiony w profilu, przyjmuje się, że wszystkie dane MQI znajdują się w stronie kodowej 850.

**Uwaga:** Założenie dotyczące strony kodowej 850 nie ma zastosowania do danych aplikacji w komunikacie.

Jeśli aplikacja wykonuje wiele operacji PUTs, które zawierają nagłówki IBM MQ po deskryptorze komunikatu (MQMD), należy pamiętać, że pola CCSID i kodowania deskryptora MQMD są nadpisywane po zakończeniu pierwszego operacji PUT.

Po wykonaniu pierwszej operacji PUT te pola zawierają wartość używaną przez połączony menedżer kolejek do przekształcania nagłówków IBM MQ . Upewnij się, że aplikacja zresetuje wartości do wymaganych wartości.

## *Używanie MQINQ w aplikacji klienckiej*

Niektóre wartości odpytywane za pomocą MQINQ są modyfikowane przez kod klienta.

**CCSID**
jest ustawiony na identyfikator CCSID klienta, a nie menedżera kolejek.

***MaxMsgdługość komunikatu***
jest zmniejszona, jeśli jest ograniczona przez definicję kanału. Będzie to niższa z następujących wartości:

- Wartość zdefiniowana w definicji kolejki lub
- Wartość zdefiniowana w definicji kanału

Więcej informacji na ten temat zawiera podręcznik MQINQ.

## *Korzystanie z koordynacji punktów synchronizacji w aplikacji klienckiej*

Aplikacja działająca na kliencie podstawowym może wydać komendy MQCMIT i MQBACK, ale zasięg sterowania punktem synchronizacji jest ograniczony do zasobów MQI. Zewnętrznego menedżera transakcji można używać z rozszerzonym klientem transakcyjnym.

W programie IBM MQjedną z ról menedżera kolejek jest sterowanie punktem synchronizacji w aplikacji. Jeśli aplikacja działa na kliencie podstawowym systemu IBM MQ , może wydać komendy MQCMIT i MQBACK, ale zasięg sterowania punktem synchronizacji jest ograniczony do zasobów MQI. Komenda IBM MQ MQBEGIN nie jest poprawna w środowisku klienta podstawowego.

Aplikacje działające w pełnym środowisku menedżera kolejek na serwerze mogą koordynować wiele zasobów (na przykład bazy danych) za pośrednictwem monitora transakcji. Na serwerze można użyć monitora transakcji dostarczanego z produktami IBM MQ lub innego monitora transakcji, takiego jak CICS. Nie można używać monitora transakcji z podstawową aplikacją kliencką.

Zewnętrznego menedżera transakcji można używać z rozszerzonym klientem transakcyjnym IBM MQ . Więcej informacji na ten temat zawiera sekcja Czym jest rozszerzony klient transakcyjny? .

## *Korzystanie z odczytu z wyprzedzeniem w aplikacji klienckiej*

Można użyć odczytu z wyprzedzeniem na kliencie, aby umożliwić wysyłanie nietrwałych komunikatów do klienta bez konieczności żądania komunikatów przez aplikację kliencką.

Gdy klient wymaga komunikatu od serwera, wysyła żądanie do serwera. Wysyła osobne żądanie dla każdego komunikatu, który konsumuje. Aby zwiększyć wydajność klienta korzystającego z nietrwałych komunikatów, unikając konieczności wysyłania tych komunikatów żądań, można skonfigurować klienta do korzystania z odczytu z wyprzedzeniem. Odczyt z wyprzedzeniem umożliwia wysyłanie komunikatów do klienta bez konieczności żądania ich przez aplikację.

Użycie odczytu z wyprzedzeniem może zwiększyć wydajność podczas konsumowania nietrwałych komunikatów z aplikacji klienckiej. Ten wzrost wydajności jest dostępny zarówno dla aplikacji MQI, jak i JMS . Aplikacje klienckie używające usługi MQGET lub wykorzystania asynchronicznego korzystają z poprawy wydajności podczas korzystania z nietrwałych komunikatów.

W przypadku wywołania MQOPEN z opcją MQOO_READ_AHEAD klient IBM MQ umożliwia odczyt z wyprzedzeniem, jeśli spełnione są pewne warunki. Są one następujące:

- Aplikacja kliencka musi zostać skompilowana i powiązana z wątkowymi bibliotekami klienta MQI IBM MQ.
- Kanał klienta musi używać protokołu TCP/IP.
- Ustawienie SharingConversations (SHARECNV) kanału musi mieć wartość niezerową w definicji kanału zarówno klienta, jak i serwera.

Gdy odczyt z wyprzedzeniem jest włączony, komunikaty są wysyłane do buforu pamięci na kliencie nazywanego buforem odczytu z wyprzedzeniem. Klient ma bufor odczytu z wyprzedzeniem dla każdej

kolejki, która jest otwarta z włączonym odczytem z wyprzedzeniem. Komunikaty w buforze odczytu z wyprzedzeniem nie są utrwalane. Klient okresowo aktualizuje serwer informacjami o ilości danych, które wykorzystał.

Nie wszystkie projekty aplikacji klienckich nadają się do korzystania z odczytu z wyprzedzeniem, ponieważ nie wszystkie opcje są obsługiwane. Niektóre opcje muszą być spójne między wywołaniami MQGET, gdy włączony jest odczyt z wyprzedzeniem. Jeśli klient zmienia kryteria wyboru między wywołaniami MQGET, komunikaty przechowywane w buforze odczytu z wyprzedzeniem pozostają porzucone w buforze odczytu z wyprzedzeniem klienta. Więcej informacji na ten temat zawiera sekcja "Zwiększanie wydajności komunikatów nietrwałych" na stronie 811.

Konfiguracją odczytu z wyprzedzeniem sterują trzy atrybuty: MaximumSize, PurgeTimei UpdatePercentage, które są określone w sekcji MessageBuffer pliku konfiguracyjnego klienta IBM MQ .

## *Używanie asynchronicznego umieszczania w aplikacji klienckiej*

Za pomocą asynchronicznego umieszczania aplikacja może umieścić komunikat w kolejce bez oczekiwania na odpowiedź z menedżera kolejek. W niektórych sytuacjach można użyć tej opcji, aby zwiększyć wydajność przesyłania komunikatów.

Zwykle, gdy aplikacja umieszcza komunikat lub komunikaty w kolejce za pomocą wywołania MQPUT lub MQPUT1, musi czekać, aż menedżer kolejek potwierdzi przetworzenie żądania MQI. Wydajność przesyłania komunikatów można zwiększyć, szczególnie w przypadku aplikacji, które używają powiązań klienta, oraz w przypadku aplikacji, które umieszczają w kolejce dużą liczbę małych komunikatów, wybierając opcję asynchronicznego umieszczania komunikatów. Jeśli aplikacja umieszcza komunikat asynchronicznie, menedżer kolejek nie zwraca informacji o powodzeniu lub niepowodzeniu każdego wywołania, ale można sprawdzać okresowo występowanie błędów.

Aby umieścić komunikat w kolejce asynchronicznie, należy użyć opcji MQPMO_ASYNC_RESPONSE w polu *Options* struktury MQPMO.

Jeśli komunikat nie jest zakwalifikowany do asynchronicznego umieszczania, jest umieszczany w kolejce synchronicznie.

W przypadku żądania asynchronicznej odpowiedzi umieszczania dla MQPUT lub MQPUT1wartość CompCode i Reason of MQCC_OK oraz MQRC_NONE nie musi oznaczać, że komunikat został pomyślnie umieszczony w kolejce. Mimo że powodzenie lub niepowodzenie każdego pojedynczego wywołania MQPUT lub MQPUT1 może nie zostać zwrócone natychmiast, pierwszy błąd, który wystąpił w ramach wywołania asynchronicznego, można określić później za pomocą wywołania MQSTAT.

Więcej informacji na temat opcji MQPMO_ASYNC_RESPONSE zawiera sekcja Opcje MQPMO.

Przykładowy program Asynchronous Put demonstruje niektóre z dostępnych funkcji. Szczegółowe informacje na temat funkcji i projektu programu oraz sposobu jego uruchamiania zawiera sekcja "Przykładowy program asynchronicznego umieszczania" na stronie 1109.

## *Współużytkowanie konwersacji w aplikacji klienckiej*

W środowisku, w którym współużytkowanie konwersacji jest dozwolone, konwersacje mogą współużytkować instancję kanału MQI.

Współużytkowanie konwersacji jest sterowane przez dwa pola, oba nazywane SharingConversations, z których jedno jest częścią struktury definicji kanału (MQCD), a drugie jest częścią struktury parametru wyjścia kanału (MQCXP). Pole SharingConversations w MQCD jest liczbą całkowitą określającą maksymalną liczbę konwersacji, które mogą współużytkować instancję kanału powiązaną z kanałem. Pole SharingConversations w MQCXP jest wartością boolowską wskazującą, czy instancja kanału jest obecnie współużytkowana.

W środowisku, w którym współużytkowanie konwersacji nie jest dozwolone, nowe połączenia klienckie określające identyczne dyski MQCD nie będą współużytkować instancji kanału.

Nowe połączenie aplikacji klienckiej będzie współużytkować instancję kanału, jeśli spełnione zostaną następujące warunki:

- Zarówno końcówki połączenia klienta, jak i połączenia serwera instancji kanału są skonfigurowane na potrzeby współużytkowania konwersacji, a wartości te nie są nadpisywane przez wyjścia kanału.
- Wartość MQCD połączenia klienckiego (podana w wywołaniu MQCONNX klienta lub w tabeli definicji kanału klienta (CCDT)) jest dokładnie zgodna z wartością MQCD połączenia klienckiego podaną w wywołaniu MQCONNX klienta lub z tabeli CCDT podczas pierwszego ustanowienia istniejącej instancji kanału. Należy zauważyć, że pierwotny plik MQCD mógł zostać później zmieniony przez wyjścia lub negocjację kanału, ale zgodność jest porównywana z wartością, która została dostarczona do systemu klienckiego przed wprowadzeniem tych zmian.
- Limit współużytkowania konwersacji po stronie serwera nie został przekroczony.

Jeśli nowe połączenie aplikacji klienckiej jest zgodne z kryteriami uruchamiania współużytkowania instancji kanału z innymi konwersacjami, ta decyzja jest podejmowana przed wywołaniem wyjścia z tej konwersacji. Zakończenie takiej konwersacji nie może zmienić faktu, że współużytkuje ona instancję kanału z innymi konwersacjami. Jeśli nie ma istniejących instancji kanału zgodnych z nową definicją kanału, zostanie podłączona nowa instancja kanału.

Negocjacja kanału ma miejsce tylko w przypadku pierwszej konwersacji w instancji kanału. Negocjowane wartości dla instancji kanału są stałe na tym etapie i nie można ich zmienić podczas uruchamiania kolejnych konwersacji. Uwierzytelnianie TLS ma również miejsce tylko w przypadku pierwszej konwersacji.

Jeśli wartość SharingConversations dla MQCD zostanie zmieniona podczas inicjowania jakichkolwiek zabezpieczeń, wyjść wysyłania lub odbierania dla pierwszej konwersacji w gnieździe na końcu połączenia z klientem lub na końcu połączenia z serwerem instancji kanału, nowa wartość, którą ma po zainicjowaniu wszystkich wyjść, zostanie użyta do określenia wartości konwersacji współużytkowania dla instancji kanału (najniższa wartość ma pierwszeństwo).

Jeśli negocjowana wartość dla współużytkowania konwersacji wynosi zero, instancja kanału nigdy nie jest współużytkowana. Kolejne programy obsługi wyjścia, które ustawiły w tym polu wartość zero, są podobnie uruchamiane w ich własnej instancji kanału.

Jeśli negocjowana wartość dla konwersacji współużytkowanych jest większa od zera, opcja SharingConversations dla MQCXP jest ustawiona na wartość TRUE dla kolejnych wywołań wyjść, co oznacza, że inne programy obsługi wyjścia w tej instancji kanału mogą być wprowadzane jednocześnie z tym programem.

Podczas pisania programu obsługi wyjścia kanału należy rozważyć, czy będzie on uruchamiany w instancji kanału, która może wymagać współużytkowania konwersacji. Jeśli instancja kanału może obejmować współużytkowanie konwersacji, należy rozważyć wpływ zmiany pól MQCD na inne instancje wyjścia kanału. Wszystkie pola MQCD mają wspólne wartości we wszystkich konwersacjach współużytkowanych. Jeśli po ustanowieniu instancji kanału programy obsługi wyjścia podejmą próbę zmiany pól MQCD, mogą napotkać problemy, ponieważ inne instancje programów obsługi wyjścia działające w instancji kanału mogą próbować zmienić te same pola w tym samym czasie. Jeśli taka sytuacja może wystąpić w przypadku programów obsługi wyjścia, należy przekształcić do postaci szeregowej dostęp do MQCD w kodzie wyjścia.

W przypadku pracy z kanałem, który jest zdefiniowany do współużytkowania konwersacji, ale nie ma potrzeby współużytkowania w konkretnej instancji kanału, należy ustawić wartość MQCD SharingConversations na 1 lub 0 podczas inicjowania wyjścia kanału w pierwszej konwersacji w instancji kanału. Sekcja SharingConversations zawiera wyjaśnienie wartości atrybutu SharingConversations.

**Przykład**

Współużytkowanie konwersacji jest włączone.

Używana jest definicja kanału połączenia klienckiego, która określa program obsługi wyjścia.

Przy pierwszym uruchomieniu tego kanału program obsługi wyjścia zmienia niektóre parametry MQCD podczas inicjowania. Są one wykonywane przez kanał, dlatego definicja, z którą działa kanał, jest teraz inna niż ta, która została pierwotnie podana. Parametr SharingConversations systemu MQCXP jest ustawiony na wartość TRUE.

Następnym razem, gdy aplikacja nawiąże połączenie przy użyciu tego kanału, konwersacja będzie działać w instancji kanału, która została wcześniej uruchomiona, ponieważ ma tę samą definicję kanału. Instancja kanału, z którą aplikacja łączy się za drugim razem, jest tą samą instancją, co przy pierwszym połączeniu. W związku z tym używa definicji, które zostały zmienione przez program obsługi wyjścia. Jeśli program obsługi wyjścia jest inicjowany dla drugiej konwersacji, mimo że może zmieniać pola MQCD, nie są one wykonywane przez kanał. Te same parametry mają zastosowanie do wszystkich kolejnych konwersacji, które współużytkują instancję kanału.

### *Korzystanie z usługi MQCONNX*

Za pomocą wywołania MQCONNX można określić strukturę definicji kanału (MQCD) w strukturze MQCNO.

Dzięki temu wywołująca aplikacja kliencka może określić definicję kanału połączenia klienckiego w czasie wykonywania. Więcej informacji na ten temat zawiera sekcja Tworzenie kanału połączenia klienckiego w produkcie IBM MQ MQI client przy użyciu usługi MQCNO. Jeśli używana jest opcja MQCONNX, wywołanie wykonywane na serwerze zależy od poziomu serwera i konfiguracji programu nasłuchującego.

Jeśli komenda MQCONNX jest używana z klienta, następujące opcje są ignorowane:

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING,

Struktura MQCD, której można użyć, zależy od używanego numeru wersji MQCD. Informacje na temat wersji MQCD (MQCD_VERSION) zawiera sekcja Wersja MQCD. Struktury MQCD można użyć na przykład do przekazania programów obsługi wyjścia kanału do serwera. Jeśli używany jest produkt MQCD w wersji 3 lub nowszej, można użyć struktury do przekazania tablicy wyjść do serwera. Funkcji tej można użyć do wykonania więcej niż jednej operacji na tym samym komunikacie, takiej jak szyfrowanie i kompresja, przez dodanie wyjścia dla każdej operacji zamiast modyfikowania istniejącego wyjścia. Jeśli w strukturze MQCD nie zostanie podana tablica, zostaną sprawdzone pola pojedynczego wyjścia. Więcej informacji na temat programów obsługi wyjścia kanału zawiera sekcja “Kanał-programy obsługi wyjścia dla kanałów przesyłania komunikatów” na stronie 990.

*Współużytkowane uchwyty połączeń w MQCONNX*

Za pomocą współużytkowanych uchwytów połączeń można współużytkować uchwyty między różnymi wątkami w ramach tego samego procesu.

Jeśli zostanie określony uchwyt połączenia współużytkowanego, uchwyt połączenia zwrócony przez wywołanie MQCONNX może zostać przekazany w kolejnych wywołaniach MQI w dowolnym wątku w procesie.

**Uwaga:** Aby nawiązać połączenie z menedżerem kolejek serwera, który nie obsługuje uchwytów połączeń współużytkowanych, można użyć uchwytu połączenia współużytkowanego w systemie IBM MQ MQI client .

Więcej informacji na ten temat zawiera sekcja “Korzystanie z usługi MQCONNX” na stronie 947.

## Budowanie aplikacji dla produktu IBM MQ MQI clients

Aplikacje można budować i uruchamiać w środowisku IBM MQ MQI client . Aplikacja musi zostać zbudowana i powiązana z używanym IBM MQ MQI client . Sposób, w jaki aplikacje są budowane i łączone, zależy od używanej platformy i języka programowania.

Jeśli aplikacja ma być uruchamiana w środowisku klienta, można ją zapisać w językach przedstawionych w poniższej tabeli:

*Tabela 133. Języki programowania obsługiwane w środowiskach klienckich*

| Platforma klienta | C | C++ | kompilatory | pTAL | RPG | Visual Basic |
|---|---|---|---|---|---|---|
| AIX | Tak | Tak | Tak | | | |
| IBM i | Tak | | Tak | | Tak | |

| *Tabela 133. Języki programowania obsługiwane w środowiskach klienckich (kontynuacja)* | | | | | | |
|---|---|---|---|---|---|---|
| **Platforma klienta** | **C** | **C++** | **kompilatory** | **pTAL** | **RPG** | **Visual Basic** |
| Linux Linux | Tak | Tak | Tak | | | |
| Windows Windows | Tak | Tak | Tak | | | Tak |

Multi

## *Łączenie aplikacji w języku C z kodem IBM MQ MQI client*

Po napisaniu aplikacji IBM MQ , która ma być uruchamiana na serwerze IBM MQ MQI client, należy połączyć ją z kodem IBM MQ MQI client .

Aplikację można połączyć z kodem IBM MQ MQI client na dwa sposoby:

1. Bezpośrednio, przez połączenie aplikacji z menedżerem kolejek. W takim przypadku menedżer kolejek musi znajdować się na tym samym komputerze, co aplikacja.
2. Do pliku biblioteki klienta, który umożliwia dostęp do menedżerów kolejek na tym samym lub na innym komputerze.

IBM MQ udostępnia plik biblioteki klienta dla każdego środowiska:

AIX **AIX**
Biblioteka libmqic.a dla aplikacji niewielowątkowych lub biblioteka libmqic_r.a dla aplikacji wielowątkowych.

Linux **Linux**
Biblioteka libmqic.so dla aplikacji niewielowątkowych lub biblioteka libmqic_r.so dla aplikacji wielowątkowych.

IBM i **IBM i**
Powiąż aplikację kliencką z programem usługowym klienta LIBMQIC dla aplikacji niewielowątkowych lub programem usługowym LIBMQIC_R dla aplikacji wielowątkowych.

Windows **Windows**
MQIC32.LIB.

ALW

## *Łączenie aplikacji C++ z kodem IBM MQ MQI client*

Istnieje możliwość napisania aplikacji uruchamianych na kliencie w języku C + +. Metody budowania różnią się w zależności od środowiska.

Informacje na temat łączenia aplikacji w języku C++ zawiera sekcja Budowanie programów w języku IBM MQ C++.

Szczegółowe informacje na temat wszystkich aspektów korzystania z języka C + + zawiera sekcja Korzystanie z języka C++

Multi

## *Łączenie aplikacji w języku COBOL z kodem IBM MQ MQI client*

Po napisaniu aplikacji w języku COBOL, która ma być uruchamiana na serwerze IBM MQ MQI client, należy połączyć ją z odpowiednią biblioteką.

IBM MQ udostępnia plik biblioteki klienta dla każdego środowiska:

AIX **AIX**
Połącz niewielowątkową aplikację w języku COBOL z biblioteką libmqicb.a lub wielowątkową aplikację w języku COBOL z biblioteką libmqicb_r.a.

IBM i **IBM i**
Powiąż aplikację kliencką w języku COBOL z programem usługowym AMQCSTUB dla aplikacji niewielowątkowych lub program usługowy AMQCSTUB_R dla aplikacji wielowątkowych.

Windows **Windows**

Połącz kod aplikacji z biblioteką MQICCBB dla 32-bitowego języka COBOL. IBM MQ MQI client for Windows nie obsługuje 16-bitowego języka COBOL.

### Windows *Łączenie aplikacji Visual Basic z kodem IBM MQ MQI client*

Aplikacje Microsoft Visual Basic można połączyć za pomocą kodu IBM MQ MQI client w systemie Windows.

Deprecated

Począwszy od wersji IBM MQ 9.0, obsługa języka Microsoft Visual Basic 6.0 jest nieaktualna. Zalecaną technologią zastępczą są klasy IBM MQ dla .NET. Aby uzyskać więcej informacji, zapoznaj się z sekcją: Tworzenie aplikacji .NET.

Połącz aplikację Visual Basic z następującymi plikami włączanymi:

**CMQB.bas**
MQI

**CMQBB.bas**
MQAI

**CMQCFB.bas**
Komendy PCF

**CMQXB.bas**
Kanały

Ustaw wartość `mqtype=2` dla klienta w kompilatorze Visual Basic , aby zapewnić poprawny automatyczny wybór biblioteki DLL klienta:

**MQIC32.dll**
Windows 7, Windows 8, Windows 2008 i Windows 2012

**Pojęcia pokrewne**

"Kodowanie w języku Visual Basic" na stronie 1084
Informacje, które należy wziąć pod uwagę podczas kodowania programów IBM MQ w języku Microsoft Visual Basic. Parametr Visual Basic jest obsługiwany tylko w systemie Windows.

"Przygotowywanie programów Visual Basic w systemie Windows" na stronie 1050
Informacje, które należy wziąć pod uwagę podczas używania programów Microsoft Visual Basic w systemie Windows.

## Uruchamianie aplikacji w środowisku IBM MQ MQI client

Aplikację IBM MQ można uruchomić zarówno w pełnym środowisku IBM MQ , jak i w środowisku IBM MQ MQI client bez zmiany kodu, o ile spełnione są pewne warunki.

Warunki te są następujące:

- Aplikacja nie musi łączyć się jednocześnie z więcej niż jednym menedżerem kolejek.
- Nazwa menedżera kolejek nie jest poprzedzona gwiazdką (*) w wywołaniu `MQCONN` lub `MQCONNX` .
- Aplikacja nie musi korzystać z żadnego z wyjątków wymienionych w sekcji Jakie aplikacje działają w systemie IBM MQ MQI client?

**Uwaga:** Biblioteki używane podczas konsolidacji określają środowisko, w którym musi działać aplikacja.

Podczas pracy w środowisku IBM MQ MQI client należy pamiętać, że:

- Każda aplikacja działająca w środowisku IBM MQ MQI client ma własne połączenia z serwerami. Aplikacja nawiązuje jedno połączenie z serwerem za każdym razem, gdy wysyła wywołanie `MQCONN` lub `MQCONNX` .
- Aplikacja wysyła i pobiera komunikaty synchronicznie. Oznacza to oczekiwanie między wysłaniem wywołania przez klienta a zwrótem kodu zakończenia i kodu przyczyny w sieci.

- Wszystkie konwersje danych są wykonywane przez serwer, ale informacje na temat nadpisywania skonfigurowanego identyfikatora CCSID maszyny zawiera sekcja MQCCSID .

### *Łączenie aplikacji IBM MQ MQI client z menedżerami kolejek*

Aplikacja działająca w środowisku IBM MQ MQI client może łączyć się z menedżerem kolejek na różne sposoby. Można użyć zmiennych środowiskowych, struktury MQCNO lub tabeli definicji klienta.

Gdy aplikacja działająca w środowisku klienta IBM MQ wywołuje komendę MQCONN lub MQCONNX, klient identyfikuje sposób nawiązywania połączenia. Gdy wywołanie MQCONNX jest wykonywane przez aplikację na kliencie IBM MQ , biblioteka klienta MQI wyszukuje informacje o kanale klienta w następującej kolejności:

1. Przy użyciu zawartości pól `ClientConnOffset` lub `ClientConnPtr` struktury MQCNO (jeśli została podana). Pola te identyfikują strukturę definicji kanału (MQCD), która ma być używana jako definicja kanału połączenia klienckiego. Szczegóły połączenia można przesłonić przy użyciu wyjścia połączenia wstępnego. Więcej informacji na ten tematzawiera sekcja “Odwoływanie się do definicji połączeń przy użyciu wyjścia przed nawiązaniem połączenia z repozytorium” na stronie 1022.
2. Jeśli zmienna środowiskowa **MQSERVER** jest ustawiona, używany jest zdefiniowany przez nią kanał.
3. Jeśli plik `mqclient.ini` jest zdefiniowany, a sekcja Kanały zawiera atrybut **ServerConnectionParms** , używany jest kanał, który definiuje. Więcej informacji na ten temat zawiera sekcja Plik konfiguracyjnyIBM MQ MQI client , sekcja `mqclient.ini` i sekcja Kanały w pliku konfiguracyjnym klienta.
4. Jeśli zmienne środowiskowe **MQCHLLIB** i **MQCHLTAB** są ustawione, używana jest tabela definicji kanału klienta, na którą wskazują. Alternatywnie, w systemie IBM MQ 9.0, zmienna środowiskowa **MQCCDTURL** umożliwia ustawienie kombinacji zmiennych środowiskowych **MQCHLLIB** i **MQCHLTAB** . Jeśli ustawiona jest wartość **MQCCDTURL** , używana jest wskazywana przez nią tabela definicji kanału klienta. Więcej informacji na ten temat zawiera sekcja Adres URLURL dostępu do tabeli definicji kanału klienta.
5. Jeśli plik `mqclient.ini` jest zdefiniowany, a sekcja Kanały zawiera atrybuty **ChannelDefinitionDirectory** i **ChannelDefinitionFile** , atrybuty te są używane do zlokalizowania tabeli definicji kanału klienta. Więcej informacji na ten temat zawiera sekcja Plik konfiguracyjnyIBM MQ MQI client , sekcja `mqclient.ini` i sekcja Kanały w pliku konfiguracyjnym klienta.
6. Na koniec, jeśli zmienne środowiskowe nie są ustawione, klient wyszukuje tabelę definicji kanału klienta ze ścieżką i nazwą, które są określone na podstawie atrybutu **DefaultPrefix** w sekcji AllQueueManagers w pliku `mqs.ini` . Więcej informacji na ten temat zawiera sekcja AllQueueManagers w pliku mqs.ini.

   Jeśli wyszukiwanie tabeli definicji kanału klienta nie powiedzie się, klient użyje następujących ścieżek:

   - Linux AIX W systemie AIX and Linux: `/var/mqm/AMQCLCHL.TAB`
   - Windows W systemie Windows: `C:\Program Files\IBM\MQ\amqclchl.tab`
   - IBM i W systemie IBM i: `/QIBM/UserData/mqm/@ipcc`
   - MQ Appliance W systemie IBM MQ Appliance: *QMname*`_AMQCLCHL.TAB`. Są one wyświetlane w obszarze `mqbackup:// URI`.

Pierwsza z opcji opisanych na poprzedniej liście (przy użyciu pól `ClientConnOffset` lub `ClientConnPtr` MQCNO) jest obsługiwana tylko przez wywołanie MQCONNX. Jeśli aplikacja używa MQCONN, a nie MQCONNX, informacje o kanale są wyszukiwane na pozostałe pięć sposobów w kolejności pokazanej na liście. Jeśli klient nie może znaleźć informacji o kanale, wywołanie MQCONN lub MQCONNX kończy się niepowodzeniem.

Aby wywołanie MQCONN lub MQCONNX powiodło się, nazwa kanału (dla połączenia klienta) musi być zgodna z nazwą kanału połączenia z serwerem zdefiniowaną na serwerze.

**Pojęcia pokrewne**

Dostęp WWW do tabeli definicji kanału klienta

**Zadania pokrewne**

Konfigurowanie połączeń między serwerem i klientem

**Odsyłacze pokrewne**

Tabela definicji kanału klienta

MQCNO-opcje połączenia

*Łączenie aplikacji klienckich z menedżerami kolejek przy użyciu zmiennych środowiskowych*
Informacje o kanale klienta mogą być dostarczane do aplikacji działającej w środowisku klienta za pomocą zmiennych środowiskowych.

Aplikacja działająca w środowisku IBM MQ MQI client może nawiązać połączenie z menedżerem kolejek przy użyciu następujących zmiennych środowiskowych:

**SERWER MQSERVER**
: Zmienna środowiskowa **MQSERVER** jest używana do zdefiniowania minimalnego kanału. **MQSERVER** określa położenie serwera IBM MQ i metodę komunikacji, która ma być używana.

**Biblioteka MQCHLLIB**
: Zmienna środowiskowa **MQCHLLIB** określa ścieżkę do pliku zawierającego tabelę definicji kanału klienta (CCDT). Plik jest tworzony na serwerze, ale można go skopiować na stację roboczą IBM MQ MQI client .

**MQCHLTAB**
: Zmienna środowiskowa **MQCHLTAB** określa nazwę pliku zawierającego tabelę definicji kanału klienta (CCDT).

W systemie IBM MQ 9.0zmienna środowiskowa **MQCCDTURL** jest równoważna możliwości ustawienia kombinacji zmiennych środowiskowych **MQCHLLIB** i **MQCHLTAB** . **MQCCDTURL** umożliwia podanie pliku, ftp lub http URL jako pojedynczej wartości, z której można uzyskać tabelę definicji kanału klienta. Więcej informacji na ten temat zawiera sekcja Dostęp do tabeli definicji kanału klienta z możliwością adresowania w sieci WWW.

*Łączenie aplikacji klienckich z menedżerami kolejek przy użyciu struktury MQCNO*
Definicję kanału można określić w strukturze definicji kanału (MQCD), która jest dostarczana przy użyciu struktury MQCNO wywołania MQCONNX.

Więcej informacji na ten temat zawiera sekcja Tworzenie kanału połączenia klienckiego w produkcie IBM MQ MQI client przy użyciu usługi MQCNO.

*Łączenie aplikacji klienckich z menedżerami kolejek przy użyciu tabeli definicji kanału klienta*
Jeśli używana jest komenda MQSC DEFINE CHANNEL, podane szczegóły są umieszczane w tabeli definicji kanału klienta (ccdt). Treść parametru **QMgrName** wywołania MQCONN lub MQCONNX określa, z którym menedżerem kolejek łączy się klient.

Klient uzyskuje dostęp do tego pliku w celu określenia kanału, który będzie używany przez aplikację. Jeśli istnieje więcej niż jedna odpowiednia definicja kanału, na wybór kanału mają wpływ atrybuty kanału wagi kanału klienta (CLNTWGHT) i powinowactwa połączenia (AFFINITY).

*Używanie automatycznego ponownego połączenia klienta*
Istnieje możliwość automatycznego ponownego nawiązania połączenia przez aplikacje klienckie bez pisania dodatkowego kodu. W tym celu należy skonfigurować wiele komponentów.

Automatyczne ponowne łączenie klienta jest *bezpośrednie*. Połączenie jest automatycznie przywracane w dowolnym momencie w programie aplikacji klienckiej. Przywracane są również wszystkie uchwyty umożliwiające otwieranie obiektów.

Z drugiej strony ręczne ponowne nawiązanie połączenia wymaga odtworzenia połączenia przez aplikację kliencką przy użyciu wywołania MQCONN lub MQCONNX oraz ponownego otwarcia obiektów. Automatyczne ponowne nawiązanie połączenia klienta jest odpowiednie w przypadku wielu aplikacji klienckich, ale nie wszystkich.

Więcej informacji na ten temat zawiera sekcja Automatyczne ponowne łączenie klienta.

*Rola tabeli definicji kanału klienta*
Tabela definicji kanału klienta (CCDT) zawiera definicje kanałów połączenia klienta. Jest to szczególnie przydatne, gdy aplikacje klienckie mogą wymagać nawiązania połączenia z wieloma alternatywnymi menedżerami kolejek.

Tabela definicji kanału klienta jest tworzona podczas definiowania menedżera kolejek. Ten sam plik może być używany przez więcej niż jednego klienta IBM MQ .

Istnieje wiele sposobów korzystania z tabeli definicji kanału klienta przez aplikację kliencką. Tabelę CCDT można skopiować na komputer kliencki. Tabelę definicji kanału klienta można skopiować do położenia współużytkowanego przez więcej niż jednego klienta. Tabelę CCDT można udostępnić klientowi jako plik współużytkowany, gdy pozostaje ona na serwerze.

W produkcie IBM MQ 9.0tabela CCDT może być udostępniana w centralnej lokalizacji dostępnej za pośrednictwem identyfikatora URI, co eliminuje konieczność indywidualnego aktualizowania tabeli CCDT dla każdego wdrożonego klienta.

**Pojęcia pokrewne**
Dostęp WWW do tabeli definicji kanału klienta
**Zadania pokrewne**
Uzyskiwanie dostępu do definicji kanału połączenia klienckiego
**Odsyłacze pokrewne**
Tabela definicji kanału klienta

*Grupy menedżerów kolejek w tabeli definicji kanału klienta*
Zestaw połączeń można zdefiniować w tabeli definicji kanału klienta (CCDT) jako *grupę menedżerów kolejek*. Aplikację można połączyć z menedżerem kolejek, który jest częścią grupy menedżerów kolejek. Można to zrobić, poprzedzając nazwę menedżera kolejek w wywołaniu programu `MQCONN` lub `MQCONNX` gwiazdką.

Można zdefiniować połączenia z więcej niż jednym serwerem, ponieważ:

- W celu zwiększenia dostępności należy połączyć klienta z jednym z uruchomionych menedżerów kolejek.
- Klient ma zostać ponownie połączony z tym samym menedżerem kolejek, z którym został pomyślnie połączony w ostatnim czasie, ale w przypadku niepowodzenia połączenia ma zostać nawiązane połączenie z innym menedżerem kolejek.
- W przypadku niepowodzenia połączenia należy ponowić próbę nawiązania połączenia klienta z innym menedżerem kolejek, wprowadzając ponownie komendę `MQCONN` w programie klienckim.
- Użytkownik chce automatycznie ponownie nawiązać połączenie klienta z innym menedżerem kolejek, jeśli połączenie nie powiedzie się, bez zapisywania kodu klienta.
- Użytkownik chce automatycznie ponownie połączyć połączenie klienta z inną instancją menedżera kolejek z wieloma instancjami, jeśli instancja rezerwowa przejmie kontrolę, bez zapisywania kodu klienta.
- Użytkownik chce zrównoważyć połączenia klienckie między wieloma menedżerami kolejek, przy czym z niektórymi menedżerami kolejek łączy się więcej klientów niż z innymi.
- Użytkownik chce rozłożyć ponowne połączenie wielu połączeń klienckich na wiele menedżerów kolejek i w czasie, jeśli duża liczba połączeń powoduje awarię.
- Użytkownik chce mieć możliwość przenoszenia menedżerów kolejek bez zmiany kodu aplikacji klienckiej.
- Użytkownik chce napisać aplikacje klienckie, które nie muszą znać nazw menedżerów kolejek.

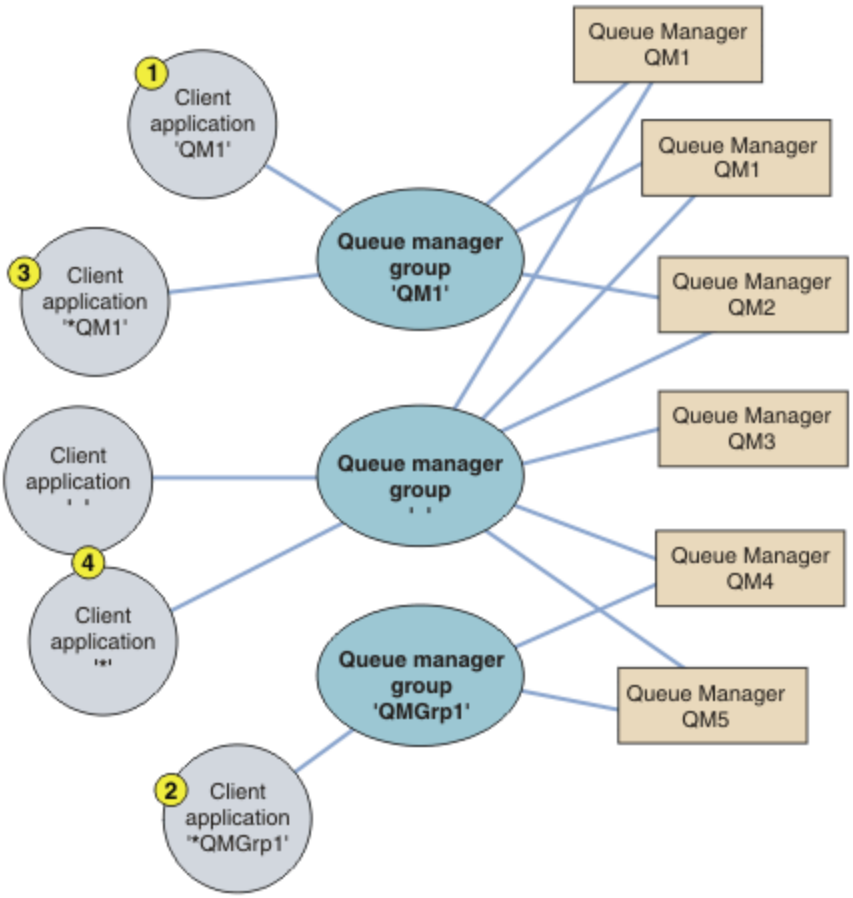
Nie zawsze jest właściwe nawiązywanie połączenia z różnymi menedżerami kolejek. Na przykład w przypadku rozszerzonego klienta transakcyjnego lub klienta Java w produkcie WebSphere Application Servermoże być konieczne nawiązanie połączenia z przewidywalną instancją menedżera kolejek.

Automatyczne ponowne nawiązywanie połączenia przez klient nie jest obsługiwane przez produkt IBM MQ classes for Java.

Grupa menedżerów kolejek jest zestawem połączeń zdefiniowanych w tabeli definicji kanału klienta (CCDT). Zestaw jest definiowany przez jego elementy, które w swoich definicjach kanałów mają taką samą wartość atrybutu **QMNAME** .

Rysunek 97 na stronie 953 jest graficzną reprezentacją tabeli połączeń klienta, w której przedstawiono trzy grupy menedżerów kolejek, dwie nazwane grupy menedżerów kolejek zapisane w tabeli definicji kanału klienta jako **QMNAME** (QM1) i **QMNAME** (QMGrp1)oraz jedną pustą lub domyślną grupę zapisaną jako **QMNAME** (' ').

1. Grupa menedżerów kolejek QM1 ma trzy kanały połączenia klienta, łącząc się z menedżerami kolejek QM1 i QM2. QM1 może być menedżerem kolejek z wieloma instancjami znajdującym się na dwóch różnych serwerach.
2. Domyślna grupa menedżerów kolejek ma sześć kanałów połączenia klienta łączących ją ze wszystkimi menedżerami kolejek.
3. QMGrp1 zawiera kanały połączenia klienta z dwoma menedżerami kolejek, QM4 i QM5.




*Rysunek 97. Grupy menedżerów kolejek*

Cztery przykłady użycia tej tabeli połączeń klienckich zostały opisane w pomocy numerowanych aplikacji klienckich w sekcji Rysunek 97 na stronie 953.

1. W pierwszym przykładzie aplikacja kliencka przekazuje nazwę menedżera kolejek QM1jako parametr **QmgrName** do wywołania MQI produktu MQCONN lub MQCONNX . Kod klienta IBM MQ wybiera zgodną grupę menedżerów kolejek QM1. Grupa zawiera trzy kanały połączenia, a IBM MQ MQI client próbuje połączyć się z QM1 za pomocą każdego z tych kanałów, aż znajdzie program nasłuchujący IBM MQ dla połączenia przyłączonego do działającego menedżera kolejek o nazwie QM1.

Kolejność prób połączenia zależy od wartości atrybutu AFFINITY połączenia klienta i wag kanału klienta. W tych ograniczeniach kolejność prób połączenia jest losowa, zarówno w trzech możliwych połączeniach, jak i w czasie, w celu rozłożenia obciążenia związanego z nawiązyowaniem połączeń.

Wywołanie MQCONN lub MQCONNX wysłane przez aplikację kliencką powiodło się po nawiązaniu połączenia z działającą instancją QM1.

2. W drugim przykładzie aplikacja kliencka przekazuje nazwę menedżera kolejek z przedrostkiem w postaci gwiazdki *QMGrp1 jako parametr **QmgrName** do wywołania MQI produktu MQCONN lub MQCONNX . Klient IBM MQ wybiera zgodną grupę menedżerów kolejek QMGrp1. Ta grupa zawiera dwa kanały połączenia klienta, a program IBM MQ MQI client próbuje połączyć się z *dowolnym* menedżerem kolejek, używając kolejno każdego kanału. W tym przykładzie program IBM MQ MQI client musi pomyślnie nawiązać połączenie. Nazwa menedżera kolejek, z którym nawiązuje połączenie, nie ma znaczenia.

   Reguła kolejności prób nawiązania połączenia jest taka sama, jak wcześniej. Jedyna różnica polega na tym, że przez poprzedzenie nazwy menedżera kolejek gwiazdką klient wskazuje, że nazwa menedżera kolejek nie jest odpowiednia.

   Wywołanie MQCONN lub MQCONNX wykonywane przez aplikację kliencką powiedzie się, gdy połączenie zostanie nawiązane z działającą instancją dowolnego menedżera kolejek połączonego kanałami z grupy menedżerów kolejek QMGrp1 .

3. Trzeci przykład jest zasadniczo taki sam jak drugi, ponieważ parametr **QmgrName** jest poprzedzony gwiazdką *QM1. W przykładzie pokazano, że nie można określić, z którym menedżerem kolejek ma zostać połączone połączenie kanału klienta, sprawdzając atrybut QMNAME w jednej definicji kanału. Fakt, że atrybut **QMNAME** definicji kanału ma wartość QM1, nie jest wystarczający, aby wymagać połączenia z menedżerem kolejek o nazwie QM1. Jeśli aplikacja kliencka poprzedza swój parametr **QmgrName** gwiazdką, każdy menedżer kolejek jest możliwym miejscem docelowym połączenia.

   W tym przypadku wywołania MQCONN lub MQCONNX wykonane przez aplikację kliencką powiodą się po nawiązaniu połączenia z działającą instancją QM1 lub QM2.

4. Czwarty przykład ilustruje użycie grupy domyślnej. W tym przypadku aplikacja kliencka przekazuje gwiazdkę '*' lub pustą wartość '' jako parametr **QmgrName** w wywołaniu MQI produktu MQCONN lub MQCONNX . Zgodnie z przyjętą konwencją w definicji kanału klienta pusty atrybut **QMNAME** oznacza domyślną grupę menedżerów kolejek, a parametr **QmgrName** (pusty) lub gwiazdka (gwiazdka) jest zgodny z pustym atrybutem **QMNAME** .

   W tym przykładzie domyślna grupa menedżerów kolejek ma połączenia kanału klienta ze wszystkimi menedżerami kolejek. Po wybraniu domyślnej grupy menedżerów kolejek aplikacja może być połączona z dowolnym menedżerem kolejek w grupie.

   Wywołanie MQCONN lub MQCONNX wysłane przez aplikację kliencką zakończy się powodzeniem po nawiązaniu połączenia z działającą instancją dowolnego menedżera kolejek.

   **Uwaga:** Domyślna grupa jest inna niż domyślny menedżer kolejek, chociaż aplikacja używa pustego parametru **QmgrName** do nawiązania połączenia z domyślną grupą menedżerów kolejek lub z domyślnym menedżerem kolejek. Pojęcie domyślnej grupy menedżerów kolejek dotyczy tylko aplikacji klienckiej i domyślnego menedżera kolejek dla aplikacji serwera.

Zdefiniuj kanały połączenia klienta tylko w jednym menedżerze kolejek, w tym w tych kanałach, które łączą się z drugim lub trzecim menedżerem kolejek. Nie definiuj ich w dwóch menedżerach kolejek, a następnie spróbuj scalić dwie tabele definicji kanału klienta. Klient może uzyskać dostęp tylko do jednej tabeli definicji kanału klienta.

### Przykłady

Zapoznaj się ponownie z listą przyczyn używania grup menedżerów kolejek na początku tematu. W jaki sposób grupa menedżerów kolejek udostępnia te możliwości?

**Nawiąż połączenie z dowolnym zestawem menedżerów kolejek.**
Zdefiniuj grupę menedżerów kolejek z połączeniami ze wszystkimi menedżerami kolejek w zestawie i nawiąż połączenie z grupą przy użyciu parametru **QmgrName** z przedrostkiem w postaci gwiazdki.

**Ponownie nawiąż połączenie z tym samym menedżerem kolejek, ale nawiąż połączenie z innym, jeśli menedżer kolejek połączony z ostatnim czasem jest niedostępny.**

Zdefiniuj grupę menedżerów kolejek jak wcześniej, ale w każdej definicji kanału klienta ustaw atrybut **AFFINITY** (PREFERRED) .

**Ponów próbę nawiązania połączenia z innym menedżerem kolejek, jeśli nawiązanie połączenia nie powiedzie się.**

Nawiąż połączenie z grupą menedżerów kolejek i ponownie wywołaj wywołanie MQI produktu MQCONN lub MQCONNX , jeśli połączenie zostało zerwane lub jeśli działanie menedżera kolejek nie powiodło się.

**Automatyczne ponowne nawiązywanie połączenia z innym menedżerem kolejek w przypadku niepowodzenia połączenia.**

Nawiąż połączenie z grupą menedżera kolejek przy użyciu opcji MQCNO_RECONNECTproduktu MQCONNX **MQCNO** .

**Automatycznie ponownie nawiąż połączenie z inną instancją menedżera kolejek z wieloma instancjami.**

Należy wykonać te same czynności, co w poprzednim przykładzie. W takim przypadku, aby ograniczyć grupę menedżerów kolejek do łączenia się z instancjami konkretnego menedżera kolejek z wieloma instancjami, należy zdefiniować grupę z połączeniami tylko do instancji menedżera kolejek z wieloma instancjami.

Można również poprosić aplikację kliencką o wywołanie jej wywołania MQI MQCONN lub MQCONNX bez przedrostka gwiazdki w parametrze **QmgrName** . W ten sposób aplikacja kliencka może nawiązać połączenie tylko z określonym menedżerem kolejek. Na koniec można ustawić opcję **MQCNO** na wartość MQCNO_RECONNECT_Q_MGR. Ta opcja akceptuje ponowne połączenia z tym samym menedżerem kolejek, który był wcześniej połączony. Można również użyć tej wartości, aby ograniczyć ponowne połączenia do tej samej instancji normalnego menedżera kolejek.

**Zrównoważenie połączeń klienckich między menedżerami kolejek z większą liczbą klientów połączonych z niektórymi menedżerami kolejek niż z innymi.**

Zdefiniuj grupę menedżerów kolejek i ustaw atrybut **CLNTWGHT** w każdej definicji kanału klienta, aby rozdzielić połączenia nierównomiernie.

**Po awarii połączenia lub menedżera kolejek należy rozłożyć obciążenie związane z ponownym ładowaniem połączenia klienta w sposób nierównomierny i rozłożyć je na czas.**

Należy wykonać te same czynności, co w poprzednim przykładzie. Parametr IBM MQ MQI client losuje liczbę ponownych połączeń między menedżerami kolejek i rozprzestrzenia liczbę ponownych połączeń w czasie.

**Przenieś menedżery kolejek bez zmiany kodu klienta.**

Tabela CCDT izoluje aplikację kliencką od miejsca, w którym znajduje się menedżer kolejek. CCDT to plik danych, który można zdefiniować na kliencie, odczytać z położenia współużytkowanego lub pobrać z serwera WWW. Więcej informacji na ten temat zawiera sekcja Tabela definicji kanału klienta.

**Napisz aplikację kliencką, która nie zna nazw menedżerów kolejek.**

Należy użyć nazw grup menedżerów kolejek i ustanowić konwencję nazewnictwa dla nazw grup menedżerów kolejek, która jest odpowiednia dla aplikacji klienckich w danej organizacji i odzwierciedla architekturę rozwiązań, a nie nazewnictwo menedżerów kolejek.

z/OS *Nawiązywanie połączenia z grupami współużytkowania kolejek*

Aplikację można połączyć z menedżerem kolejek, który jest częścią grupy współużytkowania kolejek. W tym celu można użyć nazwy grupy współużytkowania kolejek zamiast nazwy menedżera kolejek w wywołaniu MQCONN lub MQCONNX.

Nazwy grup współużytkujących kolejki składają się z maksymalnie czterech znaków. Nazwa taka musi być unikalna w danej sieci i nie może być identyczna z nazwą menedżera kolejek.

Definicja kanału klienta powinna używać ogólnego interfejsu grupy współużytkowania kolejek w celu nawiązania połączenia z dostępnym menedżerem kolejek w grupie. Więcej informacji na ten temat zawiera sekcja Nawiązywanie połączenia klienta z grupą współużytkowania kolejek. Wykonywane jest sprawdzenie, czy menedżer kolejek, z którym łączy się program nasłuchujący, jest elementem grupy współużytkowania kolejek.

Więcej informacji na temat kolejek współużytkowanych zawiera sekcja Kolejki współużytkowane i grupy współużytkowania kolejek.

*Przykłady wag i powinowactwa kanałów*
Poniższe przykłady ilustrują sposób wybierania kanałów połączenia klienckiego, gdy używane są niezerowe wagi `ClientChannel` .

Atrybuty kanału `ClientChannelWeight` (Waga kanału klienta) i `ConnectionAffinity` sterują sposobem wybierania kanałów połączenia klienckiego, gdy dla połączenia dostępny jest więcej niż jeden odpowiedni kanał. Te kanały są skonfigurowane do nawiązywania połączeń z różnymi menedżerami kolejek w celu zapewnienia wyższej dostępności, równoważenia obciążenia lub obu tych funkcji. Wywołania MQCONN, które mogą spowodować nawiązanie połączenia z jednym z kilku menedżerów kolejek, muszą poprzedzać nazwę menedżera kolejek gwiazdką zgodnie z opisem w sekcji Przykłady wywołań MQCONN: przykład 1. Nazwa menedżera kolejek zawiera gwiazdkę (*).

Odpowiednie kanały kandydujące dla połączenia to te, w których atrybut QMNAME jest zgodny z nazwą menedżera kolejek określoną w wywołaniu MQCONN. Jeśli wszystkie odpowiednie kanały dla połączenia mają wagę `ClientChannel` równą zero (domyślnie), są one wybierane w kolejności alfabetycznej, tak jak w przykładzie: Przykłady wywołań MQCONN: Przykład 1. Nazwa menedżera kolejek zawiera gwiazdkę (*).

Poniższe przykłady ilustrują, co się dzieje, gdy użyto wartości `ClientChannelWagi` różnej od zera. Należy zauważyć, że ponieważ ta funkcja obejmuje pseudolosowy wybór kanału, przykłady pokazują sekwencję działań, które mogą się zdarzyć, a nie to, co zdecydowanie będzie.

*Przykład 1. Wybieranie kanałów, gdy opcja ConnectionAffinity ma wartość PREFERRED*
Ten przykład ilustruje sposób, w jaki IBM MQ MQI client wybiera kanał z tabeli definicji kanału klienta, w której właściwość ConnectionAffinity jest ustawiona na wartość PREFERRED.

W tym przykładzie pewna liczba komputerów klienckich używa tabeli definicji kanału klienta (CCDT) udostępnianej przez menedżer kolejek. Tabela CCDT zawiera kanały połączenia klienta z następującymi atrybutami (przedstawione przy użyciu składni komendy DEFINE CHANNEL):

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(PREFERRED)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(PREFERRED)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(PREFERRED)
```

Aplikacja wywołuje komendę MQCONN (*CORE)

Kanał A nie jest kandydatem dla tego połączenia, ponieważ atrybut QMNAME nie jest zgodny. Kanały B, C i D są identyfikowane jako kandydaci i są umieszczane w preferowanym porządku na podstawie ich wagi. W tym przykładzie zamówienie może mieć postać C, B, D. Klient próbuje nawiązać połączenie z menedżerem kolejek w pliku core2.ops.company.example. Nazwa menedżera kolejek pod tym adresem nie jest sprawdzana, ponieważ wywołanie MQCONN zawierało gwiazdkę w nazwie menedżera kolejek.

Należy zauważyć, że w systemie `AFFINITY(PREFERRED)` za każdym razem, gdy ten konkretny klient łączy się, kanały będą miały tę samą początkową kolejność preferencji. Ma to zastosowanie nawet wtedy, gdy połączenia są z różnych procesów lub w różnym czasie.

W tym przykładzie menedżer kolejek w core.2.ops.company.example jest nieosiągalny. Klient próbuje połączyć się z core1.ops.company.example , ponieważ kanał B jest następny w kolejności preferencji. Ponadto kanał C jest zdegradowany, aby stać się najmniej preferowanym.

Drugie wywołanie MQCONN (*CORE) jest wykonywane przez tę samą aplikację. Kanał C został zdegradowany przez poprzednie połączenie, więc najbardziej preferowanym kanałem jest teraz B. To połączenie jest nawiązywane z adresem core1.ops.company.example.

Drugi komputer, który współużytkuje tę samą tabelę definicji kanału klienta, może umieścić kanały w innej preferowanej kolejności początkowej. Na przykład D, B, C. W normalnych okolicznościach, gdy działają wszystkie kanały, aplikacje na tym komputerze są połączone z core3.ops.company.example , podczas gdy aplikacje na pierwszym komputerze są połączone z core2.ops.company.example. Umożliwia

to równoważenie obciążenia dużej liczby klientów w wielu menedżerach kolejek przy jednoczesnym umożliwieniu każdemu klientowi połączenia się z tym samym menedżerem kolejek, jeśli jest on dostępny.

*Przykład 2. Wybieranie kanałów, gdy właściwość ConnectionAffinity ma wartość NONE*
W tym przykładzie przedstawiono sposób, w jaki IBM MQ MQI client wybiera kanał z tabeli definicji kanału klienta, w której właściwość ConnectionAffinity jest ustawiona na wartość NONE.

W tym przykładzie wiele klientów używa tabeli definicji kanału klienta (CCDT) udostępnianej przez menedżer kolejek. Tabela CCDT zawiera kanały połączenia klienta z następującymi atrybutami (przedstawione przy użyciu składni komendy DEFINE CHANNEL):

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(NONE)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(NONE)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(NONE)
```

Aplikacja wywołuje komendę MQCONN (*CORE). Podobnie jak w poprzednim przykładzie, kanał A nie jest brany pod uwagę, ponieważ nazwa QMNAME nie jest zgodna. Kanał B, C lub D są wybierane na podstawie ich wagi, z prawdopodobieństwem 50%, 30% lub 20%. W tym przykładzie można wybrać kanał B. Nie utworzono trwałej kolejności preferencji.

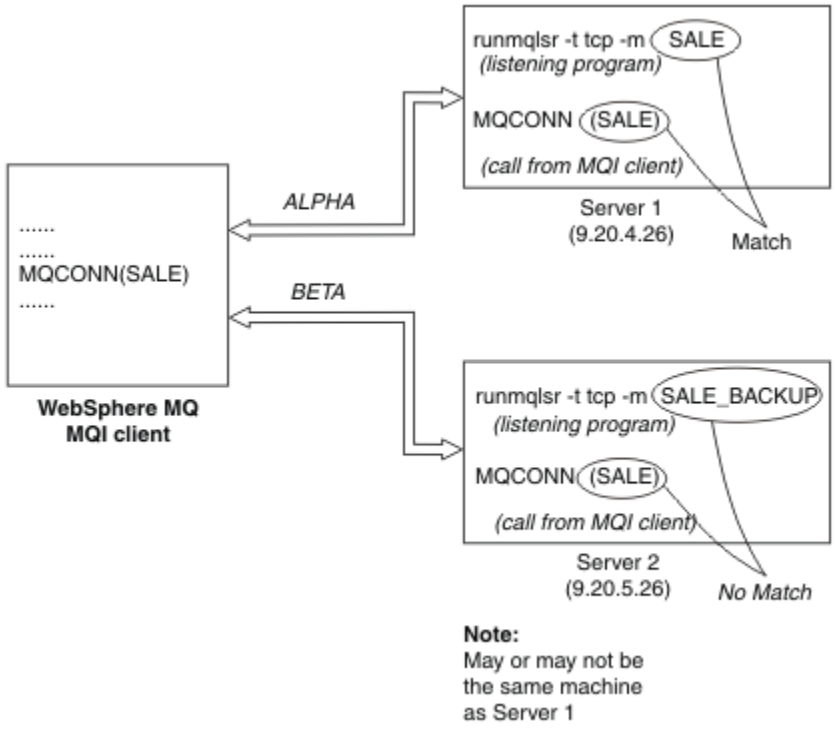
Wykonywane jest drugie wywołanie MQCONN (*CORE). Ponownie wybrano jeden z trzech odpowiednich kanałów z takim samym prawdopodobieństwem. W tym przykładzie wybrano kanał C. Jednak plik core2.ops.company.example nie odpowiada, więc inny wybór jest dokonywany między pozostałymi kanałami kandydackymi. Wybrano kanał B i aplikacja jest połączona z core1.ops.company.example.

W przypadku parametru AFFINITY (NONE) każde wywołanie MQCONN jest niezależne od innych wywołań. Dlatego gdy ta przykładowa aplikacja wykona trzecią komendę MQCONN (*CORE), może ponownie podjąć próbę nawiązania połączenia za pośrednictwem uszkodzonego kanału C przed wybraniem jednej z następujących wartości: B lub D.

*Przykłady wywołań MQCONN*
Przykłady użycia komendy MQCONN do nawiązania połączenia z konkretnym menedżerem kolejek lub z jednym z grup menedżerów kolejek.

W każdym z poniższych przykładów sieć jest taka sama; istnieje połączenie zdefiniowane dla dwóch serwerów z tego samego serwera IBM MQ MQI client. W tych przykładach zamiast wywołania MQCONN można użyć wywołania MQCONNX.

Na komputerach serwera działają dwa menedżery kolejek, jeden o nazwie SALE , a drugi o nazwie SALE_BACKUP.




*Rysunek 98. Przykład wywołania MQCONN*

Definicje kanałów w tych przykładach są następujące:

Definicje SPRZEDAŻY:

```
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)
```

Definicja właściwości SALE_BACKUP:

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')
```

Definicje kanałów klienta można podsumować w następujący sposób:

| Nazwa | CHLTYPE | TRPTYPE | CONNAME | QMNAME |
|---|---|---|---|---|
| ALPHA | CLNTCONN, | TCP | 9.20.4.26 | Sprz. |
| BETA | CLNTCONN, | TCP | 9.20.5.26 | Sprz. |

*Co pokazują przykłady MQCONN*
Przykłady demonstrują użycie wielu menedżerów kolejek jako systemu zapasowego.

Załóżmy, że łącze komunikacyjne z serwerem 1 jest tymczasowo zerwane. Zademonstrowano użycie wielu menedżerów kolejek jako systemu zapasowego.

Każdy przykład obejmuje inne wywołanie MQCONN i zawiera wyjaśnienie tego, co dzieje się w konkretnym przykładzie, stosując następujące reguły:

1. Tabela definicji kanału klienta (CCDT) jest skanowana w kolejności alfabetycznej nazwy kanału dla nazwy menedżera kolejek (pole QMNAME) odpowiadającej nazwie podanej w wywołaniu MQCONN.
2. Jeśli zostanie znalezione dopasowanie, zostanie użyta definicja kanału.
3. Podjęto próbę uruchomienia kanału na komputerze identyfikowanym przez nazwę połączenia (CONNAME). Jeśli operacja powiedzie się, aplikacja będzie kontynuować działanie. Wymaga:
   - Obiekt nasłuchiwania, który ma być uruchomiony na serwerze.
   - Program nasłuchujący, który ma być połączony z tym samym menedżerem kolejek, co program, z którym klient chce się połączyć (jeśli został określony).
4. Jeśli próba uruchomienia kanału nie powiedzie się i w tabeli definicji kanału klienta znajduje się więcej niż jedna pozycja (w tym przykładzie istnieją dwie pozycje), plik jest przeszukiwany pod kątem dalszego dopasowania. Jeśli zostanie znalezione dopasowanie, przetwarzanie będzie kontynuowane od kroku 1.
5. Jeśli nie znaleziono zgodnych pozycji lub nie ma więcej pozycji w tabeli definicji kanału klienta i uruchomienie kanału nie powiodło się, aplikacja nie może nawiązać połączenia. W wywołaniu MQCONN zwracany jest odpowiedni kod przyczyny i kod zakończenia. Aplikacja może podjąć działanie na podstawie zwróconego kodu przyczyny i zakończenia.

*Przykład 1. Nazwa menedżera kolejek zawiera gwiazdkę (*)*
W tym przykładzie aplikacja nie jest zaniepokojona tym, z którym menedżerem kolejek nawiązuje połączenie. Aplikacja wysyła wywołanie MQCONN dla nazwy menedżera kolejek z gwiazdką. Wybierany jest odpowiedni kanał.

Aplikacja wydaje:

```
MQCONN (*SALE)
```

Zgodnie z regułami dzieje się tak w tej instancji:

1. Tabela definicji kanału klienta (CCDT) jest skanowana w poszukiwaniu nazwy menedżera kolejek SALEzgodnej z wywołaniem MQCONN aplikacji.
2. Znaleziono definicje kanałów dla ALPHA i BETA .
3. Jeśli jeden kanał ma wartość CLNTWGHT równą 0, ten kanał jest wybrany. Jeśli oba mają wartość CLNTWGHT równą 0, kanał ALPHA jest wybierany, ponieważ jest on pierwszy w kolejności alfabetycznej. Jeśli oba kanały mają niezerową wartość CLNTWGHT, jeden kanał jest wybierany losowo na podstawie jego wagi.
4. Podjęto próbę uruchomienia kanału.
5. Jeśli kanał BETA został wybrany, próba jego uruchomienia powiodła się.
6. Jeśli wybrano kanał ALPHA , próba jego uruchomienia NIE powiedzie się, ponieważ łącze komunikacyjne jest zerwane. Następnie należy wykonać następujące kroki:
   a. Jedynym innym kanałem dla menedżera kolejek o nazwie SALE jest BETA.
   b. Podjęto próbę uruchomienia tego kanału-to się powiodło.
7. Sprawdzenie, czy program nasłuchujący jest uruchomiony, wskazuje, że jest on uruchomiony. Nie jest on połączony z menedżerem kolejek produktu SALE , ale ponieważ parametr wywołania MQI zawiera znak gwiazdki (*), nie jest wykonywane żadne sprawdzenie. Aplikacja jest połączona z menedżerem kolejek produktu SALE_BACKUP i kontynuuje przetwarzanie.

*Przykład 2. Podana nazwa menedżera kolejek*
W tym przykładzie aplikacja musi nawiązać połączenie z konkretnym menedżerem kolejek. Aplikacja wysyła wywołanie MQCONN dla tej nazwy menedżera kolejek. Wybierany jest odpowiedni kanał.

Aplikacja wymaga połączenia z konkretnym menedżerem kolejek o nazwie SALE, co jest widoczne w wywołaniu MQI:

```
MQCONN (SALE)
```

Zgodnie z regułami dzieje się tak w tej instancji:

1. Tabela definicji kanału klienta (CCDT) jest skanowana w kolejności alfabetycznej nazwy kanału dla nazwy menedżera kolejek SALEzgodnej z wywołaniem MQCONN aplikacji.
2. Pierwsza znaleziona definicja kanału do dopasowania to ALPHA.
3. Próba uruchomienia kanału nie powiodła się, ponieważ łącze komunikacyjne jest zerwane.
4. Tabela definicji kanału klienta jest ponownie skanowana w poszukiwaniu nazwy menedżera kolejek SALE i nazwy kanału BETA .
5. Podjęto próbę uruchomienia kanału-to działanie powiodło się.
6. Sprawdzenie, czy proces nasłuchujący jest uruchomiony, wykazuje, że jest on uruchomiony, ale nie jest połączony z menedżerem kolejek produktu SALE .
7. W tabeli definicji kanału klienta nie ma dalszych pozycji. Aplikacja nie może kontynuować działania i otrzymuje kod powrotu MQRC_Q_MGR_NOT_AVAILABLE.

*Przykład 3. Nazwa menedżera kolejek jest pusta lub zawiera gwiazdkę (*)*
W tym przykładzie aplikacja nie jest zaniepokojona tym, z którym menedżerem kolejek nawiązuje połączenie. Aplikacja wysyła komendę MQCONN, podając pustą nazwę menedżera kolejek lub gwiazdkę. Wybierany jest odpowiedni kanał.

Jest ona traktowana w taki sam sposób, jak “Przykład 1. Nazwa menedżera kolejek zawiera gwiazdkę (*)” na stronie 959.

**Uwaga:** Jeśli ta aplikacja była uruchomiona w środowisku innym niż IBM MQ MQI client, a nazwa była pusta, podejmowana jest próba nawiązania połączenia z domyślnym menedżerem kolejek. Nie ma to miejsca w przypadku uruchamiania z poziomu środowiska klienta. Dostęp do menedżera kolejek jest powiązany z programem nasłuchującym, z którym łączy się kanał.

Aplikacja wydaje:

```
MQCONN ("")
```

lub wersji

```
MQCONN (*)
```

Zgodnie z regułami dzieje się tak w tej instancji:

1. Tabela definicji kanału klienta (CCDT) jest skanowana w kolejności alfabetycznej w poszukiwaniu nazwy menedżera kolejek, która jest pusta i jest zgodna z wywołaniem MQCONN aplikacji.
2. Pozycja dla nazwy kanału ALPHA ma nazwę menedżera kolejek w definicji SALE. Nie jest to zgodne z parametrem wywołania MQCONN, który wymaga, aby nazwa menedżera kolejek była pusta.
3. Następna pozycja dotyczy nazwy kanału BETA.
4. `queue manager name` w definicji to SALE. Po raz kolejny nie jest to zgodne z parametrem wywołania MQCONN, który wymaga, aby nazwa menedżera kolejek była pusta.
5. W tabeli definicji kanału klienta nie ma dalszych pozycji. Aplikacja nie może kontynuować działania i otrzymuje kod powrotu MQRC_Q_MGR_NOT_AVAILABLE.

### *Wyzwalanie w środowisku klienta*

Komunikaty wysyłane przez aplikacje IBM MQ działające w systemie IBM MQ MQI clients mają wpływ na wyzwalanie w taki sam sposób, jak inne komunikaty, i mogą być używane do wyzwalania programów zarówno na serwerze, jak i na kliencie.

Wyzwalanie jest szczegółowo wyjaśnione w sekcji Wyzwalanie kanałów.

Monitor wyzwalacza i aplikacja, która ma zostać uruchomiona, muszą znajdować się w tym samym systemie.

Domyślne parametry wyzwalanej kolejki są takie same jak w środowisku serwera. W szczególności, jeśli w aplikacji klienckiej umieszczającej komunikaty w wyzwalanej kolejce, która jest lokalna względem menedżera kolejek produktu z/OS , nie określono żadnych opcji sterowania punktem synchronizacji MQPMO, komunikaty są umieszczane w jednostce pracy. Jeśli warunek wyzwalania jest spełniony, komunikat wyzwalacza jest umieszczany w kolejce inicjowania w tej samej jednostce pracy i nie może zostać wczytany przez monitor wyzwalacza, dopóki jednostka pracy nie zostanie zakończona. Proces, który ma zostać wyzwolony, nie jest uruchamiany do momentu zakończenia jednostki pracy.

*Definicja procesu*
Definicję procesu należy zdefiniować na serwerze, ponieważ jest ona powiązana z kolejką, dla której ustawiono wyzwalanie.

Obiekt procesu definiuje, co ma być wyzwalane. Jeśli klient i serwer nie działają na tej samej platformie, wszystkie procesy uruchomione przez monitor wyzwalacza muszą definiować *ApplType*, w przeciwnym razie serwer przyjmuje swoje domyślne definicje (tj. typ aplikacji, która jest zwykle powiązana z serwerem) i powoduje awarię.

Jeśli na przykład monitor wyzwalacza działa w systemie IBM MQ MQI client i chce wysłać żądanie do serwera w innym systemie operacyjnym, należy zdefiniować parametr MQAT_WINDOWS_NT, w przeciwnym razie inny system operacyjny użyje swoich domyślnych definicji i proces zakończy się niepowodzeniem.

*monitor wyzwalacza*
Monitor wyzwalacza udostępniany przez produkty inne niżz/OS IBM MQ działa w środowiskach klienckich dla systemów IBM i IBM ii AIX, Linux, and Windows .

Aby uruchomić monitor wyzwalacza, wykonaj jedną z następujących komend:

- IBM i W systemie IBM i:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QmgrName '-q' InitQ)
```

- ALW Na platformach AIX, Linux, and Windows :

```
runmqtmc [-m QMgrName] [-q InitQ]
```

Domyślną kolejką inicjującą jest SYSTEM.DEFAULT.INITIATION.QUEUE w domyślnym menedżerze kolejek. Kolejka inicjująca jest miejscem, w którym monitor wyzwalacza szuka komunikatów wyzwalacza. Następnie wywołuje programy dla odpowiednich komunikatów wyzwalacza. Ten monitor wyzwalacza obsługuje domyślny typ aplikacji i jest taki sam jak `runmqtrm` z tą różnicą, że łączy biblioteki klienta.

Łańcuch komendy zbudowany przez monitor wyzwalacza jest następujący:

1. Wartość *ApplicId* z odpowiedniej definicji procesu. *ApplicId* jest nazwą programu, który ma zostać uruchomiony, tak jak w wierszu komend.
2. Struktura MQTMC2 , ujęta w cudzysłów, uzyskana z kolejki inicjującej. Uruchamiany jest łańcuch komendy, który zawiera ten łańcuch, dokładnie tak, jak został podany, w cudzysłowie, w kolejności, w jakiej komenda systemowa akceptuje go jako jeden parametr.
3. Wartość *EnvrData* z odpowiedniej definicji procesu.

Monitor wyzwalacza nie sprawdza, czy w kolejce inicjującej znajduje się inny komunikat do czasu zakończenia uruchomionej aplikacji. Jeśli aplikacja ma wiele do wykonania, monitor wyzwalacza może nie nadążyć za liczbą nadchodzących komunikatów wyzwalacza. Istnieją dwa sposoby postępowania z tą sytuacją:

1. Uruchom więcej monitorów wyzwalaczy

   Jeśli ma być uruchomionych więcej monitorów wyzwalaczy, można kontrolować maksymalną liczbę aplikacji, które mogą być uruchomione jednocześnie.

2. Uruchamianie uruchomionych aplikacji w tle

   Jeśli aplikacje mają być uruchamiane w tle, program IBM MQ nie nakłada żadnych ograniczeń na liczbę aplikacji, które mogą być uruchamiane.

Aby uruchomić uruchomioną aplikację w tle w systemach AIX and Linux , należy umieścić znak & (ampersand) na końcu znaku *EnvrData* definicji procesu.

*Aplikacje CICS (inne niżz/OS)*
Aplikacja inna niżz/OS CICS , która wywołuje wywołanie MQCONN lub MQCONNX , musi być zdefiniowana w CEDA jako RESIDENT. W przypadku ponownego połączenia aplikacji serwera CICS jako klienta istnieje ryzyko utraty obsługi punktu synchronizacji.

Aplikacja inna niżz/OS CICS , która wywołuje wywołanie MQCONN lub MQCONNX , musi być zdefiniowana w CEDA jako RESIDENT. Aby kod rezydentny był jak najmały, można utworzyć odsyłacz do oddzielnego programu w celu wywołania funkcji MQCONN lub MQCONNX .

Jeśli zmienna środowiskowa MQSERVER jest używana do zdefiniowania połączenia klienta, musi być ona określona w zmiennej środowiskowej CICSENV.CMD .

Aplikacje IBM MQ mogą być uruchamiane w środowisku serwera IBM MQ lub na kliencie IBM MQ bez zmiany kodu. Jednak w środowisku serwera IBM MQ CICS może działać jako koordynator punktu synchronizacji, a użytkownik używa EXEC CICS SYNCPOINT i EXEC CICS SYNCPOINT ROLLBACK zamiast **MQCMIT** i **MQBACK**. Jeśli aplikacja CICS jest po prostu ponownie zliczana jako klient, obsługa punktów synchronizacji zostanie utracona. Wartości **MQCMIT** i **MQBACK** muszą być używane dla aplikacji działającej w systemie IBM MQ MQI client.

## ALW Przygotowywanie i uruchamianie aplikacji CICS i Tuxedo

Aby uruchomić aplikacje CICS i Tuxedo jako aplikacje klienckie, należy użyć innych bibliotek niż te, które są używane z aplikacjami serwera. Identyfikator użytkownika, który jest używany do uruchamiania aplikacji, również jest inny.

Aby przygotować aplikacje CICS i Tuxedo do uruchamiania jako aplikacje IBM MQ MQI client , należy postępować zgodnie z instrukcjami zawartymi w sekcji Konfigurowanie rozszerzonego klienta transakcyjnego.

Należy jednak zauważyć, że informacje dotyczące przygotowania aplikacji CICS i Tuxedo, w tym programów przykładowych dostarczonych z produktem IBM MQ, zakładają, że użytkownik przygotowuje aplikacje do uruchomienia w systemie serwera IBM MQ . W związku z tym informacje odnoszą się tylko do bibliotek IBM MQ , które są przeznaczone do użycia w systemie serwera. Podczas przygotowywania aplikacji klienckich należy wykonać następujące czynności:

- Użyj odpowiedniej biblioteki systemowej klienta dla powiązań języka używanych przez aplikację. Na przykład:
  - Linux AIX W przypadku aplikacji napisanych w języku C w systemie AIX and Linuxnależy użyć biblioteki libmqic zamiast biblioteki libmqm.
  - Windows W systemach Windows należy użyć biblioteki mqic.lib zamiast biblioteki mqm.lib.
- Zamiast bibliotek systemowych serwera przedstawionych w sekcji Tabela 134 na stronie 963 i Tabela 135 na stronie 963należy użyć odpowiednich bibliotek systemowych klienta. Jeśli biblioteka systemowa serwera nie jest wymieniona w tych tabelach, należy użyć tej samej biblioteki w systemie klienta.

| *Tabela 134. Biblioteki systemowe klienta w systemie AIX and Linux* | |
|---|---|
| **Biblioteka dla systemu serwera IBM MQ** | **Równoważna biblioteka do użycia w systemie klienckim IBM MQ** |
| libmqmxa, | libmqcxa, |

| *Tabela 135. Biblioteki systemowe klienta w systemach Windows* | |
|---|---|
| **Biblioteka dla systemu serwera IBM MQ** | **Równoważna biblioteka do użycia w systemie klienckim IBM MQ** |
| mqmxa.lib | mqcxa.lib |
| mqmtux.lib | mqcxa.lib |
| mqmenc.lib | mqcxa.lib |
| mqmcics4.lib | mqccics4.lib |

## Identyfikator użytkownika używany przez aplikację kliencką

Po uruchomieniu aplikacji serwera IBM MQ w systemie CICSzwykle przełącza się ona z użytkownika CICS na identyfikator użytkownika transakcji. Jednak po uruchomieniu aplikacji IBM MQ MQI client w systemie CICSzachowuje ona uprawnienie uprzywilejowane CICS .

## ALW *Przykładowe programy CICS i Tuxedo*

Przykładowe programy CICS i Tuxedo do użytku w systemach AIX, Linux, and Windows .

Tabela 136 na stronie 963 zawiera listę przykładowych programów CICS i Tuxedo, które są dostarczane do użytku w systemach klienckich AIX and Linux . Tabela 137 na stronie 963 zawiera listę równoważnych informacji dla systemów klienckich Windows . Tabele zawierają również listę plików używanych do przygotowywania i uruchamiania programów. Opis programów przykładowych znajduje się w sekcji “Przykład transakcji CICS” na stronie 1112 i “Korzystanie z przykładów TUXEDO w systemie AIX, Linux, and Windows” na stronie 1157.

| *Tabela 136. Przykładowe programy dla systemów klienckich AIX and Linux* | | |
|---|---|---|
| **Opis** | **Źródło** | **Moduł wykonywalny** |
| CICS program | amqscic0.ccs | amqscicc |
| Plik nagłówkowy programu CICS | amqscih0.h | - |
| Program kliencki Tuxedo do umieszczania komunikatów | amqstxpx.c | - |
| Program kliencki Tuxedo do pobierania komunikatów | amqstxgx.c | - |
| Program serwera Tuxedo dla dwóch programów klienckich | amqstxsx.c | - |
| Plik UBBCONFIG dla programów Tuxedo | ubbstxcx.cfg | - |
| Plik tabeli pól dla programów Tuxedo | amqstxvx.flds | - |
| Wyświetl plik opisu dla programów Tuxedo | amqstxvx.v | - |

| *Tabela 137. Przykładowe programy dla systemów klienckich Windows* | | |
|---|---|---|
| **Opis** | **Źródło** | **Moduł wykonywalny** |
| CICSTransakcja | amqscic0.ccs | amqscicc |
| Plik nagłówkowy transakcji CICS | amqscih0.h | - |

| *Tabela 137. Przykładowe programy dla systemów klienckich Windows (kontynuacja)* | | |
|---|---|---|
| **Opis** | **Źródło** | **Moduł wykonywalny** |
| Program kliencki Tuxedo do umieszczania komunikatów | amqstxpx.c | - |
| Program kliencki Tuxedo do pobierania komunikatów | amqstxgx.c | - |
| Program serwera Tuxedo dla dwóch programów klienckich | amqstxsx.c | - |
| Plik UBBCONFIG dla programów Tuxedo | ubbstxcx.cfg | - |
| Plik tabeli pól dla programów Tuxedo | amqstxvx.fld | - |
| Wyświetl plik opisu dla programów Tuxedo | amqstxvx.v | - |
| Plik makefile dla programów Tuxedo | amqstxmc.mak | - |
| Plik ENVFILE dla programów Tuxedo | amqstxen.env | - |

### ALW *Komunikat o błędzie AMQ5203zmodyfikowany dla aplikacji CICS i Tuxedo*

Po uruchomieniu aplikacji CICS lub Tuxedo, które używają rozszerzonego klienta transakcyjnego, mogą zostać wyświetlone standardowe komunikaty diagnostyczne. Jedna z nich została zmodyfikowana do użytku z rozszerzonym klientem transakcyjnym

Komunikaty, które mogą być wyświetlane w plikach dziennika błędów systemu IBM MQ , są opisane w sekcji Komunikaty diagnostyczne: AMQ4000-9999. Komunikat AMQ5203 został zmodyfikowany do użycia z rozszerzonym klientem transakcyjnym. Oto tekst zmodyfikowanej wiadomości:

#### AMQ5203: Wystąpił błąd podczas wywoływania interfejsu XA.

**Objaśnienie**

Numer błędu to & 2, gdzie wartość 1 wskazuje, że podana wartość opcji & 1 była niepoprawna, 2 wskazuje, że w tym samym procesie podjęto próbę użycia bibliotek wielowątkowych i niewielowątkowych, 3 wskazuje, że wystąpił błąd z podaną nazwą menedżera kolejek '& 3', 4 wskazuje, że identyfikator menedżera zasobów & 1 był niepoprawny, 5 wskazuje, że podjęto próbę użycia drugiego menedżera kolejek o nazwie '& 3 ', gdy inny menedżer kolejek był już połączony, wartość 6 wskazuje, że menedżer transakcji został wywołany, gdy aplikacja nie jest połączona z menedżerem kolejek, wartość 7 wskazuje, że wywołanie XA zostało wykonane, gdy inne wywołanie było w toku, wartość 8 wskazuje, że łańcuch xa_info' & 4 'w wywołaniu xa_open zawierał niepoprawną wartość parametru dla nazwy parametru' & 5 ', a wartość 9 wskazuje, że łańcuch xa_info' & 4 'w wywołaniu xa_open nie zawiera wymaganego parametru, nazwy parametru' & 5 '.

**Odpowiedź użytkownika**

Popraw błąd i ponów operację.

## Windows Przygotowywanie i uruchamianie aplikacji produktu Microsoft Transaction Server

Aby przygotować aplikację MTS do działania jako aplikacja IBM MQ MQI client , należy postępować zgodnie z instrukcjami odpowiednimi dla danego środowiska.

Ogólne informacje na temat tworzenia aplikacji Microsoft Transaction Server (MTS) uzyskujących dostęp do zasobów IBM MQ zawiera sekcja dotycząca MTS w Centrum pomocy produktu IBM MQ .

Aby przygotować aplikację MTS do działania jako aplikacja IBM MQ MQI client , wykonaj jedną z następujących czynności dla każdego komponentu aplikacji:

- Jeśli komponent używa powiązań języka C dla interfejsu MQI, należy postępować zgodnie z instrukcjami zawartymi w sekcji “Przygotowywanie programów w języku C w systemie Windows” na stronie 1046 , ale powiązać komponent z biblioteką mqicxa.lib zamiast z biblioteką mqic.lib.

- Jeśli komponent używa klas języka C++ IBM MQ , należy postępować zgodnie z instrukcjami zawartymi w sekcji “Budowanie programów w języku C++ w systemie Windows” na stronie 569 , ale połączyć komponent z biblioteką imqx23vn.lib zamiast z biblioteką imqc23vn.lib.
- Jeśli komponent używa powiązań języka Visual Basic dla interfejsu MQI, należy postępować zgodnie z instrukcjami zawartymi w sekcji “Przygotowywanie programów Visual Basic w systemie Windows” na stronie 1050 , ale podczas definiowania projektu Visual Basic należy wpisać wartość MqType=3 w polu **Argumenty kompilacji warunkowej** .

### Przygotowywanie i uruchamianie aplikacji produktu IBM MQ JMS

Aplikacje IBM MQ JMS można uruchamiać w trybie klienta, z menedżerem transakcji WebSphere Application Server . Mogą zostać wyświetlone pewne komunikaty ostrzegawcze.

Aby przygotować i uruchomić aplikacje IBM MQ JMS w trybie klienta z menedżerem transakcji WebSphere Application Server , należy postępować zgodnie z instrukcjami zawartymi w sekcji “Korzystanie z IBM MQ classes for JMS/Jakarta Messaging” na stronie 84.

Po uruchomieniu aplikacji klienckiej IBM MQ JMS mogą zostać wyświetlone następujące komunikaty ostrzegawcze:

**MQJE080**
Niewystarczająca liczba jednostek licencji-uruchom komendę setmqcap

**MQJE081**
Plik zawierający informacje o jednostce licencji ma niepoprawny format-uruchom komendę setmqcap

**MQJE082**
Nie można znaleźć pliku zawierającego informacje o jednostce licencji-uruchom komendę setmqcap

## Wyjścia użytkownika, wyjścia funkcji API i instalowalne usługi systemu IBM MQ

Ten temat zawiera odsyłacze do informacji o korzystaniu z tych programów i ich tworzeniu.

Wprowadzenie do używania wyjść użytkownika, wyjść funkcji API i instalowalnych usług w celu rozszerzenia narzędzi menedżera kolejek zawiera sekcja Rozszerzanie narzędzi menedżera kolejek.

Informacje na temat zapisywania i kompilowania wyjść i instalowalnych usług zawierają podtematy.

**Pojęcia pokrewne**
Kanał-programy obsługi wyjścia dla kanałów MQI
**Odsyłacze pokrewne**
Odwołanie do wyjścia funkcji API
Informacje uzupełniające o interfejsie usług instalowalnych
IBM i Informacje uzupełniające o interfejsie instalowalnych usług w systemie IBM i

### ALW Zapisywanie wyjść i usług instalowalnych w systemie AIX, Linux, and Windows

Istnieje możliwość pisania i kompilowania wyjść bez łączenia z bibliotekami IBM MQ w systemie AIX, Linux, and Windows.

#### O tym zadaniu

Ta sekcja dotyczy tylko systemów AIX, Linux, and Windows . Szczegółowe informacje na temat zapisywania wyjść i instalowalnych usług dla innych platform zawierają tematy dotyczące konkretnej platformy.

Jeśli produkt IBM MQ jest zainstalowany w położeniu innym niż domyślne, należy zapisać i skompilować wyjścia bez tworzenia dowiązań do bibliotek produktu IBM MQ .

Można pisać i kompilować wyjścia w systemach AIX, Linux, and Windows bez łączenia żadnej z następujących bibliotek IBM MQ :

- mqmzf,
- MQM
- mqmvx,
- mqmvxd,
- Mqic
- mqutl

Istniejące wyjścia, które są połączone z tymi bibliotekami, nadal działają, pod warunkiem, że w systemach AIX and Linux IBM MQ są zainstalowane w położeniu domyślnym.

## Procedura

1. Dołącz plik nagłówkowy cmqec.h .

   Włączenie tego pliku nagłówkowego powoduje automatyczne włączenie plików nagłówkowych cmqc.h, cmqxc.h i cmqzc.h .
2. Zapisz wyjście, aby wywołania MQI i DCI były wykonywane za pośrednictwem struktury MQIEP. Więcej informacji na temat struktury MQIEP zawiera sekcja Struktura MQIEP.
   - Usługi instalowalne
     - Użyj parametru **Hconfig** , aby wskazać wywołanie MQZEP.
     - Przed użyciem parametru **Hconfig** należy sprawdzić, czy pierwsze 4 bajty pliku **Hconfig** są zgodne z wartością **StrucId** struktury MQIEP.
     - Więcej informacji na temat pisania instalowalnych komponentów usług zawiera sekcja MQIEP.
   - Wyjścia funkcji API
     - Użyj parametru **Hconfig** , aby wskazać wywołanie MQXEP.
     - Przed użyciem parametru **Hconfig** należy sprawdzić, czy pierwsze 4 bajty pliku **Hconfig** są zgodne z wartością **StrucId** struktury MQIEP.
     - Więcej informacji na temat zapisywania wyjść funkcji API zawiera sekcja “Zapisywanie wyjść funkcji API” na stronie 983.
   - Wyjścia kanału
     - Parametr **pEntryPoints** struktury MQCXP wskazuje wywołania MQI i DCI.
     - Przed użyciem produktu **pEntryPoints**należy sprawdzić, czy numer wersji MQCXP jest w wersji 8 lub nowszej.
     - Więcej informacji na temat zapisywania wyjść kanałów zawiera sekcja “Zapisywanie programów obsługi wyjścia kanału” na stronie 993.
   - Wyjścia konwersji danych
     - Parametr **pEntryPoints** struktury MQDXP wskazuje wywołania MQI i DCI.
     - Przed użyciem produktu **pEntryPoints**należy sprawdzić, czy numer wersji MQDXP jest w wersji 2 lub nowszej.
     - Do utworzenia kodu konwersji danych korzystającego z parametru **pEntryPoints** można użyć komendy **crtmqcvx** i pliku źródłowego amqsvfc0.c . Patrz “Zapisywanie wyjścia konwersji danych dla IBM MQ for Windows” na stronie 1020 oraz “Zapisywanie wyjścia konwersji danych dla systemów IBM MQ for AIX or Linux” na stronie 1017.
     - Jeśli istnieją wyjścia konwersji danych, które zostały wygenerowane za pomocą komendy **crtmqcvx** , należy ponownie wygenerować wyjście za pomocą zaktualizowanej komendy.
     - Więcej informacji na temat zapisywania wyjść konwersji danych zawiera sekcja “Zapisywanie wyjść konwersji danych” na stronie 1013.

- Wyjścia przed połączeniem
  - Parametr **pEntryPoints** struktury MQNXP wskazuje wywołania MQI i DCI.
  - Przed użyciem produktu **pEntryPoints**należy sprawdzić, czy numer wersji MQNXP jest w wersji 2 lub nowszej.
  - Więcej informacji na temat zapisywania wyjść przed połączeniem zawiera sekcja "Odwoływanie się do definicji połączeń przy użyciu wyjścia przed nawiązaniem połączenia z repozytorium" na stronie 1022.
- Wyjścia publikowania
  - Parametr **pEntryPoints** struktury MQPSXP służy do wskazywania wywołań MQI i DCI.
  - Przed użyciem produktu **pEntryPoints**należy sprawdzić, czy numer wersji MQPSXP jest w wersji 2 lub nowszej.
  - Więcej informacji na temat zapisywania wyjść publikowania zawiera sekcja "Zapisywanie i kompilowanie wyjść publikowania" na stronie 1024.
- Wyjścia obciążenia klastra
  - Parametr **pEntryPoints** struktury MQWXP wskazuje na wywołania MQXCLWLN.
  - Przed użyciem pakietu **pEntryPoints**należy sprawdzić, czy numer wersji MQWXP jest w wersji 4 lub nowszej.
  - Więcej informacji na temat zapisywania wyjść obciążenia klastra zawiera sekcja "Zapisywanie i kompilowanie wyjść obciążenia klastra" na stronie 1026.

Na przykład w wyjściu kanału wywołującym komendę MQPUT:

```
pChannelExitParms -> pEntryPoints -> MQPUT_Call(pChannelExitParms -> Hconn,
                                                Hobj,
                                                &md,
                                                &pmo,
                                                messlen,
                                                buffer,
                                                &CompCode,
                                                &Reason);
```

Więcej przykładów można znaleźć w pliku "Korzystanie z przykładowych programów proceduralnych IBM MQ" na stronie 1089.

3. Skompiluj wyjście:
   - Nie należy tworzyć dowiązań do bibliotek IBM MQ .
   - W wyjściu nie należy dołączać osadzonej ścieżki RPath do żadnych bibliotek IBM MQ .
   - Więcej informacji na temat kompilowania wyjścia zawiera jeden z następujących tematów:
     - Wyjścia funkcji API: "Kompilowanie wyjść funkcji API" na stronie 985.
     - Wyjścia kanału, wyjścia publikowania, wyjścia obciążenia klastra: "Kompilowanie programów obsługi wyjścia kanału w systemach AIX, Linux, and Windows" na stronie 1011.
     - Wyjścia konwersji danych: "Zapisywanie wyjść konwersji danych" na stronie 1013.
4. Umieść wyjście w jednym z następujących miejsc:
   - Ścieżka wybrana przez użytkownika, która jest pełna podczas konfigurowania wyjścia
   - Domyślna ścieżka wyjścia w określonym katalogu instalacyjnym. Na przykład: *MQ_DATA_PATH*/exits/installation2.
   - Domyślna ścieżka wyjścia

     Domyślna ścieżka wyjścia to *MQ_DATA_PATH*/exits dla 32-bitowych wyjść i *MQ_DATA_PATH*/exits64 dla 64-bitowych wyjść. Ścieżki te można zmienić w pliku qm.ini lub mqclient.ini . Więcej informacji na ten temat zawiera sekcja Ścieżka wyjścia. W systemach Windows i Linuxmożna użyć Eksploratora IBM MQ , aby zmienić ścieżkę:

     a. Kliknij prawym przyciskiem myszy nazwę menedżera kolejek.

b. Kliknij opcję **Właściwości ...**

c. Kliknij opcję **Wyjścia** .

d. W polu domyślnej ścieżki wyjścia podaj nazwę ścieżki katalogu, w którym znajduje się program obsługi wyjścia.

Jeśli wyjście jest umieszczone zarówno w określonym katalogu instalacyjnym, jak i w domyślnym katalogu ścieżki, jest ono używane przez instalację produktu IBM MQ o nazwie podanej w ścieżce. Na przykład wyjście jest umieszczane w katalogu `/exits/installation2` oraz w katalogu `/exits`, ale nie w katalogu `/exits/installation1`. IBM MQ Instalacja `installation2` korzysta z wyjścia z programu `/exits/installation2`. IBM MQ Instalacja `installation1` korzysta z wyjścia z katalogu `/exits` .

5. W razie potrzeby skonfiguruj wyjście:
   - Usługi do zainstalowania: “Konfigurowanie usług i komponentów” na stronie 976.
   - Wyjścia funkcji API: “Konfigurowanie wyjść funkcji API” na stronie 988.
   - Wyjścia kanału: “Konfigurowanie wyjść kanałów” na stronie 1012.
   - Wyjścia publikowania: “Konfigurowanie wyjść publikowania” na stronie 1025.
   - Wyjścia przed połączeniem: sekcjaPreConnect w pliku konfiguracyjnym klienta.

### ALW *Wyjścia funkcji API nie są połączone z biblioteką MQI*

W pewnych okolicznościach należy połączyć istniejące wyjście funkcji API, którego nie można ponownie zakodować w celu użycia wskaźników funkcji MQIEP, z biblioteką funkcji API IBM MQ .

Jest to konieczne, aby istniejące wyjście funkcji API mogło zostać pomyślnie załadowane przez konsolidator środowiska wykonawczego systemu do programów, które nie mają jeszcze załadowanych wskaźników funkcji.

**Uwaga:** Informacje te są ograniczone do istniejących wyjść funkcji API, które bezpośrednio wywołują wywołania MQI. Oznacza to, że wyjścia, które nie są używane, MQIEP. Jeśli to możliwe, należy zaplanować ponowne zakodowanie wyjścia w taki sposób, aby używały punktów wejścia MQIEP.

W produkcie IBM MQ 8.0 **runmqsc** jest przykładem programu, który nie łączy się bezpośrednio z biblioteką MQI.

Oznacza to, że nie można załadować do produktu **runmqsc**wyjścia API, które nie zostało połączone z wymaganą biblioteką API IBM MQ lub ponownie zakodowane w celu użycia MQIEP.

W dzienniku błędów menedżera kolejek znajdują się błędy, na przykład AMQ6175: `System nie może dynamicznie załadować biblioteki współużytkowanej`wraz z zakwalifikowanym tekstem, takim jak `undefined symbol: MQCONN`.

i AMQ7214: `Nie można załadować modułu dla wyjścia funkcji API o nazwie myexitname`.

**Zadania pokrewne**

“Zapisywanie wyjść i usług instalowalnych w systemie AIX, Linux, and Windows” na stronie 965
Istnieje możliwość pisania i kompilowania wyjść bez łączenia z bibliotekami IBM MQ w systemie AIX, Linux, and Windows.

## ALW Usługi i komponenty, które można zainstalować w systemie AIX, Linux, and Windows

Ta sekcja zawiera wprowadzenie do instalowalnych usług oraz powiązanych z nimi funkcji i komponentów. Interfejs do tych funkcji jest udokumentowany, aby użytkownik lub dostawcy oprogramowania mógł dostarczać komponenty.

Główne powody udostępniania usług instalowalnych dla systemu IBM MQ są następujące:

- W celu zapewnienia elastyczności pozwalającej wybrać, czy mają być używane komponenty udostępniane przez produkty IBM MQ , czy też należy je zastąpić lub rozszerzyć o inne osoby.

- Aby umożliwić dostawcom uczestnictwo, udostępniając komponenty, które mogą korzystać z nowych technologii, bez wprowadzania wewnętrznych zmian w produktach IBM MQ .
- Aby umożliwić IBM MQ wykorzystanie nowych technologii szybciej i taniej, a tym samym dostarczać produkty wcześniej i po niższych cenach.

*Usługi instalowalne* i *komponenty usług* są częścią struktury produktu IBM MQ . W centrum tej struktury znajduje się część menedżera kolejek, która implementuje funkcję i reguły powiązane z interfejsem kolejki komunikatów (Message Queue Interface-MQI). Ta centralna część wymaga szeregu funkcji serwisowych, nazywanych *usługami instalowalnymi*, do wykonania swojej pracy. Możliwe do zainstalowania usługi to:

- Usługa autoryzacji
- usługa nazw

Każda instalowalna usługa jest pokrewnym zestawem funkcji zaimplementowanych przy użyciu jednego lub większej liczby *komponentów usług*. Każdy komponent jest wywoływany przy użyciu poprawnie zbudowanego, publicznie dostępnego interfejsu. Dzięki temu niezależni dostawcy oprogramowania i inne firmy mogą udostępniać instalowalne komponenty w celu rozszerzenia lub zastąpienia komponentów dostarczanych przez produkty IBM MQ . Tabela 138 na stronie 969 zawiera podsumowanie usług i komponentów, które mogą być używane.

*Tabela 138. Podsumowanie instalowalnych komponentów usług*

| usługa instalowalna | Dostarczony komponent | Funkcja | Wymagania |
|---|---|---|---|
| Usługa autoryzacji | menedżer uprawnień obiektu (OAM) | Udostępnia sprawdzanie autoryzacji dla komend i wywołań MQI. Użytkownicy mogą napisać własny komponent w celu rozszerzenia lub zastąpienia OAM.<br>Na przykład, aby sprawdzić, czy ID użytkownika ma uprawnienia do otwarcia kolejki. | (Przyjmuje się odpowiednie narzędzia autoryzacji platformy) |
| usługa nazw | Brak | Udostępnia menedżerowi kolejek obsługę wyszukiwania nazwy menedżera kolejek, który jest właścicielem określonej kolejki.<br>• Zdefiniowany przez użytkownika | • Menedżer nazw innej firmy lub napisany przez użytkownika |

Interfejs usług instalowalnych jest opisany w sekcji Informacje uzupełniające o interfejsie usług instalowalnych.

**Zadania pokrewne**

Konfigurowanie instalowalnych usług

## *Pisanie komponentu usługi*

W tej sekcji opisano relacje między usługami, komponentami, punktami wejścia i kodami powrotu.

## Funkcje i komponenty

Każda usługa składa się z zestawu powiązanych funkcji. Na przykład usługa nazw zawiera funkcję dla:

- Wyszukiwanie nazwy kolejki i zwracanie nazwy menedżera kolejek, w którym zdefiniowano kolejkę

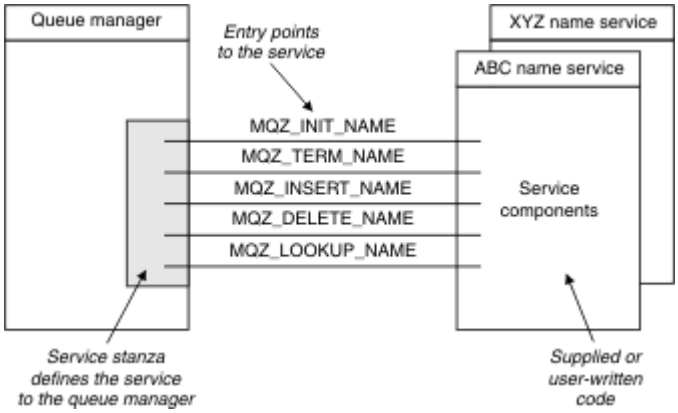
- Wstawianie nazwy kolejki do katalogu usługi
- Usuwanie nazwy kolejki z katalogu usługi

Zawiera również funkcje inicjowania i zakończenia.

Instalowalna usługa jest udostępniana przez co najmniej jeden komponent usługi. Każdy komponent może wykonywać niektóre lub wszystkie funkcje zdefiniowane dla tej usługi. Na przykład w systemie IBM MQ for AIXdostarczony komponent usługi autoryzacji, OAM, wykonuje wszystkie dostępne funkcje. Więcej informacji zawiera sekcja “Interfejs usługi autoryzacji” na stronie 973. Komponent jest również odpowiedzialny za zarządzanie zasobami bazowymi lub oprogramowaniem (na przykład katalogiem LDAP), które jest potrzebne do zaimplementowania usługi. Pliki konfiguracyjne udostępniają standardowy sposób ładowania komponentu i określania adresów udostępnianych przez niego procedur funkcjonalnych.

Rysunek 99 na stronie 970 przedstawia sposób, w jaki usługi i komponenty są powiązane:

- Usługa jest definiowana w menedżerze kolejek za pomocą sekcji w pliku konfiguracyjnym.
- Każda usługa jest obsługiwana przez kod dostarczony w menedżerze kolejek. Użytkownicy nie mogą zmieniać tego kodu i dlatego nie mogą tworzyć własnych usług.
- Każda usługa jest implementowana przez jeden lub więcej komponentów. Można je dostarczyć wraz z produktem lub napisem dla użytkownika. Można wywołać wiele komponentów dla usługi, z których każdy obsługuje różne narzędzia w ramach usługi.
- Punkty wejścia łączą komponenty usługi z kodem obsługi w menedżerze kolejek.




*Rysunek 99. Podstawowe informacje o usługach, komponentach i punktach wejścia*

## Punkty wejścia

Każdy komponent usługi jest reprezentowany przez listę adresów punktów wejścia procedur, które obsługują konkretną usługę instalowalną. Instalowalna usługa definiuje funkcję, która ma być wykonywana przez każdą procedurę.

Kolejność komponentów usługi podczas ich konfigurowania definiuje kolejność, w jakiej punkty wejścia są wywoływane w celu spełnienia żądania usługi.

W dostarczonym pliku nagłówkowym `cmqzc.h`podane punkty wejścia do każdej usługi mają przedrostek MQZID_.

Jeśli usługi są obecne, są one ładowane w predefiniowanej kolejności. Poniższa lista przedstawia usługi i kolejność ich inicjowania.

1. `NameService`
2. `AuthorizationService`
3. `UserIdentifierService`

`AuthorizationService` jest jedyną usługą, która jest domyślnie skonfigurowana. Jeśli mają być używane `NameService` i `UserIdentifierService` , należy je skonfigurować ręcznie.

Usługi i komponenty usług mają odwzorowanie jeden do jednego lub jeden do wielu. Dla każdej usługi można zdefiniować wiele komponentów usług. W systemach AIX and Linux wartość Service w sekcji `ServiceComponent` musi być zgodna z wartością Name w sekcji Service w pliku qm.ini . W systemie Windowswartość klucza rejestru usług `ServiceComponent` musi być zgodna z wartością klucza rejestru nazw i jest zdefiniowana jako: `HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere MQ\Installation\`*`MQ_INSTALLATION_NAME`*`\Configuration\QueueManager\`*`qmname`*`\` , gdzie *qmname* jest nazwą menedżera kolejek.

W systemach AIX and Linux komponenty usług są uruchamiane w kolejności, w jakiej zostały zdefiniowane w pliku qm.ini . W systemie Windows, ponieważ używany jest rejestr Windows , IBM MQ wywołuje funkcję **RegEnumKey** , która zwraca wartości w porządku alfabetycznym. Dlatego w systemie Windows usługi są wywoływane w porządku alfabetycznym, zgodnie z ich definicją w rejestrze.

Kolejność definicji `ServiceComponent` jest istotna. Ta kolejność określa kolejność uruchamiania komponentów dla danej usługi. Na przykład `AuthorizationService` w systemie Windows jest skonfigurowany z domyślnym komponentem OAM o nazwie `MQSeries.WindowsNT.auth.service`. Dla tej usługi można zdefiniować dodatkowe komponenty w celu nadpisania domyślnego OAM. Jeśli nie określono parametru `MQCACF_SERVICE_COMPONENT` , do przetworzenia żądania używany jest pierwszy napotkany komponent w porządku alfabetycznym, a następnie używana jest nazwa tego komponentu.

## Kody powrotu

Komponenty usług udostępniają menedżerowi kolejek kody powrotu w celu raportowania różnych warunków. Zgłaszają one powodzenie lub niepowodzenie operacji i wskazują, czy menedżer kolejek ma przejść do następnego komponentu usługi. Do wskazania tego służy osobny parametr *Continuation* .

## Dane komponentu

Pojedynczy komponent usługi może wymagać współużytkowania danych między różnymi funkcjami. Usługi instalowalne udostępniają opcjonalny obszar danych, który ma być przekazywany przy każdym wywołaniu komponentu usługi. Ten obszar danych jest przeznaczony do wyłącznego użytku komponentu usługi. Jest ona współużytkowana przez wszystkie wywołania konkretnej funkcji, nawet jeśli są one wykonywane z różnych przestrzeni adresowych lub procesów. Gwarantowana jest możliwość adresowania z komponentu usługi za każdym razem, gdy jest on wywoływany. Należy zadeklarować wielkość tego obszaru w sekcji *`ServiceComponent`* .

*Inicjowanie i kończenie komponentów*
Użycie opcji inicjowania i kończenia komponentu.

Po wywołaniu procedury inicjowania komponentu musi ona wywołać funkcję **MQZEP** menedżera kolejek dla każdego punktu wejścia obsługiwanego przez komponent. **MQZEP** definiuje punkt wejścia do usługi. Przyjmuje się, że wszystkie niezdefiniowane punkty wyjścia mają wartość NULL.

Komponent jest zawsze wywoływany raz z podstawową opcją inicjowania, zanim zostanie wywołany w inny sposób.

Komponent można wywołać z dodatkową opcją inicjowania na niektórych platformach. Można go na przykład wywołać raz dla każdego procesu, wątku lub zadania systemu operacyjnego, za pomocą którego uzyskiwany jest dostęp do usługi.

Jeśli używane jest inicjowanie dodatkowe:

- Komponent można wywołać więcej niż jeden raz w celu zainicjowania dodatkowego. Dla każdego takiego wywołania, gdy usługa nie jest już potrzebna, jest wysyłane zgodne wywołanie dla dodatkowego zakończenia.

  W przypadku usług nazewnictwa jest to wywołanie MQZ_TERM_NAME.

  W przypadku usług autoryzacji jest to wywołanie MQZ_TERM_AUTHORITY.
- Punkty wejścia muszą być ponownie określone (przez wywołanie MQZEP) za każdym razem, gdy komponent jest wywoływany w celu zainicjowania podstawowego i dodatkowego.

- Dla komponentu używana jest tylko jedna kopia danych komponentu; nie ma innej kopii dla każdego dodatkowego inicjowania.
- Komponent nie jest wywoływany dla żadnych innych wywołań usługi (z procesu, wątku lub czynności systemu operacyjnego, w zależności od przypadku) przed wykonaniem dodatkowego inicjowania.
- Komponent musi ustawić parametr **Version** na taką samą wartość dla inicjowania podstawowego i dodatkowego.

Komponent jest zawsze wywoływany z podstawową opcją zakończenia, gdy nie jest już wymagany. Do tego komponentu nie są wykonywane żadne dalsze wywołania.

Komponent jest wywoływany z opcją dodatkowego zakończenia, jeśli został wywołany w celu dodatkowego zainicjowania.

*menedżer uprawnień obiektu (OAM)*
Komponent usługi autoryzacji dostarczany z produktami IBM MQ nosi nazwę Object Authority Manager (OAM).

Domyślnie moduł OAM jest aktywny i działa z komendami sterującymi **dspmqaut** (uprawnienie do wyświetlania), **dmpmqaut** (uprawnienie do zrzutu) i **setmqaut** (uprawnienie do ustawiania lub resetowania).

Składnia tych komend i sposób ich użycia opisano w sekcji Administrowanie produktem IBM MQ for Multiplatforms za pomocą komend sterujących.

OAM współpracuje z *jednostką* nazwy użytkownika lub grupy:

- Linux AIX W systemach AIX and Linux jednostka główna jest identyfikatorem użytkownika lub identyfikatorem powiązanym z aplikacją działającą w imieniu użytkownika; grupa jest zdefiniowaną w systemie kolekcją jednostek głównych.
- Windows W systemach Windows jednostka główna to identyfikator użytkownika systemu Windows lub identyfikator powiązany z aplikacją działającą w imieniu użytkownika, a grupa to grupa systemu Windows .

Autoryzacje mogą być nadawane lub odbierane na poziomie użytkownika lub grupy.

Po wykonaniu żądania MQI lub wykonaniu komendy funkcja OAM sprawdza, czy jednostka powiązana z operacją ma uprawnienia do wykonania żądanej operacji i uzyskania dostępu do określonych zasobów menedżera kolejek.

Usługa autoryzacji umożliwia rozszerzenie lub zastąpienie sprawdzania uprawnień udostępnionego dla menedżerów kolejek przez napisanie własnego komponentu usługi autoryzacji.

*usługa nazw*
Usługa nazw jest instalowalną usługą zapewniającą obsługę menedżera kolejek w celu wyszukiwania nazwy menedżera kolejek, do którego należy określona kolejka. Z usługi nazw nie można pobrać żadnych innych atrybutów kolejki.

Usługa nazw umożliwia aplikacji otwieranie kolejek zdalnych dla danych wyjściowych tak, jakby były kolejkami lokalnymi. Usługa nazw nie jest wywoływana dla obiektów innych niż kolejki.

**Uwaga:** Dla kolejek zdalnych atrybut **Scope** musi być ustawiony na wartość CELL.

Gdy aplikacja otwiera kolejkę, najpierw szuka nazwy kolejki w katalogu menedżera kolejek. Jeśli go tam nie znajdzie, przeszukuje tyle usług nazw, ile zostało skonfigurowanych, dopóki nie znajdzie takiej, która rozpoznaje nazwę kolejki. Jeśli żaden nie rozpozna nazwy, otwarcie nie powiedzie się.

Usługa nazw zwraca menedżera kolejek będącego właścicielem tej kolejki. Następnie menedżer kolejek kontynuuje żądanie MQOPEN, tak jakby komenda określała nazwę kolejki i menedżera kolejek w oryginalnym żądaniu.

Interfejs usługi nazw (Name Service Interface-NSI) jest częścią środowiska IBM MQ .

## Sposób działania usługi nazw

Jeśli definicja kolejki określa atrybut **Scope** jako menedżer kolejek, to znaczy SCOPE (QMGR) w programie MQSC, definicja kolejki (wraz ze wszystkimi atrybutami kolejki) jest przechowywana tylko w katalogu menedżera kolejek. Nie można go zastąpić instalowalną usługą.

Jeśli definicja kolejki określa atrybut **Scope** jako komórkę, czyli SCOPE (CELL) w programie MQSC, definicja kolejki zostanie ponownie zapisana w katalogu menedżera kolejek wraz ze wszystkimi atrybutami kolejki. Jednak nazwa kolejki i nazwa menedżera kolejek są również przechowywane w usłudze nazw. Jeśli nie jest dostępna żadna usługa, która może przechowywać te informacje, nie można zdefiniować kolejki z komórką *Scope* .

Katalog, w którym są przechowywane informacje, może być zarządzany przez usługę lub usługa może używać w tym celu usługi bazowej, na przykład katalogu LDAP. W obu przypadkach definicje przechowywane w katalogu muszą być trwałe, nawet po zakończeniu działania komponentu i menedżera kolejek, dopóki nie zostaną jawnie usunięte.

**Uwaga:**

1. Aby wysłać komunikat do definicji kolejki lokalnej hosta zdalnego (z zasięgiem CELL) w innym menedżerze kolejek w obrębie komórki katalogu nazewnictwa, należy zdefiniować kanał.
2. Nie można pobrać komunikatów bezpośrednio z kolejki zdalnej, nawet jeśli ma zasięg CELL.
3. Podczas wysyłania do kolejki o zasięgu CELL nie jest wymagana definicja kolejki zdalnej.
4. Usługa nazewnictwa centralnie definiuje kolejkę docelową, mimo że nadal potrzebna jest kolejka transmisji do menedżera kolejek docelowych i para definicji kanałów. Ponadto kolejka transmisji w systemie lokalnym musi mieć taką samą nazwę jak menedżer kolejek, do którego należy kolejka docelowa, z zasięgiem komórki, w systemie zdalnym.

   Na przykład, jeśli zdalny menedżer kolejek ma nazwę QM01, kolejka transmisji w systemie lokalnym musi również mieć nazwę QM01.

*Interfejs usługi autoryzacji*
Usługa autoryzacji udostępnia punkty wejścia do użycia przez menedżer kolejek.

Punkty wejścia są następujące:

**MQZ_AUTHENTICATE_USER**
Uwierzytelnia ID użytkownika i hasło oraz może ustawić pola kontekstu tożsamości.

**MQZ_CHECK_AUTHORITY (uprawnienie MQZ_CHECK_)**
Sprawdza, czy jednostka ma uprawnienia do wykonywania jednej lub kilku operacji na określonym obiekcie.

**MQZ_CHECK_PRIVILEGED**
Sprawdza, czy podany użytkownik jest użytkownikiem uprzywilejowanym.

**MQZ_COPY_ALL_AUTHORITY,**
Kopiuje wszystkie bieżące autoryzacje istniejące dla obiektu odniesienia do innego obiektu.

**MQZ_DELETE_AUTHORITY, UPRAWNIENIE**
Usuwa wszystkie autoryzacje powiązane z określonym obiektem.

**MQZ_ENUMERATE_AUTHORITY_DATA**
Pobiera wszystkie dane uprawnień, które są zgodne z podanymi kryteriami wyboru.

**MQZ_FREE_USER**
Zwalnia powiązane przydzielone zasoby.

**MQZ_GET_AUTHORITY (uprawnienie GET_AUTHORITY)**
Pobiera uprawnienia, które jednostka ma, aby uzyskać dostęp do określonego obiektu.

**MQZ_GET_EXPLICIT_AUTHORITY**
Pobiera albo uprawnienie, które grupa nazwana ma, aby uzyskać dostęp do określonego obiektu (ale bez dodatkowych uprawnień grupy **nobody** (nikt)), albo uprawnienie, które grupa podstawowa nazwanej jednostki głównej ma, aby uzyskać dostęp do określonego obiektu.

**MQZ_INIT_AUTHORITY, UPRAWNIENIE**
Inicjuje komponent usługi autoryzacji.

**MQZ_INQUIRE**
Wysyła zapytanie o obsługiwaną funkcjonalność usługi autoryzacji.

**MQZ_REFRESH_CACHE**
Odśwież wszystkie autoryzacje.

**MQZ_SET_AUTHORITY (uprawnienia MQZ_SET_)**
Ustawia uprawnienie jednostki do określonego obiektu.

**MQZ_TERM_AUTHORITY (uprawnienia MQZ_TERM)**
Kończy działanie komponentu usługi autoryzacji.

Ponadto w systemie IBM MQ for Windowsusługa autoryzacji udostępnia następujące punkty wejścia do użycia przez menedżer kolejek:

- **MQZ_CHECK_AUTHORITY_2**
- **MQZ_GET_AUTHORITY_2**
- **MQZ_GET_EXPLICIT_AUTHORITY_2**
- **MQZ_SET_AUTHORITY_2**

Te punkty wejścia obsługują użycie identyfikatora bezpieczeństwa systemu Windows (NT SID).

Nazwy te są definiowane jako **typedef** w pliku nagłówkowym `cmqzc.h`, który może być używany do prototypowania funkcji komponentu.

Funkcja inicjowania ( **MQZ_INIT_AUTHORITY** ) musi być głównym punktem wejścia dla komponentu. Inne funkcje są wywoływane przez adres punktu wejścia, który funkcja inicjowania dodała do wektora punktu wejścia komponentu.

*Interfejs usługi nazw*
Usługa nazw udostępnia punkty wejścia do użycia przez menedżer kolejek.

Dostępne są następujące punkty wejścia:

**MQZ_INIT_NAME,**
Zainicjuj komponent usługi nazw.

**MQZ_TERM_NAME,**
Zakończ działanie komponentu usługi nazw.

**MQZ_LOOKUP_NAZWA**
Wyszukaj nazwę menedżera kolejek dla określonej kolejki.

**MQZ_INSERT_NAME**
Wstaw pozycję zawierającą nazwę menedżera kolejek będącego właścicielem określonej kolejki do katalogu używanego przez usługę.

**MQZ_DELETE_NAME (Nazwa tabeli)**
Usuń pozycję dla określonej kolejki z katalogu używanego przez usługę.

Jeśli skonfigurowano więcej niż jedną usługę nazw:

- W przypadku wyszukiwania funkcja MQZ_LOOKUP_NAME jest wywoływana dla każdej usługi na liście do czasu rozstrzygnięcia nazwy kolejki (chyba że dowolny komponent wskazuje, że wyszukiwanie powinno zostać zatrzymane).
- W przypadku wstawiania funkcja MQZ_INSERT_NAME jest wywoływana dla pierwszej usługi na liście, która obsługuje tę funkcję.
- W przypadku usuwania funkcja MQZ_DELETE_NAME jest wywoływana dla pierwszej usługi na liście, która obsługuje tę funkcję.

Nie może zawierać więcej niż jednego komponentu, który obsługuje funkcje wstawiania i usuwania. Jednak komponent, który obsługuje tylko wyszukiwanie, jest wykonalny i może być używany na przykład

jako ostatni komponent na liście w celu rozstrzygnięcia dowolnej nazwy, która nie jest znana żadnemu innemu komponentowi usługi nazw, do menedżera kolejek, w którym można zdefiniować nazwę.

W języku programowania C nazwy są definiowane jako typy danych funkcji przy użyciu instrukcji typedef. Można ich użyć do tworzenia prototypów funkcji usługi, aby upewnić się, że parametry są poprawne.

Plik nagłówkowy zawierający wszystkie materiały specyficzne dla usług instalowalnych to `cmqzc.h` dla języka C.

Oprócz funkcji inicjowania (MQZ_INIT_NAME), która musi być głównym punktem wejścia komponentu, funkcje są wywoływane przez adres punktu wejścia dodany przez funkcję inicjowania przy użyciu wywołania MQZEP.

*Korzystanie z wielu komponentów usług*
Dla usługi można zainstalować więcej niż jeden komponent. Dzięki temu komponenty mogą udostępniać tylko częściowe implementacje usługi, a pozostałe funkcje mogą być oparte na innych komponentach.

## Przykład korzystania z wielu komponentów

Załóżmy, że utworzono dwa komponenty usług nazw o nazwach `ABC_name_serv` i `XYZ_name_serv`.

**`ABC_name_serv`**
Ten komponent obsługuje wstawianie lub usuwanie nazwy z katalogu usług, ale nie obsługuje wyszukiwania nazwy kolejki.

**`XYZ_name_serv`**
Ten komponent obsługuje wyszukiwanie nazwy kolejki, ale nie obsługuje wstawiania nazwy do katalogu usług ani usuwania nazwy z tego katalogu.

Komponent `ABC_name_serv` przechowuje bazę danych nazw kolejek i używa dwóch prostych algorytmów do wstawiania lub usuwania nazwy z katalogu usług.

Komponent `XYZ_name_serv` używa prostego algorytmu, który zwraca stałą nazwę menedżera kolejek dla każdej nazwy kolejki, z którą jest wywoływany. Nie zawiera on bazy danych nazw kolejek i dlatego nie obsługuje funkcji wstawiania i usuwania.

Komponenty są instalowane w tym samym menedżerze kolejek. Sekcje *`ServiceComponent`* są porządkowane w taki sposób, aby komponent `ABC_name_serv` był wywoływany jako pierwszy. Wszystkie wywołania wstawiania lub usuwania kolejki w katalogu komponentu są obsługiwane przez komponent `ABC_name_serv` ; Jest to jedyna funkcja, która implementuje te funkcje. Jednak wywołanie wyszukiwania, którego komponent `ABC_name_serv` nie może rozstrzygnąć, jest przekazywane do komponentu tylko wyszukiwania `XYZ_name_serv`. Ten komponent udostępnia nazwę menedżera kolejek z prostego algorytmu.

## Pomijanie punktów wejścia podczas używania wielu komponentów

Jeśli do udostępniania usługi ma być używanych wiele komponentów, można zaprojektować komponent usługi, który nie implementuje niektórych funkcji. Struktura instalowalnych usług nie nakłada żadnych ograniczeń, które można pominąć. Jednak w przypadku konkretnych instalowalnych usług pominięcie jednej lub większej liczby funkcji może być logicznie niespójne z przeznaczeniem usługi.

## Przykład punktów wejścia używanych z wieloma komponentami

Tabela 139 na stronie 976 przedstawia przykład instalowalnej usługi nazw, dla której zainstalowano dwa komponenty. Każdy z nich obsługuje inny zestaw funkcji powiązanych z tą konkretną instalowalną usługą. W przypadku funkcji wstawiania najpierw wywoływany jest punkt wejścia komponentu ABC. Punkty wejścia, które nie zostały zdefiniowane dla usługi (przy użyciu **MQZEP** ) przyjmuje się, że mają wartość NULL. W tabeli znajduje się punkt wejścia dla inicjowania, ale nie jest to wymagane, ponieważ inicjowanie jest wykonywane przez główny punkt wejścia komponentu.

Jeśli menedżer kolejek musi używać instalowalnej usługi, używa punktów wejścia zdefiniowanych dla tej usługi (kolumny w tabeli Tabela 139 na stronie 976 ). Po kolei dla każdego komponentu menedżer kolejek określa adres procedury, która implementuje wymaganą funkcję. Następnie wywołuje procedurę,

jeśli istnieje. Jeśli operacja powiedzie się, wszystkie wyniki i informacje o statusie są używane przez menedżer kolejek.

*Tabela 139. Przykład punktów wejścia dla instalowalnej usługi*

| Numer funkcji | Komponent usługi nazw ABC | Komponent usługi nazw XYZ |
|---|---|---|
| MQZID_INIT_NAME (inicjowanie) | Funkcja ABC_initialize () | XYZ_initialize () |
| MQZID_TERM_NAME (zakończenie) | Funkcja ABC_terminate () | XYZ_terminate () |
| MQZID_INSERT_NAME (Wstawianie) | Funkcja ABC_Insert () | NULL |
| MQZID_DELETE_NAME (Usuń) | Funkcja ABC_Delete () | NULL |
| MQZID_LOOKUP_NAME (wyszukiwanie) | NULL | XYZ_Lookup () |

Jeśli procedura nie istnieje, menedżer kolejek powtarza ten proces dla następnego komponentu na liście. Ponadto, jeśli procedura istnieje, ale zwraca kod wskazujący, że nie może wykonać operacji, próba jest kontynuowana od następnego dostępnego komponentu. Procedury w komponentach usług mogą zwracać kod, który wskazuje, że nie należy podejmować dalszych prób wykonania operacji.

### *Konfigurowanie usług i komponentów*

Komponenty usług konfiguruje się za pomocą plików konfiguracyjnych menedżera kolejek, z wyjątkiem systemów Windows , w których każdy menedżer kolejek ma własną sekcję w rejestrze.

## Procedura

1. Dodaj sekcje do pliku konfiguracyjnego menedżera kolejek `qm.ini`, aby zdefiniować usługę dla menedżera kolejek i określić położenie modułu:
   - Każda używana usługa musi mieć sekcję `Service` , która definiuje usługę dla menedżera kolejek. Więcej informacji na ten temat zawiera sekcja Service pliku qm.ini.
   - Dla każdego komponentu w usłudze musi istnieć sekcja `ServiceComponent` . Ta sekcja identyfikuje nazwę i ścieżkę modułu zawierającego kod dla tego komponentu. Więcej informacji na ten temat zawiera sekcja ServiceComponent w pliku qm.ini.

   Wraz z produktem dostarczany jest komponent usługi autoryzacji, zwany menedżerem uprawnień do obiektów (Object Authority Manager-OAM). Podczas tworzenia menedżera kolejek plik konfiguracyjny menedżera kolejek (lub rejestr w systemach Windows ) jest automatycznie aktualizowany w celu uwzględnienia odpowiednich sekcji usługi autoryzacji i komponentu domyślnego (OAM). W przypadku innych komponentów należy ręcznie skonfigurować plik konfiguracyjny menedżera kolejek.

   Kod dla każdego komponentu usługi jest ładowany do menedżera kolejek podczas uruchamiania menedżera kolejek przy użyciu powiązania dynamicznego, jeśli jest to obsługiwane na platformie.
2. Zatrzymaj i zrestartuj menedżer kolejek, aby aktywować komponent.

**Odsyłacze pokrewne**

Sekcja service pliku qm.ini

Sekcja ServiceComponent w pliku qm.ini

*Odświeżanie OAM po zmianie autoryzacji użytkownika*

W programie IBM MQmożna odświeżyć informacje o grupie autoryzacji OAM natychmiast po zmianie przypisania do grupy autoryzacji użytkownika, odzwierciedlając zmiany wprowadzone na poziomie systemu operacyjnego, bez konieczności zatrzymywania i restartowania menedżera kolejek. W tym celu należy wydać komendę **REFRESH SECURITY** .

**Uwaga:** Po zmianie autoryzacji za pomocą komendy `setmqaut` OAM natychmiast implementuje takie zmiany.

Menedżery kolejek przechowują dane autoryzacji w kolejce lokalnej o nazwie SYSTEM.AUTH.DATA.QUEUE. Te dane są zarządzane przez produkt **amqzfuma.exe**.

**Odsyłacze pokrewne**

REFRESH SECURITY

## IBM i Instalowalne usługi i komponenty w systemie IBM i

Informacje o instalowalnych usługach oraz powiązanych z nimi funkcjach i komponentach. Interfejs do tych funkcji jest udokumentowany, aby użytkownik lub dostawcy oprogramowania mógł dostarczać komponenty.

Główne powody udostępniania usług instalowalnych dla systemu IBM MQ są następujące:

- W celu zapewnienia elastyczności pozwalającej wybrać, czy mają być używane komponenty udostępniane przez produkt IBM MQ for IBM i, czy też zastąpić je lub rozszerzyć o inne osoby.
- Aby umożliwić dostawcom uczestnictwo, udostępniając komponenty, które mogą korzystać z nowych technologii, bez wprowadzania zmian wewnętrznych w produkcie IBM MQ for IBM i.
- Aby umożliwić IBM MQ wykorzystanie nowych technologii szybciej i taniej, a tym samym dostarczać produkty wcześniej i po niższych cenach.

*Usługi instalowalne* i *komponenty usług* są częścią struktury produktu IBM MQ . W centrum tej struktury znajduje się część menedżera kolejek, która implementuje funkcję i reguły powiązane z interfejsem kolejki komunikatów (Message Queue Interface-MQI). Ta centralna część wymaga szeregu funkcji serwisowych, nazywanych *usługami instalowalnymi*, do wykonania swojej pracy. Instalowalną usługą dostępną w produkcie IBM MQ for IBM i jest usługa autoryzacji.

Każda instalowalna usługa jest pokrewnym zestawem funkcji zaimplementowanych przy użyciu jednego lub większej liczby *komponentów usług*. Każdy komponent jest wywoływany przy użyciu poprawnie zbudowanego, publicznie dostępnego interfejsu. Dzięki temu niezależni dostawcy oprogramowania i inne firmy mogą udostępniać instalowalne komponenty w celu rozszerzenia lub zastąpienia komponentów dostarczanych przez IBM MQ for IBM i. Tabela 140 na stronie 977 zawiera podsumowanie obsługi usługi autoryzacji.

*Tabela 140. Podsumowanie komponentów usługi autoryzacji*

| Dostarczony komponent | Funkcja | Wymagania |
|---|---|---|
| menedżer uprawnień obiektu (OAM) | Udostępnia sprawdzanie autoryzacji dla komend i wywołań MQI. Użytkownicy mogą napisać własny komponent w celu rozszerzenia lub zastąpienia OAM. | (Przyjmuje się odpowiednie narzędzia autoryzacji platformy) |
| Komponent usługi nazw DCE<br><br>**Uwaga:** DCE jest obsługiwane tylko w wersjach systemu IBM MQ wcześniejszych niż V6.0. | • Umożliwia menedżerom kolejek współużytkowanie kolejek lub<br>• Zdefiniowany przez użytkownika<br><br>**Uwaga:** Kolejki współużytkowane muszą mieć atrybut **Scope** ustawiony na wartość CELL. | • DCE jest wymagane dla podanego komponentu lub<br>• Menedżer nazw innej firmy lub napisany przez użytkownika |

### IBM i *Funkcje i komponenty w systemie IBM i*

Te informacje umożliwiają zrozumienie funkcji i komponentów, punktów wejścia, kodów powrotu i danych komponentu, których można użyć w programie IBM MQ for IBM i.

Każda usługa składa się z zestawu powiązanych funkcji. Na przykład usługa nazw zawiera funkcję dla:

- Wyszukiwanie nazwy kolejki i zwracanie nazwy menedżera kolejek, w którym zdefiniowano kolejkę
- Wstawianie nazwy kolejki do katalogu usługi

- Usuwanie nazwy kolejki z katalogu usługi

Zawiera również funkcje inicjowania i zakończenia.

Instalowalna usługa jest udostępniana przez co najmniej jeden komponent usługi. Każdy komponent może wykonywać niektóre lub wszystkie funkcje zdefiniowane dla tej usługi. Komponent jest również odpowiedzialny za zarządzanie zasobami bazowymi lub oprogramowaniem, które są potrzebne do zaimplementowania usługi. Pliki konfiguracyjne udostępniają standardowy sposób ładowania komponentu i określania adresów udostępnianych przez niego procedur funkcjonalnych.

Usługi i komponenty są powiązane w następujący sposób:

- Usługa jest definiowana w menedżerze kolejek za pomocą sekcji w pliku konfiguracyjnym.
- Każda usługa jest obsługiwana przez kod dostarczony w menedżerze kolejek. Użytkownicy nie mogą zmieniać tego kodu i dlatego nie mogą tworzyć własnych usług.
- Każda usługa jest implementowana przez jeden lub więcej komponentów. Można je dostarczyć wraz z produktem lub napisem dla użytkownika. Można wywołać wiele komponentów dla usługi, z których każdy obsługuje różne narzędzia w ramach usługi.
- Punkty wejścia łączą komponenty usługi z kodem obsługi w menedżerze kolejek.

## Punkty wejścia

Każdy komponent usługi jest reprezentowany przez listę adresów punktów wejścia procedur, które obsługują konkretną usługę instalowalną. Instalowalna usługa definiuje funkcję, która ma być wykonywana przez każdą procedurę. Kolejność komponentów usługi podczas ich konfigurowania definiuje kolejność, w jakiej punkty wejścia są wywoływane w celu spełnienia żądania usługi. W dostarczonym pliku nagłówkowym `cmqzc.h`podane punkty wejścia do każdej usługi mają przedrostek MQZID_.

## Kody powrotu

Komponenty usług udostępniają menedżerowi kolejek kody powrotu umożliwiające raportowanie różnych warunków. Zgłaszają one powodzenie lub niepowodzenie operacji i wskazują, czy menedżer kolejek ma przejść do następnego komponentu usługi. Do wskazania tego służy osobny parametr *Continuation* .

## Dane komponentu

Pojedynczy komponent usługi może wymagać współużytkowania danych między różnymi funkcjami. Usługi instalowalne udostępniają opcjonalny obszar danych, który ma być przekazywany przy każdym wywołaniu określonego komponentu usługi. Ten obszar danych jest przeznaczony do wyłącznego użytku komponentu usługi. Jest ona współużytkowana przez wszystkie wywołania danej funkcji, nawet jeśli są one wykonywane z różnych przestrzeni adresowych lub procesów. Gwarantowana jest możliwość adresowania z komponentu usługi za każdym razem, gdy jest on wywoływany. Należy zadeklarować wielkość tego obszaru w sekcji *`ServiceComponent`* .

## IBM i *Inicjowanie w systemie IBM i*

Po wywołaniu procedury inicjowania komponentu musi ona wywołać funkcję MQZEP menedżera kolejek dla każdego punktu wejścia obsługiwanego przez komponent. MQZEP definiuje punkt wejścia do usługi. Przyjmuje się, że wszystkie niezdefiniowane punkty wyjścia mają wartość NULL.

**Inicjowanie podstawowe**
Komponent jest zawsze wywoływany z tą opcją raz, zanim zostanie wywołany w inny sposób.

**Inicjowanie dodatkowe**
Komponent można wywołać z tą opcją na niektórych platformach. Można go na przykład wywołać raz dla każdego procesu, wątku lub zadania systemu operacyjnego, za pomocą którego uzyskiwany jest dostęp do usługi.

Jeśli używane jest inicjowanie dodatkowe:

- Komponent można wywołać więcej niż jeden raz w celu zainicjowania dodatkowego. Dla każdego takiego wywołania, gdy usługa nie jest już potrzebna, jest wysyłane zgodne wywołanie dla dodatkowego zakończenia.

  W przypadku usług autoryzacji jest to wywołanie MQZ_TERM_AUTHORITY.
- Punkty wejścia muszą być ponownie określone (przez wywołanie MQZEP) za każdym razem, gdy komponent jest wywoływany w celu zainicjowania podstawowego i dodatkowego.
- Dla komponentu używana jest tylko jedna kopia danych komponentu; nie ma innej kopii dla każdego dodatkowego inicjowania.
- Komponent nie jest wywoływany dla żadnych innych wywołań usługi (z procesu, wątku lub czynności systemu operacyjnego, w zależności od przypadku) przed wykonaniem dodatkowego inicjowania.
- Komponent musi ustawić parametr **Version** na taką samą wartość dla inicjowania podstawowego i dodatkowego.

**Zakończenie podstawowe**
Komponent jest zawsze uruchamiany z tą opcją jeden raz, gdy nie jest już wymagany. Do tego komponentu nie są wykonywane żadne dalsze wywołania.

**Zakończenie wtórne**
Komponent jest uruchamiany z tą opcją, jeśli został uruchomiony dla dodatkowego inicjowania.

## IBM i *Konfigurowanie usług i komponentów w systemie IBM i*

Komponenty usług są konfigurowane przy użyciu plików konfiguracyjnych menedżera kolejek.

### Procedura

1. Dodaj sekcje do pliku konfiguracyjnego menedżera kolejek `qm.ini`, aby zdefiniować usługę dla menedżera kolejek i określić położenie modułu:
   - Każda używana usługa musi mieć sekcję `Service` , która definiuje usługę dla menedżera kolejek. Więcej informacji na ten temat zawiera sekcja Service pliku qm.ini.
   - Dla każdego komponentu w usłudze musi istnieć sekcja `ServiceComponent` . Ta sekcja identyfikuje nazwę i ścieżkę modułu zawierającego kod dla tego komponentu. Więcej informacji na ten temat zawiera sekcja ServiceComponent w pliku qm.ini.

   Wraz z produktem dostarczany jest komponent usługi autoryzacji, zwany menedżerem uprawnień do obiektów (Object Authority Manager-OAM). Podczas tworzenia menedżera kolejek plik konfiguracyjny menedżera kolejek jest automatycznie aktualizowany w celu uwzględnienia sekcji odpowiednich dla usługi autoryzacji i dla komponentu domyślnego (OAM). W przypadku innych komponentów należy ręcznie skonfigurować plik konfiguracyjny menedżera kolejek.

   Kod dla każdego komponentu usługi jest ładowany do menedżera kolejek podczas uruchamiania menedżera kolejek przy użyciu powiązania dynamicznego, jeśli jest to obsługiwane na platformie.
2.

## IBM i *Tworzenie własnego komponentu usługi w systemie IBM i*

Ten temat zawiera informacje dotyczące tworzenia komponentu usługi w systemie IBM MQ for IBM i.

Aby utworzyć własny komponent usługi:

- Upewnij się, że plik nagłówkowy `cmqzc.h` jest dołączony do programu.
- Utwórz bibliotekę współużytkowaną, kompilując program i łącząc go z bibliotekami współużytkowanymi `libmqm*` i `libmqmzf*`.

  **Uwaga:** Ponieważ agent może działać w środowisku wielowątkowym, należy utworzyć moduł OAM, aby działał w środowisku wielowątkowym. Obejmuje to używanie wersji wielowątkowych produktów `libmqm` i `libmqmzf`.

- Dodaj sekcje do pliku konfiguracyjnego menedżera kolejek, aby zdefiniować usługę dla menedżera kolejek i określić położenie modułu.
- Zatrzymaj i zrestartuj menedżer kolejek, aby aktywować komponent.

## IBM i *Usługa autoryzacji w systemie IBM i*

Usługa autoryzacji jest instalowalną usługą, która umożliwia menedżerom kolejek wywoływanie narzędzi autoryzacji, na przykład sprawdzenie, czy ID użytkownika ma uprawnienia do otwierania kolejki.

Ta usługa jest komponentem interfejsu SEI ( IBM MQ Security Włączanie Interface), który jest częścią środowiska IBM MQ . Omówione zostały następujące tematy:


- "menedżer uprawnień obiektu (OAM)" na stronie 980
- "Definiowanie usługi dla systemu operacyjnego" na stronie 980
- "Konfigurowanie sekcji usługi autoryzacji" na stronie 980
- "Interfejs usługi autoryzacji w systemie IBM i" na stronie 981


### menedżer uprawnień obiektu (OAM)

Komponent usługi autoryzacji dostarczany z produktami IBM MQ jest nazywany menedżerem uprawnień do obiektów (object authority manager-OAM). Domyślnie mechanizm OAM jest aktywny i działa z następującymi komendami sterującymi:

- **WRKMQMAUT** praca z uprawnieniami
- **WRKMQMAUTD** praca z danymi uprawnień
- **DSPMQMAUT** wyświetlanie uprawnień do obiektu
- **GRTMQMAUT** nadawanie uprawnień do obiektu
- **RVKMQMAUT** odebranie uprawnienia do obiektu
- **RFRMQMAUT** Odśwież zabezpieczenia

Składnia tych komend i sposób ich użycia zostały opisane w pomocy do komend CL. OAM współpracuje z *jednostką* nazwy użytkownika lub grupy.

Po wykonaniu żądania MQI lub komendy moduł OAM sprawdza autoryzację jednostki powiązanej z operacją, aby sprawdzić, czy może wykonać następujące działania:

- Wykonaj żądaną operację.
- Uzyskaj dostęp do określonych zasobów menedżera kolejek.

Usługa autoryzacji umożliwia rozszerzenie lub zastąpienie sprawdzania uprawnień udostępnionego dla menedżerów kolejek przez napisanie własnego komponentu usługi autoryzacji.

### Definiowanie usługi dla systemu operacyjnego

Sekcje usługi autoryzacji w pliku konfiguracyjnym menedżera kolejek `qm.ini` definiują usługę autoryzacji dla menedżera kolejek. Więcej informacji na temat typów sekcji zawiera sekcja "Konfigurowanie usług i komponentów w systemie IBM i" na stronie 979 .

### Konfigurowanie sekcji usługi autoryzacji

W systemie IBM MQ for IBM i:

**Kolumnowo-wierszowa**
Jest profilem użytkownika systemu IBM i .

**Grupa**
Jest profilem grupy systemów IBM i .

Autoryzacje mogą być nadawane lub odbierane tylko na poziomie grupy. Żądanie nadania lub odwołania uprawnień użytkownika powoduje zaktualizowanie grupy podstawowej dla tego użytkownika.

Każdy menedżer kolejek ma własny plik konfiguracyjny menedżera kolejek. Na przykład domyślna ścieżka i nazwa pliku konfiguracyjnego menedżera kolejek QMNAME to `/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini`.

Sekcja *`Service`* i sekcja *`ServiceComponent`* dla domyślnego komponentu autoryzacji są dodawane automatycznie do pliku `qm.ini` , ale mogą być nadpisywane przez `WRKENVVAR`. Wszystkie pozostałe sekcje *`ServiceComponent`* należy dodawać ręcznie.

Na przykład następujące sekcje pliku konfiguracyjnego menedżera kolejek definiują dwa komponenty usługi autoryzacji:

```
Service:
    Name=AuthorizationService
    EntryPoints=7

ServiceComponent:
    Service=AuthorizationService
    Name=MQ.UNIX.authorization.service
    Module=QMQM/AMQZFU
    ComponentDataSize=0

ServiceComponent:
    Service=AuthorizationService
    Name=user.defined.authorization.service
    Module=LIBRARY/SERVICE PROGRAM NAME
    ComponentDataSize=96
```

*Rysunek 100. Sekcje usługi autoryzacji w pliku qm.ini w systemie IBM i*

Pierwsza sekcja komponentu usługi `MQ.UNIX.authorization.service` definiuje domyślny komponent usługi autoryzacji, OAM. W przypadku usunięcia tej sekcji i zrestartowania menedżera kolejek funkcja OAM jest wyłączona i nie są wykonywane żadne sprawdzenia autoryzacji.

IBM i *Interfejs usługi autoryzacji w systemie IBM i*

Interfejs usługi autoryzacji udostępnia kilka punktów wejścia do użycia przez menedżer kolejek.

**MQZ_AUTHENTICATE_USER**
Uwierzytelnia ID użytkownika i hasło oraz może ustawić pola kontekstu tożsamości.

**MQZ_CHECK_AUTHORITY (uprawnienie MQZ_CHECK_)**
Sprawdza, czy jednostka ma uprawnienia do wykonywania jednej lub kilku operacji na określonym obiekcie.

**MQZ_COPY_ALL_AUTHORITY,**
Kopiuje wszystkie bieżące autoryzacje istniejące dla obiektu odniesienia do innego obiektu.

**MQZ_DELETE_AUTHORITY, UPRAWNIENIE**
Usuwa wszystkie autoryzacje powiązane z określonym obiektem.

**MQZ_ENUMERATE_AUTHORITY_DATA**
Pobiera wszystkie dane uprawnień, które są zgodne z podanymi kryteriami wyboru.

**MQZ_FREE_USER**
Zwalnia powiązane przydzielone zasoby.

**MQZ_GET_AUTHORITY (uprawnienie GET_AUTHORITY)**
Pobiera uprawnienia, które jednostka ma, aby uzyskać dostęp do określonego obiektu.

**MQZ_GET_EXPLICIT_AUTHORITY**
Pobiera albo uprawnienie, które grupa nazwana ma, aby uzyskać dostęp do określonego obiektu (ale bez dodatkowych uprawnień grupy **nobody** (nikt)), albo uprawnienie, które grupa podstawowa nazwanej jednostki głównej ma, aby uzyskać dostęp do określonego obiektu.

**MQZ_INIT_AUTHORITY, UPRAWNIENIE**
Inicjuje komponent usługi autoryzacji.

**MQZ_INQUIRE**
Wysyła zapytanie o obsługiwaną funkcjonalność usługi autoryzacji.

**MQZ_REFRESH_CACHE**
Odśwież wszystkie autoryzacje.

**MQZ_SET_AUTHORITY (uprawnienia MQZ_SET_)**
Ustawia uprawnienie jednostki do określonego obiektu.

**MQZ_TERM_AUTHORITY (uprawnienia MQZ_TERM)**
Kończy działanie komponentu usługi autoryzacji.

Te punkty wejścia obsługują użycie identyfikatora bezpieczeństwa systemu Windows (NT SID).

Nazwy te są definiowane jako **typedef** w pliku nagłówkowym `cmqzc.h`, który może być używany do prototypowania funkcji komponentu.

Funkcja inicjowania ( **MQZ_INIT_AUTHORITY** ) musi być głównym punktem wejścia dla komponentu. Inne funkcje są wywoływane przez adres punktu wejścia, który funkcja inicjowania dodała do wektora punktu wejścia komponentu.

Więcej informacji zawiera sekcja "Tworzenie własnego komponentu usługi w systemie IBM i" na stronie 979.

## Multi Pisanie i kompilowanie wyjść funkcji API w wersji wieloplatformowej

Wyjścia funkcji API umożliwiają napisanie kodu, który zmienia zachowanie wywołań funkcji API języka IBM MQ , takich jak MQPUT i MQGET, a następnie wstawia ten kod bezpośrednio przed tymi wywołaniami lub bezpośrednio po nich.

**Uwaga:** z/OS Nieobsługiwane w systemie IBM MQ for z/OS.

### Dlaczego warto używać wyjść funkcji API?

Każda z aplikacji ma konkretną pracę do wykonania, a jej kod powinien wykonać to zadanie tak efektywnie, jak to możliwe. Na wyższym poziomie można zastosować standardy lub procesy biznesowe do konkretnego menedżera kolejek dla wszystkich aplikacji używających tego menedżera kolejek. Bardziej efektywne jest działanie powyżej poziomu poszczególnych aplikacji, a tym samym bez konieczności zmiany kodu każdej aplikacji, której to dotyczy.

Poniżej przedstawiono kilka sugestii dotyczących obszarów, w których mogą być przydatne wyjścia funkcji API:

**Zabezpieczenia**
Ze względów bezpieczeństwa można zapewnić uwierzytelnianie, sprawdzając, czy aplikacje mają uprawnienia dostępu do kolejki lub menedżera kolejek. Można również nadzorować użycie interfejsu API przez aplikacje, uwierzytelniać poszczególne wywołania interfejsu API, a nawet używane przez nie parametry.

**Elastyczność**
W celu zapewnienia elastyczności można reagować na szybkie zmiany w środowisku biznesowym bez konieczności modyfikowania aplikacji, które opierają się na danych w tym środowisku. Na przykład mogą istnieć wyjścia interfejsu API, które reagują na zmiany stóp procentowych, kursów wymiany walut lub cen komponentów w środowisku produkcyjnym.

**Monitorowanie użycia kolejki lub menedżera kolejek**
W celu monitorowania użycia kolejki lub menedżera kolejek można śledzić przepływ aplikacji i komunikatów, rejestrować błędy w wywołaniach API, konfigurować zapisy kontrolne na potrzeby rozliczania lub gromadzić statystyki użycia na potrzeby planowania.

### Co się dzieje, gdy uruchamiane jest wyjście funkcji API?

Po napiseniu programu obsługi wyjścia i zidentyfikowaniu go w programie IBM MQ, menedżer kolejek automatycznie wywołuje kod wyjścia w zarejestrowanych punktach.

Procedury wyjścia funkcji API do uruchomienia są identyfikowane w sekcjach na platformie Multiplatforms. W tym temacie opisano sekcje w plikach konfiguracyjnych `mqs.ini` i `qm.ini`.

Definicje procedur mogą występować w trzech miejscach:

1. ApiExitCommon w pliku mqs.ini identyfikuje podprogramy, dla całego pliku IBM MQ, stosowane podczas uruchamiania menedżerów kolejek. Mogą one zostać przesłonięte przez procedury zdefiniowane dla poszczególnych menedżerów kolejek (patrz element “3” na stronie 983 na tej liście).
2. Szablon ApiExitw pliku mqs.ini identyfikuje podprogramy dla całości produktu IBM MQ, które zostały skopiowane do zestawu lokalnego ApiExit(patrz element “3” na stronie 983 na tej liście) podczas tworzenia nowego menedżera kolejek.
3. ApiExitLokalny w pliku qm.ini identyfikuje procedury, które mają zastosowanie do konkretnego menedżera kolejek.

Po utworzeniu nowego menedżera kolejek definicje szablonów ApiExitw pliku mqs.ini są kopiowane do lokalnych definicji ApiExitw pliku qm.ini dla nowego menedżera kolejek. Po uruchomieniu menedżera kolejek używane są zarówno definicje wspólne ( ApiExit), jak i definicje lokalne ( ApiExit). Definicje lokalne ApiExitzastępują wspólne definicje ApiExit, jeśli obie identyfikują procedurę o tej samej nazwie. Atrybut `Sequence` opisany w sekcji “Konfigurowanie wyjść funkcji API” na stronie 988 określa kolejność uruchamiania procedur zdefiniowanych w sekcjach.

## Korzystanie z wyjść funkcji API w wielu instalacjach produktu IBM MQ

Upewnij się, że wyjścia funkcji API napisane dla wcześniejszej wersji produktu IBM MQ są używane do pracy ze wszystkimi wersjami, ponieważ zmiany wprowadzone w wyjściach w produkcie IBM WebSphere MQ 7.1 mogą nie działać z wcześniejszą wersją. Więcej informacji na temat zmian wprowadzonych w wyjść zawiera sekcja “Zapisywanie wyjść i usług instalowalnych w systemie AIX, Linux, and Windows” na stronie 965.

Przykłady udostępnione dla wyjść funkcji API amqsaem i amqsaxe odzwierciedlają zmiany wymagane podczas zapisywania wyjść. Aplikacja kliencka musi upewnić się, że przed uruchomieniem aplikacji są z nią powiązane poprawne biblioteki produktu IBM MQ odpowiadające instalacji menedżera kolejek, z którym jest powiązana aplikacja.

### Multi *Zapisywanie wyjść funkcji API*

Wyjścia można pisać dla każdego wywołania funkcji API przy użyciu języka programowania C.

## Dostępne wyjścia

Wyjścia są dostępne dla każdego wywołania funkcji API w następujący sposób:

- MQCB, aby ponownie zarejestrować wywołanie zwrotne dla określonego uchwytu obiektu i sterować aktywacją i zmianami wywołania zwrotnego
- MQCTL, aby wykonywać działania sterujące na uchwytach obiektów otwartych dla połączenia
- MQCONN/MQCONNX, aby udostępnić uchwyt połączenia menedżera kolejek do użycia w kolejnych wywołaniach API
- MQDISC, aby rozłączyć się z menedżerem kolejek
- MQBEGIN, aby rozpocząć globalną jednostkę pracy (UOW)
- MQBACK, wycofanie jednostki pracy
- MQCMIT, aby zatwierdzić jednostkę pracy
- MQOPEN, aby otworzyć zasób IBM MQ w celu późniejszego dostępu
- MQCLOSE, aby zamknąć zasób IBM MQ , który został wcześniej otwarty w celu uzyskania do niego dostępu
- MQGET, aby pobrać komunikat z kolejki, do której wcześniej otwarto dostęp
- MQPUT1, aby umieścić komunikat w kolejce

- MQPUT-umieszczanie komunikatu w kolejce, do której wcześniej otwarto dostęp
- MQINQ, aby uzyskać informacje na temat atrybutów zasobu IBM MQ , który został wcześniej otwarty dla dostępu
- MQSET: służy do ustawiania atrybutów kolejki, do której wcześniej otwarto dostęp.
- MQSTAT, do pobierania informacji o statusie
- MQSUB, aby zarejestrować subskrypcję aplikacji w konkretnym temacie
- MQSUBRQ, aby utworzyć żądanie subskrypcji

Usługa MQ_CALLBACK_EXIT udostępnia funkcję wyjścia, która ma być wykonywana przed i po przetwarzaniu wywołania zwrotnego. Więcej informacji na ten temat zawiera sekcja Wywołanie zwrotne-MQ_CALLBACK_EXIT.

## Zapisywanie wyjść funkcji API

W obrębie wyjść funkcji API wywołania przyjmują ogólną postać:

```
MQ_call_EXIT (parameters, context, ApiCallParameters)
```

gdzie *call* jest nazwą wywołania MQI bez przedrostka MQ , na przykład PUT, GET. *parameters* steruje funkcją wyjścia, zapewniając przede wszystkim komunikację między wyjściem a zewnętrznymi blokami sterującymi MQAXP (struktura parametru wyjścia funkcji API) i MQAXC (struktura kontekstu wyjścia funkcji API). *context* opisuje kontekst, w którym wywołano wyjście funkcji API, a *ApiCallParameters* reprezentuje parametry wywołania MQI.

Aby ułatwić napisanie wyjścia funkcji API, udostępniono przykładowe wyjście amqsaxe0.c. To wyjście generuje wpisy śledzenia do podanego pliku. Tego przykładu można użyć jako punktu początkowego podczas zapisywania wyjść. Więcej informacji na temat używania przykładowego wyjścia zawiera sekcja “Przykładowy program obsługi wyjścia funkcji API” na stronie 1107.

Więcej informacji na temat wywołań wyjścia funkcji API, zewnętrznych bloków sterujących i powiązanych tematów zawiera sekcja Skorowidz wyjścia funkcji API.

Ogólne informacje na temat zapisywania, kompilowania i konfigurowania wyjścia zawiera sekcja “Zapisywanie wyjść i usług instalowalnych w systemie AIX, Linux, and Windows” na stronie 965.

## Używanie uchwytów komunikatów w wyjściach funkcji API

Istnieje możliwość sterowania właściwościami komunikatów, do których ma dostęp wyjście funkcji API. Właściwości są powiązane z uchwytem ExitMsg. Właściwości ustawione w wyjściu umieszczania są ustawiane dla umieszczanego komunikatu, ale właściwości pobrane w wyjściu pobierania nie są zwracane do aplikacji.

Po zarejestrowaniu funkcji wyjścia MQ_INIT_EXIT przy użyciu wywołania MQI MQXEP z opcją **Function** ustawioną na wartość MQXF_INIT i **ExitReason** ustawioną na wartość MQXR_CONNECTION należy przekazać strukturę MQXEPO jako parametr **ExitOpts** . Struktura MQXEPO zawiera pole ExitProperties , które określa zestaw właściwości, które mają być dostępne dla wyjścia. Jest on określony jako łańcuch znaków reprezentujący przedrostek właściwości, który odpowiada nazwie folderu MQRFH2 .

Każde wyjście funkcji API odbiera strukturę MQAXP zawierającą pole uchwytu ExitMsg. W tym polu jest ustawiona wartość generowana przez program IBM MQ i jest ona specyficzna dla połączenia. Dlatego uchwyt pozostaje niezmieniony między wyjściami funkcji API tego samego lub różnych typów w tym samym połączeniu.

W produkcie MQ_PUT_EXIT lub MQ_PUT1_EXIT z wartością **ExitReason** ustawioną na wartość MQXR_BEFORE, to znaczy wyjście funkcji API wykonywane przed umieszczeniem komunikatu, wszystkie właściwości (inne niż właściwości deskryptora komunikatu) powiązane z uchwytem ExitMsgpo zakończeniu działania wyjścia są ustawiane w umieszczanym komunikacie. Aby temu zapobiec, należy ustawić uchwyt ExitMsgna wartość MQHM_NONE. Można również podać inny uchwyt komunikatu.

W przypadku właściwości MQ_GET_EXIT i MQ_CALLBACK_EXIT uchwyt ExitMsgjest czyszczony z właściwości i zapełniany właściwościami określonymi w polu ExitProperties podczas rejestrowania pliku MQ_INIT_EXIT, innymi niż właściwości deskryptora komunikatu. Te właściwości nie są dostępne dla aplikacji pobierającej. Jeśli aplikacja pobierająca określa uchwyt komunikatu w polu MQGMO (opcje pobierania komunikatu), wszystkie właściwości powiązane z tym uchwytem, w tym właściwości deskryptora komunikatu, są dostępne dla wyjścia funkcji API. Aby zapobiec zapełnieniu uchwytu ExitMsgwłaściwościami, należy ustawić jego wartość na MQHM_NONE.

**Uwaga:** Dla właściwości komunikatu wyjścia, które mają być przetwarzane w:

- Po funkcji MQ_GET_EXIT należy zdefiniować funkcję przed funkcją MQ_GET_EXIT dla wyjścia.
- Przed funkcją MQ_CALLBACK_EXIT należy zdefiniować funkcję przed funkcją MQ_CB_EXIT dla wyjścia.

V 9.3.2 W produkcie IBM MQ 9.3.2poprzednie instrukcje dotyczące produktów MQ-GET-EXIT i MQ_CALLBACK_EXIT nie mają już zastosowania.

W celu zilustrowania użycia uchwytów komunikatów w wyjściach funkcji API udostępniono przykładowy program amqsaem0.c.

**Odsyłacze pokrewne**

Informacje dodatkowe o wyjściach użytkownika, wyjściach funkcji API i instalowalnych usługach

## Multi *Kompilowanie wyjść funkcji API*

Po napisaniu wyjścia należy je skompilować i skonsolidować w następujący sposób.

Poniższe przykłady przedstawiają komendy używane przez przykładowy program opisany w sekcji “Przykładowy program obsługi wyjścia funkcji API” na stronie 1107. Na platformach innych niż systemy Windows przykładowy kod wyjścia funkcji API można znaleźć w sekcji *MQ_INSTALLATION_PATH*/samp , a skompilowaną i dowiązaną bibliotekę współużytkowaną-w sekcji *MQ_INSTALLATION_PATH*/samp/bin.

Windows W systemach Windows przykładowy kod wyjścia funkcji API można znaleźć w sekcji *MQ_INSTALLATION_PATH* \Tools\c\Samples. *MQ_INSTALLATION_PATH* reprezentuje katalog, w którym zainstalowano produkt IBM MQ .

**Uwaga:** Wskazówki dotyczące programowania aplikacji 64-bitowych zawiera sekcja Standardy kodowania na platformach 64-bitowych.

W przypadku klientów rozsyłania grupowego wyjścia funkcji API i wyjścia konwersji danych muszą działać po stronie klienta, ponieważ niektóre komunikaty mogą nie przechodzić przez menedżer kolejek. Następujące biblioteki są częścią pakietów klienta oraz pakietów serwera:

*Tabela 141. Biblioteki, które znajdują się w pakietach klienta i serwera*

| System operacyjny | Biblioteki |
|---|---|
| AIX AIX | Wersja 32-bitowa i 64-bitowa: libmqm.a & libmqm_r.a |
| IBM i IBM i | LIBMQM i LIBMQM_R |
| Linux Linux | Wersja 32-bitowa i 64-bitowa: libmqm.so & libmqm_r.so |
| Windows Windows | Wersja 32-bitowa i 64-bitowa: mqm.dll & mqm.pdb |

Linux AIX *Kompilowanie wyjść funkcji API w systemach AIX and Linux*

Przykłady kompilowania wyjść funkcji API w systemach AIX and Linux .

Na wszystkich platformach punktem wejścia do modułu jest MQStart.

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

## wł.AIX

AIX

Skompiluj kod źródłowy wyjścia funkcji API, wydając jedną z następujących komend:

**Aplikacje 32-bitowe**

**Niewątkowy**

```
cc -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

**Wątkowy**

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

**Aplikacje 64-bitowe**

**Niewątkowy**

```
cc -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

**Wątkowy**

```
xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

## wł.Linux

Linux

Skompiluj kod źródłowy wyjścia funkcji API, wydając jedną z następujących komend:

**Aplikacje 31-bitowe**

**Niewątkowy**

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

**Wątkowy**

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

**Aplikacje 32-bitowe**

**Niewątkowy**

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

**Wątkowy**

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

**Aplikacje 64-bitowe**

**Niewątkowy**

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

**Wątkowy**

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Windows *Kompilowanie wyjść funkcji API w systemach Windows*

Skompiluj i dowiąż przykładowy program obsługi wyjścia funkcji API, `amqsaxe0.c`, w systemie Windows

Plik manifestu jest opcjonalnym dokumentem XML zawierającym wersję lub inne informacje, które mogą być osadzone w skompilowanej aplikacji lub bibliotece DLL.

Jeśli taki dokument nie jest dostępny, należy pominąć parametr -manifest *manifest.file* w komendzie **mt** .

Dostosuj komendy w przykładach w pliku Rysunek 101 na stronie 987 lub Rysunek 102 na stronie 987 , aby skompilować i skonsolidować plik `amqsaxe0.c` w systemie Windows. Komendy działają z systemem Microsoft Visual Studio 2008, 2010 lub 2012. W przykładach założono, że katalog `C:\Program Files\IBM\MQ\tools\c\samples` jest katalogiem bieżącym.

**32-bitowa**

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def



amqsaxe0.obj \
   /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

*Rysunek 101. Kompilowanie i konsolidowanie `amqsaxe0.c` w 32-bitowym systemie Windows*

**64 bity**

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def \
     /libpath:..\..\lib64 \



amqsaxe0.obj /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

*Rysunek 102. Kompilacja i konsolidacja `amqsaxe0.c` w 64-bitowym systemie Windows*

**Pojęcia pokrewne**

“Przykładowy program obsługi wyjścia funkcji API” na stronie 1107

Przykładowe wyjście funkcji API generuje śledzenie MQI do pliku określonego przez użytkownika z przedrostkiem zdefiniowanym w zmiennej środowiskowej **MQAPI_TRACE_LOGFILE** .

IBM i *Zgodność wyjść funkcji API w systemie IBM i*
Kompilowanie wyjść funkcji API w systemie IBM i.

Wyjście jest tworzone w następujący sposób (dla przykładu w języku C):

1. Utwórz moduł przy użyciu komendy CRTCMOD. Skompiluj go, aby używał teraprzestrzeni, podając parametr TERASPACE(*YES *TSIFC).
2. Utwórz program usługowy z modułu przy użyciu komendy CRTSRVPGM. Należy go powiązać z programem usługowym QMQM/LIBMQMZF_R dla wielowątkowych wyjść funkcji API.

## *Konfigurowanie wyjść funkcji API*

Produkt IBM MQ można skonfigurować w celu włączenia wyjść funkcji API przez zmianę informacji konfiguracyjnych.

Aby zmienić informacje konfiguracyjne, należy zmienić sekcje definiujące procedury obsługi wyjścia oraz kolejność ich uruchamiania. Te informacje można zmienić w następujący sposób:

- Windows Linux Przy użyciu systemu IBM MQ Explorer na platformach Windows i Linux (x86 i x86-64 ).
- Windows Przy użyciu komendy **amqmdain** w systemie Windows.
- Multi Korzystanie z plików mqs.ini i qm.ini bezpośrednio na platformie Multiplatforms.

  Plik mqs.ini zawiera informacje istotne dla wszystkich menedżerów kolejek w danym węźle. Można go znaleźć w następujących miejscach:

  – Linux AIX W katalogu /var/mqm w systemie AIX and Linux.

  – Windows W polu WorkPath określonym w kluczu HKLM\SOFTWARE\IBM\WebSphere MQ w systemach Windows .

  – IBM i W katalogu /QIBM/UserData/mqm w systemie IBM i.

  Plik qm.ini zawiera informacje dotyczące konkretnego menedżera kolejek. Dla każdego menedżera kolejek istnieje jeden plik konfiguracyjny menedżera kolejek, który znajduje się w katalogu głównym drzewa katalogów zajmowanego przez menedżer kolejek. Na przykład ścieżka i nazwa pliku konfiguracyjnego dla menedżera kolejek o nazwie QMNAME to:

  IBM i W systemach IBM i :

  ```
  /QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini
  ```

  Linux AIX W systemach AIX and Linux :

  ```
  /var/mqm/qmgrs/QMNAME/qm.ini
  ```

  Windows W systemach Windows :

  ```
  C:\ProgramData\IBM\MQ\qmgrs\QMNAME\qm.ini
  ```

  Przed edycją pliku konfiguracyjnego należy utworzyć jego kopię zapasową, aby w razie potrzeby można było przywrócić jego kopię.

  Pliki konfiguracyjne można edytować w jeden z następujących sposobów:

  – Automatycznie, przy użyciu komend zmieniających konfigurację menedżerów kolejek w węźle.

– Ręcznie, przy użyciu standardowego edytora tekstu.

Jeśli dla atrybutu pliku konfiguracyjnego zostanie ustawiona niepoprawna wartość, zostanie ona zignorowana i zostanie wyświetlony komunikat operatora informujący o problemie. Efekt jest taki sam, jak brak atrybutu w całości.

## Sekcje do skonfigurowania

Sekcje, które należy zmienić, są następujące:

**ApiExitCommon**

Zdefiniowane w pliku `mqs.ini` i w pliku IBM MQ Explorer na stronie właściwości IBM MQ w sekcji Wyjścia.

Po uruchomieniu dowolnego menedżera kolejek atrybuty w tej sekcji są odczytywane, a następnie nadpisywane przez wyjścia funkcji API zdefiniowane w pliku `qm.ini`.

**ApiExitTemplate**

Zdefiniowane w pliku `mqs.ini` i w pliku IBM MQ Explorer na stronie właściwości IBM MQ w sekcji Wyjścia.

Po utworzeniu dowolnego menedżera kolejek atrybuty w tej sekcji są kopiowane do nowo utworzonego pliku qm.ini w sekcji lokalnej ApiExit.

**ApiExitLocal**

Zdefiniowane w pliku `qm.ini` i w pliku IBM MQ Explorer na stronie właściwości menedżera kolejek w sekcji Wyjścia.

Po uruchomieniu menedżera kolejek wyjścia funkcji API zdefiniowane w tym miejscu przesłaniają wartości domyślne zdefiniowane w pliku `mqs.ini`.

## Atrybuty sekcji

- Nadaj nazwę wyjściu funkcji API przy użyciu następującego atrybutu:

  **Nazwa = ApiExit_name**
  Opisowa nazwa wyjścia funkcji API przekazana do niego w polu nazwy ExitInfostruktury MQAXP.

  Ta nazwa musi być unikalna, nie może być dłuższa niż 48 znaków i może zawierać tylko poprawne znaki dla nazw obiektów IBM MQ (na przykład nazw kolejek).

- Zidentyfikuj moduł i punkt wejścia kodu wyjścia funkcji API do uruchomienia przy użyciu następujących atrybutów:

  **Function=nazwa_funkcji**
  Nazwa punktu wejścia funkcji w module zawierającym kod wyjścia funkcji API. Ten punkt wejścia to funkcja MQ_INIT_EXIT.

  Wielkość tego pola jest ograniczona do wartości MQ_EXIT_NAME_LENGTH.

  **Module=nazwa_modułu**
  Moduł zawierający kod wyjścia funkcji API.

  Jeśli w polu znajduje się pełna nazwa ścieżki do modułu, jest ona używana w takiej postaci.

  Jeśli to pole zawiera tylko nazwę modułu, moduł znajduje się przy użyciu atrybutu `ExitsDefaultPath` w pliku `ExitPath` w pliku qm.ini.

  Na platformach, które obsługują oddzielne biblioteki wielowątkowe, należy udostępnić zarówno niewielowątkową, jak i wielowątkową wersję modułu wyjścia funkcji API. Wersja wielowątkowa musi mieć przyrostek `_r` . Wielowątkowa wersja kodu pośredniczącego aplikacji IBM MQ niejawnie dodaje łańcuch `_r` do danej nazwy modułu przed załadowaniem.

  Długość tego pola jest ograniczona do maksymalnej długości ścieżki obsługiwanej przez platformę.

- Opcjonalnie przekaż dane z wyjściem, używając następującego atrybutu:

**Data=nazwa_danych**
Dane, które mają zostać przekazane do wyjścia funkcji API w polu ExitData struktury MQAXP.

Jeśli ten atrybut zostanie podany, spacje początkowe i końcowe zostaną usunięte, pozostały łańcuch zostanie obcięty do 32 znaków, a wynik zostanie przekazany do wyjścia. Jeśli ten atrybut zostanie pominięty, do wyjścia zostanie przekazana wartość domyślna wynosząca 32 odstępy.

Maksymalna długość tego pola wynosi 32 znaki.

- Zidentyfikuj sekwencję tego wyjścia w odniesieniu do innych wyjść, używając następującego atrybutu:

**Sequence=numer_kolejny**
Kolejność wywoływania tego wyjścia funkcji API względem innych wyjść funkcji API. Wyjście z niskim numerem kolejnym jest wywoływane przed wyjściem z wyższym numerem kolejnym. Nie ma potrzeby, aby numeracja sekwencji wyjść była ciągła. Sekwencja 1, 2, 3 ma taki sam wynik jak sekwencja 7, 42, 1096. Jeśli dwa wyjścia mają ten sam numer kolejny, menedżer kolejek decyduje o tym, które z nich należy wywołać jako pierwsze. Informacje o tym, które zdarzenie zostało wywołane po zdarzeniu, można uzyskać, umieszczając znacznik czasu lub znacznik w obszarze ExitChainwskazywanym przez opcję ExitChainAreaPtr w MQAXP lub zapisując własny plik dziennika.

Ten atrybut jest wartością liczbową bez znaku.

## Przykładowe sekcje

Przykładowy plik mqs.ini zawiera następujące sekcje:

**ApiExitTemplate**
Ta sekcja definiuje wyjście z nazwą opisową `OurPayrollQueueAuditor`, nazwą modułu `auditor`i numerem kolejnym 2. Wartość danych 123 jest przekazywana do wyjścia.

**ApiExitCommon**
W tej sekcji zdefiniowano wyjście z nazwą opisową `MQPoliceman`, nazwą modułu `tmqp`i numerem kolejnym 1. Przekazywane dane są instrukcją ( `CheckEverything`).

```
mqs.ini

   ApiExitTemplate:
      Name=OurPayrollQueueAuditor
      Sequence=2
      Function=EntryPoint
      Module=/usr/ABC/auditor
      Data=123
   ApiExitCommon:
      Name=MQPoliceman
      Sequence=1
      Function=EntryPoint
      Module=/usr/MQPolice/tmqp
      Data=CheckEverything
```

Poniższy przykładowy plik qm.ini zawiera lokalną definicję wyjścia ApiExito nazwie opisowej `ClientApplicationAPIchecker`, nazwie modułu `ClientAppChecker`i numerze kolejnym 3.

```
qm.ini

   ApiExitLocal:
      Name=ClientApplicationAPIchecker
      Sequence=3
      Function=EntryPoint
      Module=/usr/Dev/ClientAppChecker
      Data=9.20.176.20
```

## Kanał-programy obsługi wyjścia dla kanałów przesyłania komunikatów

Ta kolekcja tematów zawiera informacje o programach obsługi wyjścia kanału IBM MQ dla kanałów przesyłania komunikatów.

Agenty kanału komunikatów (MCA) mogą również wywoływać wyjścia konwersji danych. Więcej informacji na temat zapisywania wyjść konwersji danych zawiera sekcja “Zapisywanie wyjść konwersji danych” na stronie 1013.

Niektóre z tych informacji dotyczą również wyjść w kanałach MQI, które łączą produkt IBM MQ MQI clients z menedżerami kolejek. Więcej informacji na ten temat zawiera sekcja Programy obsługi wyjścia kanału dla kanałów MQI.

Programy wyjścia kanału są wywoływane w zdefiniowanych miejscach przetwarzania wykonywanego przez programy MCA.

Niektóre z tych programów obsługi wyjścia użytkownika działają w pary komplementarne. Jeśli na przykład wysyłający agent MCA wywołał program obsługi wyjścia użytkownika w celu zaszyfrowania komunikatów do transmisji, proces komplementarny musi działać na odbierającym końcu, aby odwrócić proces.

Tabela 142 na stronie 991 przedstawia typy wyjść kanału, które są dostępne dla każdego typu kanału.

*Tabela 142. Wyjścia kanału dostępne dla każdego typu kanału*

| Typ kanału | Wyjście komunikatu | Wyjście ponowienia komunikatu | Wyjście odbierania | Wyjście zabezpieczeń | Wyjście wysyłania | Wyjście automatycznej definicji |
|---|---|---|---|---|---|---|
| Kanał nadawcy | Tak | | Tak | Tak | Tak | |
| Kanał serwera | Tak | | Tak | Tak | Tak | |
| Kanał nadawczy klastra | Tak | | Tak | Tak | Tak | Tak |
| Kanał odbiorcy | Tak | Tak | Tak | Tak | Tak | Tak |
| Kanał requestera | Tak | Tak | Tak | Tak | Tak | |
| Kanał odbiorczy klastra | Tak | Tak | Tak | Tak | Tak | Tak |
| Kanał połączenia klienckiego | | | Tak | Tak | Tak | |
| Kanał połączenia z serwerem | | | Tak | Tak | Tak | Tak |

**Uwagi:** z/OS

1. W systemie z/OSautomatycznie zdefiniowane wyjście ma zastosowanie tylko do kanałów nadawczych i odbiorczych klastra.

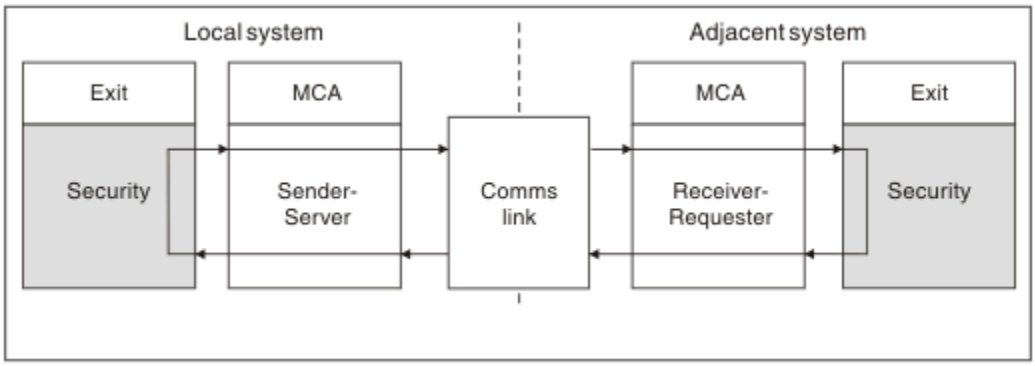
Jeśli wyjścia kanału mają być uruchamiane na kliencie, nie można użyć zmiennej środowiskowej MQSERVER. Zamiast tego należy utworzyć tabelę definicji kanału klienta (CCDT) i odwołać się do niej zgodnie z opisem w sekcji Tabela definicji kanału klienta.

## *Przetwarzanie-przegląd*

Przegląd sposobu, w jaki adaptery MCA używają programów obsługi wyjścia kanału.

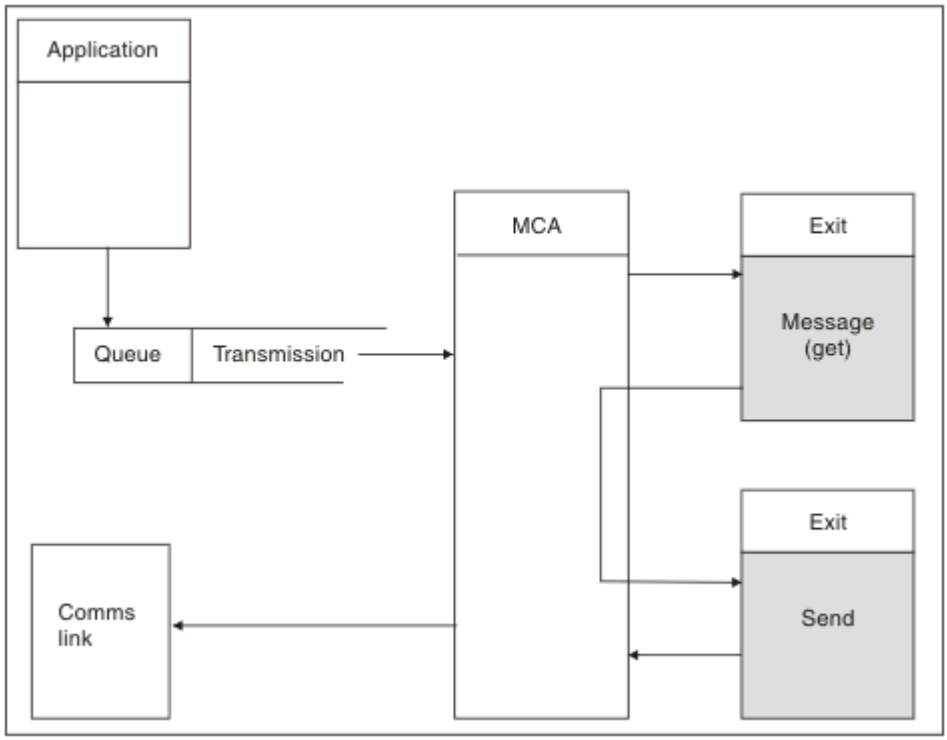
Podczas uruchamiania adaptery MCA wymieniają się oknem dialogowym uruchamiania w celu zsynchronizowania przetwarzania. Następnie przełączają się na wymianę danych, która obejmuje wyjścia bezpieczeństwa. Aby zakończyć fazę uruchamiania i umożliwić przesyłanie komunikatów, te wyjścia muszą zakończyć się pomyślnie.

Faza sprawdzania zabezpieczeń jest pętlą, co przedstawia Rysunek 103 na stronie 992.

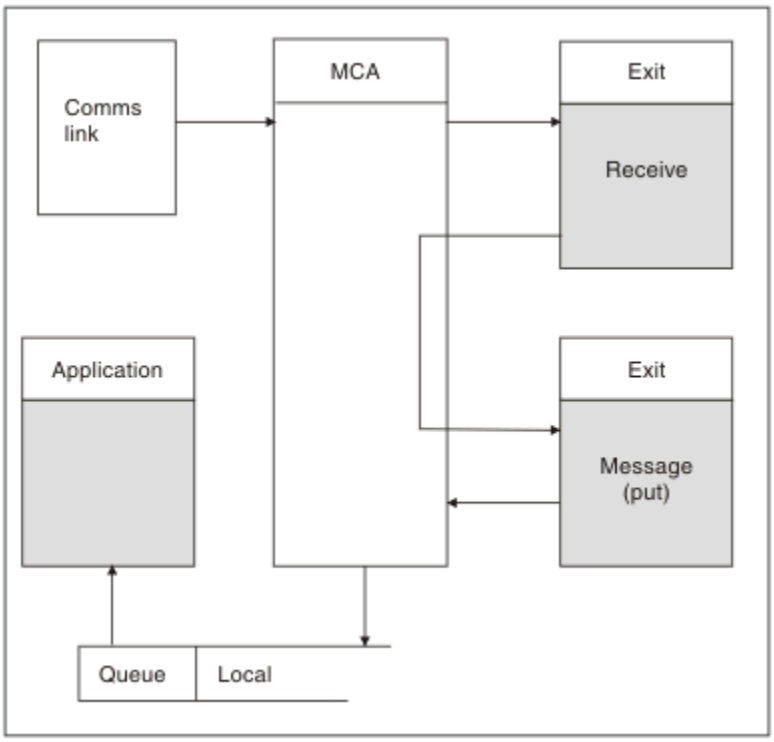



*Rysunek 103. Pętla wyjścia zabezpieczeń*

Podczas fazy przesyłania komunikatu wysyłający agent MCA pobiera komunikaty z kolejki transmisji, wywołuje wyjście komunikatu, wywołuje wyjście wysyłania, a następnie wysyła komunikat do odbierającego agenta MCA, zgodnie z opisem w sekcji Rysunek 104 na stronie 992.




*Rysunek 104. Przykład wyjścia wysyłania na końcu kanału komunikatów nadawcy*




*Rysunek 105. Przykład wyjścia odbierania na odbiorczym końcu kanału komunikatów*

Odbierający agent MCA odbiera komunikat z łącza komunikacyjnego, wywołuje wyjście odbierania, wywołuje wyjście komunikatu, a następnie umieszcza komunikat w kolejce lokalnej (patrz Rysunek 105 na stronie 993). (Wyjście odbierania może zostać wywołane więcej niż jeden raz przed wywołaniem wyjścia komunikatu).

## *Zapisywanie programów obsługi wyjścia kanału*

Poniższe informacje mogą być pomocne podczas pisania programów obsługi wyjścia kanału.

Programy użytkownika i programy obsługi wyjścia kanału mogą używać wszystkich wywołań MQI, z wyjątkiem opisanych w kolejnych sekcjach. W przypadku produktu MQ V7 lub nowszej struktura MQCXP w wersji 7 lub nowszej zawiera uchwyt połączenia hConn, którego można użyć zamiast wywołania MQCONN. W przypadku wcześniejszych wersji, aby uzyskać uchwyt połączenia, należy wykonać komendę MQCONN, nawet jeśli zostało zwrócone ostrzeżenie MQRC_ALREADY_CONNECTED, ponieważ kanał jest połączony z menedżerem kolejek.

Należy zauważyć, że wyjście kanału musi być wątkowo bezpieczne.

W przypadku wyjść w kanałach połączeń klienta menedżer kolejek, z którym wyjście próbuje nawiązać połączenie, zależy od sposobu połączenia wyjścia. Jeśli wyjście zostało połączone z produktem MQM.LIB (lub QMQM/LIBMQM w systemie IBM i ) jeśli w wywołaniu MQCONN nie zostanie podana nazwa menedżera kolejek, wyjście podejmie próbę nawiązania połączenia z domyślnym menedżerem kolejek w systemie. Jeśli wyjście zostało połączone z produktem MQM.LIB (lub QMQM/LIBMQM w systemie IBM i ) jeśli zostanie podana nazwa menedżera kolejek, która została przekazana do wyjścia za pośrednictwem pola QMgrName w programie MQCD, program zewnętrzny spróbuje nawiązać połączenie z tym menedżerem kolejek. Jeśli wyjście zostało połączone z produktem MQIC.LIB lub innej biblioteki wywołanie MQCONN kończy się niepowodzeniem bez względu na to, czy określono nazwę menedżera kolejek.

Należy unikać zmiany stanu transakcji powiązanej z przekazanym hConn w wyjściu kanału. Nie można używać komend MQCMIT, MQBACK ani MQDISC z kanałem hConni nie można używać komendy MQBEGIN z określeniem kanału hConn.

Jeśli komenda MQCONNX jest używana z opcją MQCNO_HANDLE_SHARE_BLOCK lub MQCNO_HANDLE_SHARE_NO_BLOCK w celu utworzenia nowego połączenia IBM MQ , użytkownik jest odpowiedzialny za zapewnienie poprawnego zarządzania połączeniem i jego poprawnego rozłączenia z menedżerem kolejek. Na przykład wyjście kanału, które tworzy nowe połączenie z menedżerem kolejek przy każdym wywołaniu bez rozłączania, powoduje utworzenie uchwytów połączenia i zwiększenie liczby wątków agenta.

Wyjście działa w tym samym wątku co agent MCA i używa tego samego uchwytu połączenia. Oznacza to, że działa w tej samej jednostce pracy, co agent MCA, a wszystkie wywołania wykonywane w punkcie synchronizacji są zatwierdzane lub wycofywane przez kanał na końcu zadania wsadowego.

Dlatego wyjście komunikatów kanału może wysyłać komunikaty powiadomień, które są zatwierdzone w tej kolejce tylko wtedy, gdy zatwierdzone zostanie zadanie wsadowe zawierające oryginalny komunikat. Dlatego można wywołać wywołania MQI punktu synchronizacji z wyjścia komunikatu kanału.

Wyjście kanału może zmieniać pola w MQCD. Zmiany te nie są jednak wprowadzane, z wyjątkiem wymienionych okoliczności. Jeśli program obsługi wyjścia kanału zmieni pole w strukturze danych MQCD, nowa wartość zostanie zignorowana przez proces kanału IBM MQ . Jednak nowa wartość pozostaje w MQCD i jest przekazywana do wszystkich pozostałych wyjść w łańcuchu wyjścia i do każdej konwersacji współużytkującej instancję kanału. Więcej informacji na ten temat zawiera sekcja Zmiana pól MQCD w wyjściu kanału .

Ponadto, w przypadku programów napisanych w języku C, w programie obsługi wyjścia kanału nie może być używana niewielobieżna funkcja biblioteki języka C.

Linux AIX Jeśli jednocześnie używane są biblioteki wyjścia wielu kanałów, mogą wystąpić problemy na niektórych platformach UNIX and Linux , jeśli kod dwóch różnych wyjść zawiera funkcje o takich samych nazwach. Po załadowaniu wyjścia kanału dynamiczny program ładujący tłumaczy nazwy funkcji w bibliotece wyjścia na adresy, do których biblioteka jest załadowana. Jeśli dwie biblioteki wyjścia definiują oddzielne funkcje, które mają identyczne nazwy, ten proces rozstrzygania może niepoprawnie rozstrzygnąć nazwy funkcji jednej biblioteki w celu użycia funkcji innej biblioteki. Jeśli wystąpi ten problem, należy określić dla konsolidatora, że ma on eksportować tylko wymagane funkcje wyjścia i MQStart, ponieważ nie ma to wpływu na te funkcje. Inne funkcje muszą mieć widoczność lokalną, aby nie były używane przez funkcje spoza własnej biblioteki wyjścia. Więcej informacji można znaleźć w dokumentacji konsolidatora.

Wszystkie wyjścia są wywoływane ze strukturą parametru wyjścia kanału (MQCXP), strukturą definicji kanału (MQCD), przygotowanym buforem danych, parametrem długości danych i parametrem długości buforu. Długość buforu nie może być większa niż:

- W przypadku wyjść komunikatów należy zezwolić na wysyłanie przez kanał największego wymaganego komunikatu, powiększonego o długość struktury MQXQH.
- W przypadku wyjść wysyłania i odbierania największy bufor, na który należy zezwolić, jest następujący:

  **LU 6.2**
  32 kB

  **TCP:**

  IBM i IBM i 16 kB

  IBM i Inne 32 kB

  **Uwaga:** Maksymalna użyteczna długość może być o 2 bajty mniejsza od tej długości. Sprawdź wartość zwróconą w polu MaxSegment, aby uzyskać szczegółowe informacje. Więcej informacji na temat parametru MaxSegmentLength zawiera sekcja MaxSegmentLength.

  **NetBIOS:**
  64 kB

  **SPX:**
  64 kB

**Uwaga:** Wyjścia odbierania w kanałach nadawczych i wyjścia nadawcze w kanałach odbiorczych używają buforów 2 kB dla protokołu TCP.

- W przypadku wyjść zabezpieczeń rozproszona funkcja kolejkowania przydziela bufor o wielkości 4000 bajtów.

Wyjście może zwrócić alternatywny bufor wraz z odpowiednimi parametrami. Szczegółowe informacje na ten temat zawiera sekcja “Kanał-programy obsługi wyjścia dla kanałów przesyłania komunikatów” na stronie 990 .

z/OS *Pisanie programów obsługi wyjścia kanału w systemie z/OS*

Poniższe informacje mogą być pomocne podczas pisania i kompilowania programów obsługi wyjścia kanału dla systemu z/OS.

Wyjścia są uruchamiane tak, jak za pomocą z/OS LINK, w:

- Nieautoryzowany stan programu problemu
- Tryb sterowania podstawową przestrzenią adresową
- Tryb inny niż międzypamięciowy
- Tryb rejestru bez dostępu
- 31-bitowy tryb adresowania

Moduły poddane konsolidacji muszą być umieszczone w zestawie danych określonym przez instrukcję CSQXLIB DD w procedurze przestrzeni adresowej inicjatora kanału; nazwy modułów ładowania są określone jako nazwy wyjść w definicji kanału.

Podczas zapisywania wyjść kanału dla produktu z/OSobowiązują następujące reguły:

- Wyjścia muszą być napisane w języku asembler lub C; jeśli używany jest C, musi być on zgodny ze środowiskiem programowania systemowego C dla wyjść systemowych, opisanym w podręczniku z/OS C/C++ Programming Guide.
- Wyjścia są ładowane z nieautoryzowanych bibliotek zdefiniowanych przez instrukcję DD CSQXLIB. Jeśli CSQXLIB ma wartość DISP=SHR, wyjścia mogą być aktualizowane podczas działania inicjatora kanału. Nowa wersja jest używana po zrestartowaniu kanału.
- Wyjścia muszą być wielobieżne i mogą działać w dowolnym miejscu wirtualnej pamięci masowej.
- Wyjścia muszą resetować środowisko, po powrocie, do tego w momencie wejścia.
- Wyjścia muszą zwolnić każdą uzyskaną pamięć lub zapewnić, że zostanie ona zwolniona przez kolejne wywołanie wyjścia.

  W przypadku pamięci, która ma być trwała między wywołaniami, należy użyć usługi z/OS STORAGE lub funkcji bibliotecznej 4kmalc dla programowania systemowego C.

  Więcej informacji na temat tej funkcji zawiera sekcja 4kmalc() -- Allocate Page-Aligned Storage(Pamięć wyrównana do strony).
- Można używać wszystkich wywołań MQI produktu IBM MQ z wyjątkiem MQCMIT lub CSQBCMT i MQBACK lub CSQBBAK. Muszą one być zawarte po MQCONN (z pustą nazwą menedżera kolejek). Jeśli te wywołania są używane, wyjście musi być połączone z kodem pośredniczącym CSQXSTUB.

  Wyjątkiem od tej reguły jest sytuacja, w której wyjścia kanału zabezpieczeń mogą wydawać wywołania MQI zatwierdzania i wycofywania. Aby wywołać takie wywołania, należy zakodować komendy CSQXCMT i CSQXBAK zamiast MQCMIT lub CSQBCMT i MQBACK lub CSQBBAK.
- Wszystkie wyjścia używające kodu pośredniczącego CSQXSTUB z produktu IBM WebSphere MQ 7.0 lub nowszego muszą być połączone w bibliotece dynamicznej CSQXLIB w formacie PDS-E.
- Wyjścia nie mogą używać żadnych usług systemowych, które powodują oczekiwanie, ponieważ użycie usług systemowych poważnie wpłynęłoby na obsługę niektórych lub wszystkich innych kanałów. Wiele kanałów jest zwykle uruchamianych przez pojedynczą bazę TCB. Jeśli w wyjściu zostanie użyty element powodujący oczekiwanie i nie zostanie użyty parametr MQXWAIT, spowoduje to, że wszystkie te kanały będą oczekiwać. Spowodowanie oczekiwania kanałów nie powoduje żadnych problemów

funkcjonalnych, ale może mieć negatywny wpływ na wydajność. Większość kanałów SVC wymaga oczekiwania, dlatego należy ich unikać, z wyjątkiem następujących kanałów SVC:

  – GETMAIN/FREEMAIN/STORAGE,
  – ŁADOWANIE/USUWANIE

  Z tego względu należy unikać kanałów SVC, komputerów PC i urządzeń we/wy. Zamiast tego należy użyć wywołania MQXWAIT.

- Wyjścia nie wywołują ESTAEs ani SPIEs, poza podzadaniami, które przyłączają, ponieważ ich obsługa błędów może kolidować z obsługą błędów wykonywaną przez IBM MQ. Oznacza to, że program IBM MQ może nie być w stanie naprawić błędu lub że program obsługi wyjścia może nie otrzymać wszystkich informacji o błędzie.
- Wywołanie MQXWAIT (patrz MQXWAIT ) Udostępnia usługę oczekiwania, która oczekuje na zdarzenia we/wy i inne zdarzenia. Jeśli ta usługa jest używana, wyjścia nie mogą używać stosu powiązań.

  W przypadku urządzeń we/wy i innych urządzeń, które nie udostępniają urządzeń nieblokujących lub EBC, na które należy czekać, należy wykonać osobne podzadanie ATTACHed i oczekiwać na jego zakończenie MQXWAIT; ze względu na przetwarzanie, do którego ta technika się stosuje, to urządzenie musi być używane tylko przez wyjście zabezpieczeń.

- Wywołanie MQDISC MQI nie powoduje niejawnego zatwierdzenia w programie obsługi wyjścia. Zatwierdzenie procesu kanału jest wykonywane tylko wtedy, gdy wymaga tego protokół kanału.

Następujące przykłady wyjścia są dostarczane z produktem IBM MQ for z/OS:

**CSQ4BAX0**
Ten przykład został napisany w asemblerze i ilustruje użycie komendy MQXWAIT.

**CSQ4BCX1 i CSQ4BCX2**
Przykłady te zostały napisane w języku C i ilustrują sposób uzyskiwania dostępu do parametrów.

**CSQ4BCX3 i CSQ4BAX3**
Przykłady te zostały napisane odpowiednio w języku C i w języku asembler.

Przykład CSQ4BCX3 (który jest prekompilowany do SCSQAUTH LOADLIB, powinien działać bez konieczności wprowadzania zmian w samym wyjściu. Można utworzyć bibliotekę LOADLIB (na przykład MY.TEST.LOADLIB) i skopiuj do niego podzbiór SCSQAUTH (CSQ4BCX3).

Aby skonfigurować wyjście zabezpieczeń dla połączenia klienta, wykonaj następującą procedurę:

1. Określ poprawny segment OMVS dla identyfikatora użytkownika używanego przez inicjator kanału.

   Dzięki temu inicjator kanału IBM MQ for z/OS może używać protokołu TCP/IP z interfejsem gniazda z/OS UNIX System Services (z/OS UNIX) w celu ułatwienia przetwarzania wyjścia. Należy zauważyć, że nie jest konieczne definiowanie segmentu OMVS dla identyfikatora użytkownika łączącego się klienta.
2. Upewnij się, że kod wyjścia działa tylko w środowisku sterowanym przez program.

   Oznacza to, że wszystkie elementy załadowane do przestrzeni adresowej CHINIT muszą być załadowane z biblioteki sterowanej przez program (co oznacza wszystkie biblioteki w bibliotece STEPLIB) oraz z wszystkich bibliotek wymienionych w bibliotece CSQXLIB i

   ```
   ++hlq++.SCSQANLx
   ++hlq++.SCSQMVR1
   ++hlq++.SCSQAUTH
   ```

   Aby ustawić bibliotekę ładowania jako sterowaną przez program, należy użyć komendy podobnej do poniższej:

   ```
   RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
   ```

   Następnie można aktywować lub odświeżać środowisko sterowane przez program, wydając komendę:

```
SETROPTS WHEN(PROGRAM) REFRESH
```

3. Dodaj wyjście LOADLIB do definicji danych CSQXLIB (w uruchomionej procedurze CHINIT), wydając następującą komendę:

```
ALTER CHANNEL(xxxx) CHLTYPE(SVRCONN)SCYEXIT(CSQ4BCX3)
```

   Spowoduje to aktywowanie wyjścia dla nazwanego kanału.

4. Zewnętrzny menedżer bezpieczeństwa (External Security Manager-ESM) wyświetla wszystkie inne biblioteki, które mają być kontrolowane przez program, ale należy pamiętać, że żadna z bibliotek ESM lub C nie musi być pod kontrolą programu.

Więcej informacji na temat konfigurowania wyjścia zabezpieczeń przy użyciu przykładowego CSQ4BCX3zawiera sekcja Kanał połączenia z serweremIBM MQ for z/OS .

**CSQ4BCX4**

Ten przykład został napisany w języku C i demonstruje użycie pól **RemoteProduct** i **RemoteVersion** w produkcie MQCXP.

**Pojęcia pokrewne**

“Pisanie programów obsługi wyjścia kanału w systemie IBM i” na stronie 997
Poniższe informacje mogą być pomocne podczas pisania i kompilowania programów obsługi wyjścia kanału dla systemu IBM i.

“Pisanie programów obsługi wyjścia kanału w systemie AIX, Linux, and Windows” na stronie 998
Poniższe informacje mogą być pomocne podczas pisania programów obsługi wyjścia kanału dla systemów AIX, Linux, and Windows .

**Odsyłacze pokrewne**

IBM MQ for z/OS Kanał połączenia z serwerem

## IBM i *Pisanie programów obsługi wyjścia kanału w systemie IBM i*

Poniższe informacje mogą być pomocne podczas pisania i kompilowania programów obsługi wyjścia kanału dla systemu IBM i.

Wyjście jest obiektem programu napisanym w języku ILE C, ILE RPG lub ILE COBOL. Nazwy programów obsługi wyjścia i ich biblioteki są wymienione w definicji kanału.

Podczas tworzenia i kompilowania programu obsługi wyjścia należy przestrzegać następujących warunków:

- Program musi być wątkowo bezpieczny i utworzony za pomocą kompilatora ILE C, ILE RPG lub ILE COBOL. W przypadku języka ILE RPG należy określić specyfikację sterującą THREAD (*SERIALIZE), a w przypadku języka ILE COBOL należy określić opcję SERIALIZE dla opcji THREAD instrukcji PROCESS. Programy muszą być również powiązane z wątkowymi bibliotekami IBM MQ : QMQM/LIBMQM_R w przypadku języka ILE C i ILE RPG oraz AMQ0STUB_R w przypadku języka ILE COBOL. Dodatkowe informacje na temat zabezpieczania wątków aplikacji RPG lub COBOL zawiera podręcznik programisty dla danego języka.
- System IBM MQ for IBM i wymaga, aby programy obsługi wyjścia obsługiwały teraprzestrzeń. Teraprzestrzeń jest formą pamięci współużytkowanej wprowadzoną w systemie OS/400 V4R4. W przypadku kompilatorów ILE RPG i COBOL wszystkie programy skompilowane w systemie OS/400 V4R4 lub nowszym są tak włączone. W przypadku języka C programy muszą być skompilowane z opcjami TERASPACE (*YES *TSIFC) określonymi w komendach CRTCMOD lub CRTBNDC.
- Wyjście zwracające wskaźnik do własnego obszaru buforu musi zapewnić, że wskazywany obiekt istnieje poza zakresem czasu programu obsługi wyjścia kanału. Wskaźnik nie może być adresem zmiennej w stosie programów ani zmiennej w stercie programów. Zamiast tego należy uzyskać wskaźnik z systemu. Przykładem jest przestrzeń użytkownika utworzona w programie użytkownika. Aby zapewnić, że obszar danych przydzielony przez program obsługi wyjścia kanału jest nadal dostępny dla agenta MCA po zakończeniu programu, program obsługi wyjścia kanału musi działać w grupie aktywacji

programu wywołującego lub w nazwanej grupie aktywacji. W tym celu należy ustawić parametr ACTGRP komendy CRTPGM na wartość zdefiniowaną przez użytkownika lub wartość *CALLER. Jeśli program jest tworzony w ten sposób, program obsługi wyjścia kanału może przydzielić pamięć dynamiczną i przekazać wskaźnik do tej pamięci z powrotem do agenta MCA.

**Pojęcia pokrewne**

“Pisanie programów obsługi wyjścia kanału w systemie AIX, Linux, and Windows” na stronie 998
Poniższe informacje mogą być pomocne podczas pisania programów obsługi wyjścia kanału dla systemów AIX, Linux, and Windows .

“Pisanie programów obsługi wyjścia kanału w systemie z/OS” na stronie 995
Poniższe informacje mogą być pomocne podczas pisania i kompilowania programów obsługi wyjścia kanału dla systemu z/OS.

ALW *Pisanie programów obsługi wyjścia kanału w systemie AIX, Linux, and Windows*

Poniższe informacje mogą być pomocne podczas pisania programów obsługi wyjścia kanału dla systemów AIX, Linux, and Windows .

Należy postępować zgodnie z instrukcjami przedstawionymi w sekcji “Zapisywanie wyjść i usług instalowalnych w systemie AIX, Linux, and Windows” na stronie 965. W razie potrzeby użyj następujących informacji dotyczących wyjścia kanału:

Wyjście musi być napisane w języku C i jest biblioteką DLL w systemie Windows.

Zdefiniuj fikcyjną podprogram MQStart () w wyjściu i określ `MQStart` jako punkt wejścia w bibliotece. Rysunek 106 na stronie 998 przedstawia sposób konfigurowania pozycji dla programu:

```
#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
}
```

*Rysunek 106. Przykładowy kod źródłowy wyjścia kanału*

Podczas zapisywania wyjść kanału dla produktu Windows za pomocą programu Visual C + + należy zapisać własny plik DEF . Przykład ilustrujący sposób przedstawiony na rysunku ( Rysunek 107 na stronie 998). Więcej informacji na temat pisania programów obsługi wyjścia kanału zawiera sekcja “Zapisywanie programów obsługi wyjścia kanału” na stronie 993.

```
EXPORTS
ChannelExit
```

*Rysunek 107. Przykładowy plik DEF dla Windows*

**Pojęcia pokrewne**

“Pisanie programów obsługi wyjścia kanału w systemie IBM i” na stronie 997
Poniższe informacje mogą być pomocne podczas pisania i kompilowania programów obsługi wyjścia kanału dla systemu IBM i.

“Pisanie programów obsługi wyjścia kanału w systemie z/OS” na stronie 995
Poniższe informacje mogą być pomocne podczas pisania i kompilowania programów obsługi wyjścia kanału dla systemu z/OS.

*Programy obsługi wyjścia zabezpieczeń kanału*
Za pomocą programów obsługi wyjścia zabezpieczeń można sprawdzić, czy partner na drugim końcu kanału jest autentyczny. Jest to nazywane uwierzytelnianiem.

Aby określić, że kanał musi używać wyjścia zabezpieczeń, podaj nazwę wyjścia w polu **SCYEXIT** definicji kanału.

**Uwaga:** Uwierzytelnianie można również wykonać przy użyciu rekordów uwierzytelniania kanału. Rekordy uwierzytelniania kanału zapewniają dużą elastyczność w uniemożliwianiu dostępu do menedżerów kolejek niektórym użytkownikom i kanałom oraz w odwzorowywaniu użytkowników zdalnych na identyfikatory użytkowników programu IBM MQ . Obsługa protokołu TLS jest również udostępniana przez produkt IBM MQ w celu uwierzytelniania użytkowników oraz w celu zapewnienia szyfrowania i sprawdzania integralności danych. Więcej informacji na temat protokołu TLS zawiera sekcja Protokoły zabezpieczeń TLS w produkcie IBM MQ. Jeśli jednak nadal wymagane są bardziej zaawansowane (lub różne) formy przetwarzania zabezpieczeń oraz inne rodzaje kontroli i ustanawiania kontekstu zabezpieczeń, należy rozważyć napisanie wyjść zabezpieczeń.

Atrybuty Nazwa wyróżniająca podmiotu i wystawcy są wyświetlane w następujących atrybutach statusu kanału:

- SSLPEER (selektor PCF MQCACH_SSL_SHORT_PEER_NAME)
- SSLCERTI (selektor PCF MQCACH_SSL_CERT_ISSUER_NAME)

Wartości te są zwracane przez komendy statusu kanału oraz dane przekazywane do wymienionych wyjść zabezpieczeń kanału, jak pokazano poniżej:

- MQCD SSLPeerNamePtr
- MQCXP SSLRemCertIssNamePtr

Wyjście zabezpieczeń można zapisać w języku C lub Java.

Programy obsługi wyjścia zabezpieczeń kanału są wywoływane w następujących miejscach cyklu przetwarzania agenta MCA:

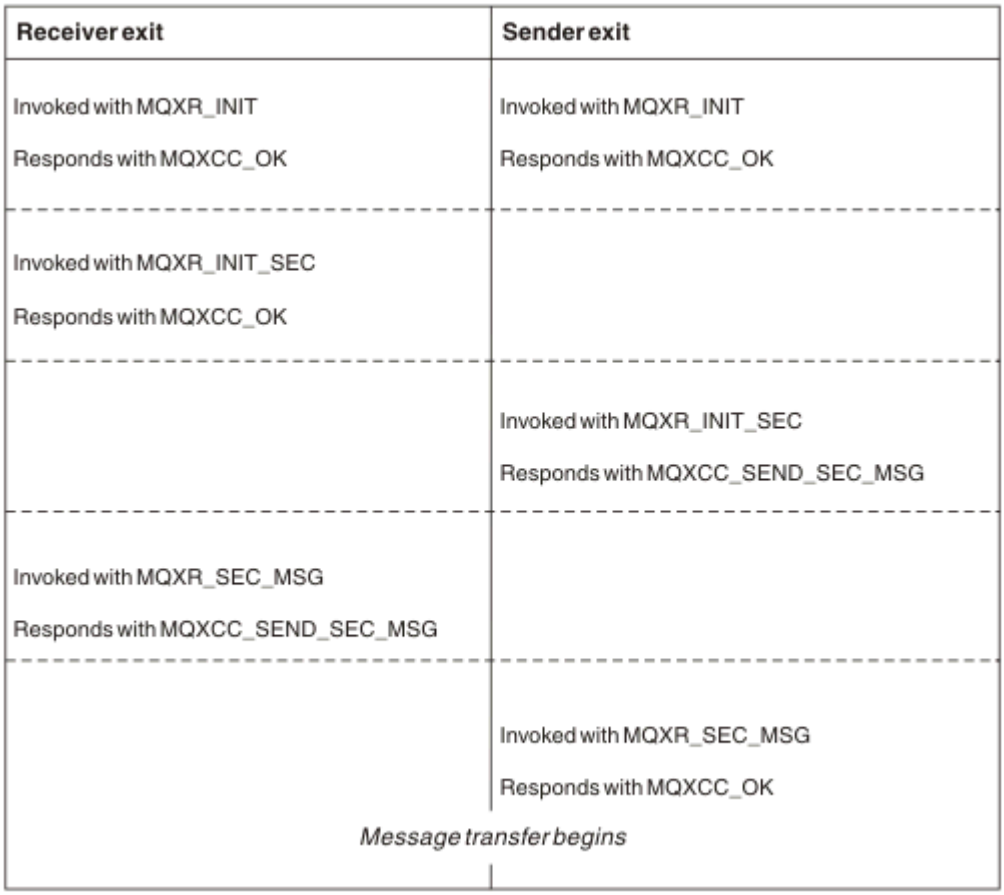
- Przy inicjacji i zakończeniu MCA.
- Bezpośrednio po zakończeniu początkowej negocjacji danych podczas uruchamiania kanału. Odbiornik lub serwer końcowy kanału może zainicjować wymianę komunikatów bezpieczeństwa ze zdalnym końcem, udostępniając komunikat, który ma zostać dostarczony do wyjścia zabezpieczeń na zdalnym końcu. Może również odmówić wykonania tej czynności. Program obsługi wyjścia jest uruchamiany ponownie w celu przetworzenia dowolnego komunikatu ochrony odebranego ze zdalnego zakończenia.
- Bezpośrednio po zakończeniu początkowej negocjacji danych podczas uruchamiania kanału. Wysyłający lub żądający koniec kanału przetwarza komunikat zabezpieczeń odebrany ze zdalnego zakończenia lub inicjuje wymianę zabezpieczeń, gdy zdalny koniec nie może. Program obsługi wyjścia jest uruchamiany ponownie w celu przetworzenia wszystkich kolejnych komunikatów ochrony, które mogą zostać odebrane.

Kanał requestera nigdy nie jest wywoływany z MQXR_INIT_SEC. Kanał powiadamia serwer, że ma program obsługi wyjścia zabezpieczeń, a następnie serwer ma możliwość zainicjowania wyjścia zabezpieczeń. Jeśli go nie ma, informuje o tym requester i do programu obsługi wyjścia zwracany jest przepływ o zerowej długości.

**Uwaga:** Należy unikać wysyłania komunikatów bezpieczeństwa o zerowej długości.

Przykłady danych wymienianych przez programy obsługi wyjścia zabezpieczeń przedstawiono na rysunkach od Rysunek 108 na stronie 1000 do Rysunek 111 na stronie 1002. W tych przykładach przedstawiono sekwencję zdarzeń, które wystąpiły z użyciem wyjścia zabezpieczeń odbiornika i wyjścia zabezpieczeń nadawcy. Kolejne rzędy w liczbach reprezentują upływ czasu. W niektórych przypadkach zdarzenia w odbiorniku i nadawcy nie są skorelowane i dlatego mogą występować w tym samym czasie lub w różnym czasie. W innych przypadkach zdarzenie w jednym programie obsługi wyjścia powoduje wystąpienie zdarzenia uzupełniającego w drugim programie obsługi wyjścia. Na przykład w pliku Rysunek 108 na stronie 1000:

1. Każdy odbiorca i nadawca jest wywoływany za pomocą komendy MQXR_INIT, ale te wywołania nie są skorelowane i mogą wystąpić w tym samym czasie lub w innym czasie.
2. Odbiornik jest następnie wywoływany za pomocą komendy MQXR_INIT_SEC, ale zwraca komunikat MQXCC_OK, który nie wymaga dodatkowego zdarzenia przy wyjściu nadawcy.
3. Nadawca jest następnie wywoływany z MQXR_INIT_SEC. Nie jest to skorelowane z wywołaniem odbiornika z MQXR_INIT_SEC. Nadawca zwraca komunikat MQXCC_SEND_SEC_MSG, który powoduje wystąpienie zdarzenia uzupełniającego w wyjściu odbiornika.
4. Następnie odbiornik jest wywoływany z MQXR_SEC_MSG i zwraca MQXCC_SEND_SEC_MSG, co powoduje wystąpienie zdarzenia uzupełniającego w wyjściu nadawcy.
5. Następnie nadawca jest wywoływany za pomocą wywołania MQXR_SEC_MSG i zwraca komunikat MQXCC_OK, który nie wymaga dodatkowego zdarzenia w wyjściu odbiorcy.




*Rysunek 108. Wymiana inicjowana przez nadawcę z umową*

| Receiver exit | Sender exit |
|---|---|
| Invoked with MQXR_INIT<br>Responds with MQXCC_OK | Invoked with MQXR_INIT<br>Responds with MQXCC_OK |
| Invoked with MQXR_INIT_SEC<br>Responds with MQXCC_OK | |
| | Invoked with MQXR_INIT_SEC<br>Responds with MQXCC_SEND_SEC_MSG |
| Invoked with MQXR_SEC_MSG<br>Responds with MQXCC_OK | |
| | Invoked with MQXR_SEC_MSG<br>Responds with MQXCC_SUPPRESS_FUNCTION |
| *Channel closes* | |
| Invoked with MQXR_TERM<br>Responds with MQXCC_OK | Invoked with MQXR_TERM<br>Responds with MQXCC_OK |

*Rysunek 109. Wymiana inicjowana przez nadawcę bez umowy*

| Receiver exit | Sender exit |
|---|---|
| Invoked with MQXR_INIT<br>Responds with MQXCC_OK | Invoked with MQXR_INIT<br>Responds with MQXCC_OK |
| Invoked with MQXR_INIT_SEC<br>Responds with MQXCC_SEND_SEC_MSG | |
| | Invoked with MQXR_SEC_MSG<br>Responds with MQXCC_SEND_SEC_MSG |
| Invoked with MQXR_SEC_MSG<br>Responds with MQXCC_OK | |
| *Message transfer begins* | |
| Invoked with MQXR_TERM<br>Responds with MQXCC_OK | Invoked with MQXR_TERM<br>Responds with MQXCC_OK |

*Rysunek 110. Wymiana inicjowana przez odbiorcę z umową*

| Receiver exit | Sender exit |
|---|---|
| Invoked with MQXR_INIT<br>Responds with MQXCC_OK | Invoked with MQXR_INIT<br>Responds with MQXCC_OK |
| Invoked with MQXR_INIT_SEC<br>Responds with MQXCC_SEND_SEC_MSG | |
| | Invoked with MQXR_SEC_MSG<br>Responds with MQXCC_OK |
| Invoked with MQXR_SEC_MSG<br>Responds with MQXCC_SUPPRESS_FUNCTION | |
| *Channel closes* | |

*Rysunek 111. Wymiana inicjowana przez odbiorcę bez umowy*

Do programu obsługi wyjścia zabezpieczeń kanału przekazywany jest bufor agenta zawierający dane zabezpieczeń, z wyłączeniem nagłówków transmisji wygenerowanych przez wyjście zabezpieczeń. Dane te mogą być dowolnymi odpowiednimi danymi, tak aby każdy koniec kanału mógł wykonać sprawdzanie poprawności zabezpieczeń.

Program obsługi wyjścia zabezpieczeń na wysyłającym i odbierającym końcu kanału komunikatów może zwrócić jeden z dwóch kodów odpowiedzi do dowolnego wywołania:

- Wymiana zabezpieczeń została zakończona bez błędów
- Zablokuj kanał i zamknij

**Uwaga:**

1. Wyjścia zabezpieczeń kanału zwykle działają w parach. Podczas definiowania odpowiednich kanałów należy upewnić się, że kompatybilne programy obsługi wyjścia są nazwane dla obu końców kanału.
2. IBM i W systemie IBM i programy obsługi wyjścia zabezpieczeń, które zostały skompilowane przy użyciu parametru `Use adopted authority` (USEADPAUT = *YES), mogą adoptować uprawnienie QMQM lub QMQMADM. Należy zachować ostrożność, aby wyjście nie używała tej opcji w celu zapewnienia bezpieczeństwa systemu.
3. W przypadku kanału TLS, w którym drugi koniec kanału udostępnia certyfikat, wyjście zabezpieczeń otrzymuje nazwę wyróżniającą podmiotu tego certyfikatu w polu MQCD, do którego dostęp ma atrybut SSLPeerNamePtr, oraz nazwę wyróżniającą wystawcy w polu MQCXP, do którego dostęp ma atrybut SSLRemCertIssNamePtr. Używa, do którego można umieścić tę nazwę:
   - Ograniczenie dostępu przez kanał TLS.
   - Aby zmienić MQCD.MCAUserIdentifier na podstawie nazwy.

**Pojęcia pokrewne**

Pojęcia związane z protokołem TLS (Transport Layer Security)

**Odsyłacze pokrewne**

Rekordy uwierzytelniania kanału

*Zapisywanie wyjścia zabezpieczeń*

Wyjście zabezpieczeń można zapisać za pomocą kodu szkieletu wyjścia zabezpieczeń.

Rysunek 112 na stronie 1003 ilustruje sposób zapisu wyjścia zabezpieczeń.

```
void MQENTRY MQStart() {;}
void MQENTRY EntryPoint (PMQVOID pChannelExitParms,
                         PMQVOID pChannelDefinition,
                         PMQLONG pDataLength,
                         PMQLONG pAgentBufferLength,
                         PMQVOID pAgentBuffer,
                         PMQLONG pExitBufferLength,
                         PMQPTR  pExitBufferAddr)
{
  PMQCXP pParms = (PMQCXP)pChannelExitParms;
  PMQCD  pChDef = (PMQCD)pChannelDefinition;
  /* TODO: Add Security Exit Code Here */

}
```

*Rysunek 112. Kod szkieletu wyjścia zabezpieczeń*

Standardowy punkt wejścia IBM MQ MQStart musi istnieć, ale nie jest wymagany do wykonywania żadnych funkcji. Nazwa funkcji (w tym przykładzieEntryPoint ) może zostać zmieniona, ale funkcja musi zostać wyeksportowana podczas kompilowania i konsolidowania biblioteki. Podobnie jak w poprzednim przykładzie, wskaźniki pChannelExitParms muszą być rzutowane na PMQCXP, a definicja pChannelmusi być rzutowana na PMQCD. Ogólne informacje na temat wywoływania wyjść kanałów i używania parametrów zawiera sekcja MQ_CHANNEL_EXIT. Te parametry są używane w wyjściu zabezpieczeń w następujący sposób:

**PMQVOID pChannelExitParms**
Wejście/wyjście

Wskaźnik do struktury MQCXP-rzutowanie na PMQCXP w celu uzyskania dostępu do pól. Ta struktura jest używana do komunikacji między wyjściem i agentem MCA. Następujące pola w MQCXP są szczególnie interesujące dla wyjść zabezpieczeń:

**ExitReason**
Informuje program Security o wyjściu z bieżącego stanu wymiany zabezpieczeń i jest używany przy podejmowaniu decyzji o podjęciu działań.

**ExitResponse**
Odpowiedź dla MCA, która dyktuje następny etap w wymianie zabezpieczeń.

**ExitResponse2**
Dodatkowe flagi sterujące służące do zarządzania interpretacją odpowiedzi wyjścia zabezpieczeń przez agent MCA.

**Obszar ExitUser**
16 bajtów (maksymalnie) pamięci, która może być używana przez wyjście zabezpieczeń do utrzymywania stanu między wywołaniami.

**ExitData**
Zawiera dane określone w polu SCYDATA definicji kanału (32 bajty dopełnione z prawej strony spacjami).

**Definicja atrybutu PMQVOID pChannel**
Wejście/wyjście

Wskaźnik do struktury MQCD-rzutowanie na PMQCD w celu uzyskania dostępu do pól. Ten parametr zawiera definicję kanału. Następujące pola w tabeli MQCD są szczególnie interesujące dla wyjść zabezpieczeń:

**ChannelName**
Nazwa kanału (20 bajtów dopełniona po prawej stronie spacjami).

**ChannelType**
Kod definiujący typ kanału.

**Identyfikator użytkownika MCA**
Ta grupa trzech pól jest inicjowana wartością pola MCAUSER określonego w definicji kanału. Identyfikator użytkownika określony przez wyjście zabezpieczeń w tych polach jest używany na potrzeby kontroli dostępu (nie dotyczy kanałów SDR, SVR, CLNTCONN ani CLUSSDR).

**MCAUserIdentifier**
Pierwsze 12 bajtów identyfikatora dopełnione po prawej stronie spacjami.

**LongMCAUserIdPtr**
Wskaźnik do buforu zawierającego identyfikator o pełnej długości (nie gwarantowany zerowy koniec) ma wyższy priorytet niż MCAUserIdentifier.

**LongMCAUserIdLength**
Długość łańcucha wskazywanego przez parametr LongMCAUserIdPtr -musi być ustawiona, jeśli ustawiona jest wartość LongMCAUserIdPtr .

**Identyfikator użytkownika zdalnego**
Dotyczy tylko par kanałów CLNTCONN/SVRCONN. Jeśli nie zdefiniowano żadnego programu zewnętrznego zabezpieczeń CLNTCONN, te trzy pola są inicjowane przez agent MCA klienta, dlatego mogą zawierać identyfikator użytkownika ze środowiska klienta, który może być używany przez program zewnętrzny zabezpieczeń SVRCONN na potrzeby uwierzytelniania oraz podczas określania identyfikatora użytkownika MCA. Jeśli zdefiniowano wyjście zabezpieczeń CLNTCONN, pola te nie są inicjowane i mogą być ustawiane przez wyjście zabezpieczeń CLNTCONN lub komunikaty bezpieczeństwa mogą być używane do przekazywania identyfikatora użytkownika z klienta do serwera.

**Identyfikator RemoteUser**
Pierwsze 12 bajtów identyfikatora dopełnione spacjami po prawej stronie.

**LongRemoteUserIdPtr**
Wskaźnik do buforu zawierającego identyfikator pełnej długości (nie gwarantowany zerowy koniec) ma wyższy priorytet niż identyfikator RemoteUser.

**LongRemoteUserIdDługość**
Długość łańcucha wskazywanego przez atrybut LongRemoteUserIdPtr-musi być ustawiona, jeśli atrybut LongRemoteUserIdPtr jest ustawiony.

**Długość PMQLONG pData**
Wejście/wyjście

Wskaźnik do MQLONG. Zawiera długość dowolnego wyjścia zabezpieczeń zawartego w elemencie AgentBuffer po wywołaniu wyjścia zabezpieczeń. Musi być ustawiona przez wyjście zabezpieczeń na długość każdego komunikatu wysyłanego w buforze AgentBuffer lub ExitBuffer.

**PMQLONG pAgentBufferLength**
dane wejściowe

Wskaźnik do MQLONG. Długość danych zawartych w elemencie AgentBuffer przy wywołaniu wyjścia zabezpieczeń.

**Bufor PMQVOID pAgent**
Wejście/wyjście

Po wywołaniu wyjścia zabezpieczeń wskazuje to na dowolny komunikat wysłany z wyjścia partnerskiego. Jeśli opcja ExitResponse2 w strukturze MQCXP ma ustawioną opcję MQXR2_USE_AGENT_BUFFER (domyślnie), to wyjście zabezpieczeń musi ustawić ten parametr tak, aby wskazywał na wysyłane dane komunikatu.

**PMQLONG pExitBufferLength**
Wejście/wyjście

Wskaźnik do MQLONG. Ten parametr jest inicjowany na wartość 0 przy pierwszym wywołaniu wyjścia zabezpieczeń, a zwracana wartość jest zachowywana między wywołaniami wyjścia zabezpieczeń podczas wymiany zabezpieczeń.

**PMQPTR pExitBufferAddr**
Wejście/wyjście

Ten parametr jest inicjowany jako wskaźnik pusty przy pierwszym wywołaniu wyjścia zabezpieczeń, a zwracana wartość jest zachowywana między wywołaniami wyjścia zabezpieczeń podczas wymiany zabezpieczeń. Jeśli opcja MQXR2_USE_EXIT_BUFFER jest ustawiona w opcji ExitResponse2 w strukturze MQCXP, to wyjście zabezpieczeń musi ustawić ten parametr w taki sposób, aby wskazywał na wysyłane dane komunikatu.

*Różnice w zachowaniu między wyjściami zabezpieczeń zdefiniowanymi dla par kanałów CLNTCONN/SVRCONN i innych par kanałów*
Wyjścia zabezpieczeń można zdefiniować dla wszystkich typów kanałów. Jednak zachowanie wyjść zabezpieczeń zdefiniowanych w parach kanałów CLNTCONN/SVRCONN różni się nieznacznie od wyjść zabezpieczeń zdefiniowanych w innych parach kanałów.

Wyjście zabezpieczeń w kanale CLNTCONN może ustawić identyfikator zdalnego użytkownika w definicji kanału na potrzeby przetwarzania przez partnerskie wyjście SVRCONN lub autoryzację OAM, jeśli nie zdefiniowano żadnego wyjścia zabezpieczeń SVRCONN i nie ustawiono pola MCAUSER SVRCONN.

Jeśli nie zdefiniowano żadnego wyjścia zabezpieczeń CLNTCONN, identyfikator użytkownika zdalnego w definicji kanału jest ustawiany na identyfikator użytkownika ze środowiska klienta (który może być pusty) przez agent MCA klienta.

Wymiana zabezpieczeń między wyjściami zabezpieczeń zdefiniowanymi w parze kanałów CLNTCONN i SVRCONN kończy się pomyślnie, gdy wyjście zabezpieczeń SVRCONN zwraca odpowiedź ExitResponse MQXCC_OK. Wymiana zabezpieczeń między innymi parami kanałów kończy się pomyślnie, gdy wyjście zabezpieczeń, które zainicjowało wymianę, zwraca odpowiedź ExitResponse z komunikatu MQXCC_OK.

Jednak kod ExitResponse komendy MQXCC_SEND_AND_REQUEST_SEC_MSG może zostać użyty w celu wymuszenia kontynuacji wymiany zabezpieczeń: jeśli odpowiedź ExitResponse komendy MQXCC_SEND_AND_REQUEST_SEC_MSG zostanie zwrócona przez wyjście zabezpieczeń CLNTCONN lub SVRCONN, wyjście partnera musi odpowiedzieć, wysyłając komunikat bezpieczeństwa (nie MQXCC_OK lub odpowiedź o wartości NULL) lub kanał zostanie zakończony. W przypadku wyjść zabezpieczeń zdefiniowanych dla innych typów kanału, odpowiedź ExitResponse MQXCC_OK zwrócona w odpowiedzi na MQXCC_SEND_AND_REQUEST_SEC_MSG z wyjścia zabezpieczeń partnera powoduje kontynuację wymiany zabezpieczeń, tak jakby została zwrócona odpowiedź pusta, a nie została zakończona.

*Wyjście zabezpieczeń SPIP*
Produkt IBM MQ for Windows dostarcza wyjście zabezpieczeń, które udostępnia uwierzytelnianie dla kanałów systemu IBM MQ za pomocą interfejsu SSPI (Security Services Programming Interface). Interfejs SSPI udostępnia zintegrowane narzędzia bezpieczeństwa systemu Windows.

To wyjście zabezpieczeń dotyczy zarówno klienta IBM MQ , jak i serwera IBM MQ .

Pakiety zabezpieczeń są ładowane z biblioteki security.dll lub secur32.dll. Te biblioteki DLL są dostarczane z systemem operacyjnym.

Uwierzytelnianie jednokierunkowe jest udostępniane w systemie Windowsprzy użyciu usług uwierzytelniania NTLM. Uwierzytelnianie dwukierunkowe jest udostępniane w systemie Windows 2000przy użyciu usług uwierzytelniania Kerberos .

Program obsługi wyjścia zabezpieczeń jest dostarczany w formacie źródłowym i obiektowym. Można użyć kodu obiektu w takim stanie, w jakim jest, lub użyć kodu źródłowego jako punktu początkowego do utworzenia własnych programów użytkownika. Więcej informacji na temat używania kodu obiektu lub kodu źródłowego wyjścia zabezpieczeń SSPI zawiera sekcja “Korzystanie z wyjścia zabezpieczeń SSPI w systemie Windows” na stronie 1167

*Programy obsługi wyjścia wysyłania i odbierania kanału*
Programów zewnętrznych wysyłania i odbierania można używać do wykonywania takich zadań, jak kompresja i dekompresja danych. Można określić listę programów obsługi wyjścia wysyłania i odbierania, które mają być uruchamiane po sobie.

Programy obsługi wyjścia wysyłania i odbierania kanału są wywoływane w następujących miejscach cyklu przetwarzania agenta MCA:

- Programy obsługi wyjścia wysyłania i odbierania są wywoływane w celu zainicjowania w inicjacji MCA oraz w celu zakończenia w momencie zakończenia MCA.
- Program obsługi wyjścia wysyłania jest wywoływany na jednym lub innym końcu kanału, w zależności od tego, na którym końcu jest wysyłana transmisja dla jednej transmisji komunikatu, bezpośrednio przed wysłaniem transmisji przez łącze. Uwaga 4 wyjaśnia, dlaczego wyjścia są dostępne w obu kierunkach, nawet jeśli kanały komunikatów wysyłają komunikaty tylko w jednym kierunku.
- Program obsługi wyjścia odbierania jest wywoływany na jednym lub innym końcu kanału, w zależności od zakończenia, na którym odbierana jest transmisja dla jednej transmisji komunikatu, natychmiast po odebraniu transmisji z łącza. Uwaga 4 wyjaśnia, dlaczego wyjścia są dostępne w obu kierunkach, nawet jeśli kanały komunikatów wysyłają komunikaty tylko w jednym kierunku.

Może istnieć wiele transmisji dla jednego przesyłania komunikatu i wiele iteracji programów obsługi wyjścia wysyłania i odbierania, zanim komunikat dotrze do wyjścia komunikatu na odbierającym końcu.

Do programów obsługi wyjścia wysyłania i odbierania kanału przekazywany jest bufor agenta zawierający dane transmisji wysłane lub odebrane z łącza komunikacyjnego. W przypadku programów obsługi wyjścia wysyłania pierwsze 8 bajtów buforu jest zarezerwowanych do użycia przez agent MCA i nie może być zmieniane. Jeśli program zwraca inny bufor, to te pierwsze 8 bajtów musi istnieć w nowym buforze. Format danych prezentowanych w programach obsługi wyjścia nie jest zdefiniowany.

Dobry kod odpowiedzi musi zostać zwrócony przez programy obsługi wyjścia wysyłania i odbierania. Każda inna odpowiedź powoduje nieprawidłowe zakończenie MCA (nieprawidłowe zakończenie).

**Uwaga:** Nie należy wywoływać wywołania MQGET, MQPUT lub MQPUT1 w punkcie synchronizacji z wyjścia wysyłania lub odbierania.

**Uwaga:**

1. Wyjścia wysyłania i odbierania zwykle działają w parach. Na przykład wyjście wysyłania może kompresować dane i dekompresować je przy użyciu wyjścia odbierania lub wyjście wysyłania może szyfrować dane i deszyfrować je przy użyciu wyjścia odbierania. Podczas definiowania odpowiednich kanałów należy upewnić się, że kompatybilne programy obsługi wyjścia są nazwane dla obu końców kanału.
2. Jeśli kompresja jest włączona dla kanału, do wyjść są przekazywane skompresowane dane.
3. Wyjścia wysyłania i odbierania kanału mogą być wywoływane dla segmentów komunikatów innych niż dla danych aplikacji, na przykład dla komunikatów o statusie. Nie są one wywoływane w oknie dialogowym uruchamiania ani w fazie sprawdzania zabezpieczeń.
4. Chociaż kanały komunikatów wysyłają komunikaty tylko w jednym kierunku, dane sterowania kanałami, takie jak puls i zakończenie przetwarzania wsadowego, przepływy w obu kierunkach i te wyjścia są również dostępne w obu kierunkach. Jednak niektóre początkowe przepływy danych dotyczące uruchamiania kanału są wyłączone z przetwarzania przez wszystkie wyjścia.
5. Istnieją okoliczności, w których wyjścia wysyłania i odbierania mogą być wywoływane poza kolejnością, na przykład w przypadku uruchamiania serii programów obsługi wyjścia lub w przypadku uruchamiania wyjść zabezpieczeń. Następnie, po pierwszym wywołaniu wyjścia odbierania w celu przetworzenia danych, może ono odebrać dane, które nie przeszły przez odpowiednie wyjście wysyłania. Jeśli wyjście odbierania właśnie wykonało operację, na przykład dekompresję, bez uprzedniego sprawdzenia, czy jest ona wymagana, wyniki będą nieoczekiwane.

   Wyjścia nadawcze i odbiorcze należy zakodować w taki sposób, aby wyjście odbiorcze sprawdzało, czy odbierane dane zostały przetworzone przez odpowiednie wyjście nadawcze. Zalecanym sposobem jest zakodowanie programów obsługi wyjścia w taki sposób, aby:

   - Wyjście wysyłania ustawia wartość dziewiątego bajtu danych na 0 i przed wykonaniem operacji przesuwa wszystkie dane wzdłuż 1 bajtu. (Pierwsze 8 bajtów jest zarezerwowanych do użytku przez MCA).
   - Jeśli wyjście odbierania odbiera dane, które mają wartość 0 w bajcie 9, wie, że dane pochodzą z wyjścia wysyłania. Usuwa 0, wykonuje operację komplementarną i przesuwa dane wynikowe z powrotem o 1 bajt.
   - Jeśli wyjście odbierania odbiera dane, które mają wartość inną niż 0 w bajcie 9, przyjmuje, że wyjście wysyłania nie zostało uruchomione, i wysyła dane z powrotem do programu wywołującego bez zmian.

   Jeśli używane są wyjścia zabezpieczeń, to jeśli kanał zostanie zakończony przez wyjście zabezpieczeń, możliwe jest, że wyjście wysyłania zostanie wywołane bez odpowiadającego mu wyjścia odbierania. Jednym ze sposobów zapobiegania temu problemowi jest zakodowanie wyjścia zabezpieczeń w celu ustawienia flagi w danych MQCD.SecurityUserData lub MQCD.SendUserData, na przykład wtedy, gdy wyjście decyduje o zakończeniu kanału. Następnie wyjście wysyłania musi zaznaczyć to pole i przetwarzać dane tylko wtedy, gdy flaga nie jest ustawiona. To sprawdzenie zapobiega niepotrzebnym modyfikacjom danych przez wyjście wysyłania, a tym samym błędom konwersji, które mogą wystąpić w przypadku odebrania zmienionych danych przez wyjście zabezpieczeń.

*Programy obsługi wyjścia wysyłania kanału-rezerwowanie miejsca*
Wyjścia wysyłania i odbierania mogą być używane do transformowania danych przed transmisją. Programy obsługi wyjścia wysyłania kanału mogą dodawać własne dane o transformacji, rezerwując miejsce w buforze transmisji.

Dane te są przetwarzane przez program obsługi wyjścia odbierania, a następnie usuwane z buforu. Na przykład można zaszyfrować dane i dodać klucz zabezpieczeń do deszyfrowania.

## Jak rezerwować miejsce i korzystać z niego

Gdy program obsługi wyjścia wysyłania jest wywoływany w celu zainicjowania, w polu *ExitSpace* programu MQXCP należy ustawić liczbę bajtów do zarezerwowania. Szczegółowe informacje na ten temat zawiera sekcja MQCXP . Parametr *ExitSpace* można ustawić tylko podczas inicjowania, czyli gdy

parametr *ExitReason* ma wartość MQXR_INIT. Gdy wyjście wysyłania jest wywoływane bezpośrednio przed transmisją, z parametrem *ExitReason* ustawionym na wartość MQXR_XMIT, *ExitSpace* bajtów jest rezerwowanych w buforze transmisji. Produkt *ExitSpace* nie jest obsługiwany w systemie z/OS.

Wyjście wysyłania nie musi używać całej zarezerwowanej przestrzeni. Może on używać mniej niż *ExitSpace* bajtów lub, jeśli bufor transmisji nie jest pełny, wyjście może używać więcej niż zarezerwowana ilość. Ustawiając wartość *ExitSpace*, należy pozostawić co najmniej 1 kB danych komunikatu w buforze transmisji. Jeśli dla dużych ilości danych jest używane zarezerwowane miejsce, może to mieć wpływ na wydajność kanału.

Bufor transmisji ma zwykle długość 32KB . Jeśli jednak kanał używa protokołu TLS, wielkość buforu transmisji jest zmniejszana do 15,352 bajtów w celu dopasowania do maksymalnej długości rekordu zdefiniowanej w dokumencie RFC 6101 i pokrewnej rodziny standardów TLS. Dalsze 1024 bajty są zarezerwowane do użytku przez IBM MQ, dlatego maksymalna przestrzeń buforu transmisji używana przez wyjścia nadawcze wynosi 14 328 bajtów.

## Co się dzieje na odbierającym końcu kanału

Programy obsługi wyjścia odbierania kanału muszą być skonfigurowane tak, aby były zgodne z odpowiednimi wyjściami wysyłania. Wyjścia odbierania muszą znać liczbę bajtów w zarezerwowanym obszarze i muszą usunąć dane w tym obszarze.

## Wiele wyjść wysyłania

Można określić listę programów obsługi wyjścia wysyłania i odbierania, które mają być uruchamiane po sobie. IBM MQ przechowuje łączną ilość miejsca zarezerwowanego przez wszystkie wyjścia wysyłania. Ta łączna ilość miejsca musi pozostać co najmniej 1 kB dla danych komunikatu w buforze transmisji.

W poniższym przykładzie przedstawiono sposób przydzielania miejsca dla trzech wyjść wysyłania wywoływanych kolejno:

1. Po wywołaniu w celu zainicjowania:
   - Wyjście wysyłania A rezerwuje 1 kB.
   - Wyjście wysyłania B rezerwuje 2 kB.
   - Wyjście wysyłania C rezerwuje 3 kB.
2. Maksymalna wielkość transmisji wynosi 32 kB, a dane użytkownika mają długość 5 kB.
3. Wyjście A jest wywoływane z 5 KB danych; dostępne jest do 27 KB, ponieważ 5 KB jest zarezerwowane dla wyjść B i C. Wyjście A dodaje 1 kB zarezerwowanej ilości.
4. Wyjście B jest wywoływane z danymi o wielkości 6 kB; dostępne jest do 29 kB, ponieważ 3 kB jest zarezerwowane dla wyjścia C. Wyjście B dodaje 1 kB, mniej niż 2 kB, które zarezerwowano.
5. Wyjście C jest wywoływane z danymi o wielkości 7 kB; dostępne są do 32 kB. Wyjście C dodaje 10Kwięcej niż 3 kB, które zarezerwowano. Ta wielkość jest poprawna, ponieważ łączna ilość danych (17 kB) jest mniejsza niż wartość maksymalna wynosząca 32 kB.

Maksymalna wielkość buforu transmisji dla kanału używającego protokołu TLS wynosi 15,352 bajtów, a nie 32KB. Wynika to z faktu, że bazowe segmenty transmisji gniazda chronionego są ograniczone do 16KB , a część miejsca jest wymagana na potrzeby narzutu rekordu TLS. Dalsze 1024 bajty są zarezerwowane do użytku przez IBM MQ, dlatego maksymalna przestrzeń buforu transmisji używana przez wyjścia nadawcze wynosi 14 328 bajtów.

*Programy obsługi wyjścia komunikatów kanału*
Wyjścia komunikatów kanału można używać do wykonywania takich zadań, jak szyfrowanie łącza, sprawdzanie poprawności lub podstawianie przychodzących identyfikatorów użytkowników, konwersja danych komunikatów, kronikowanie i obsługa komunikatów referencyjnych. Można określić listę programów obsługi wyjścia komunikatów, które mają być uruchamiane po sobie.

Programy obsługi wyjścia komunikatów kanału są wywoływane w następujących miejscach cyklu przetwarzania agenta MCA:

- Przy inicjacji i zakończeniu MCA
- Natychmiast po wysłaniu wywołania MQGET przez wysyłający agent MCA
- Przed wysłaniem przez odbierający agent MCA wywołania MQPUT

Do wyjścia komunikatu przekazywany jest bufor agenta zawierający nagłówek kolejki transmisji MQXQH oraz tekst komunikatu aplikacji pobrany z kolejki. Format MQXQH jest podany w nagłówku kolejki transmisji MQXQH.

Jeśli używane są komunikaty odniesienia (to znaczy komunikaty, które zawierają tylko nagłówek wskazujący na inny obiekt, który ma zostać wysłany), wyjście komunikatu rozpoznaje nagłówek MQRMH. Identyfikuje on obiekt, pobiera go w dowolny odpowiedni sposób, dołącza go do nagłówka i przekazuje do agenta MCA w celu transmisji do odbierającego agenta MCA. W odbierającym MCA inne wyjście komunikatu rozpoznaje, że komunikat ten jest komunikatem referencyjnym, wyodrębnia obiekt i przekazuje nagłówek do kolejki docelowej. Więcej informacji na temat komunikatów referencyjnych i przykładowych wyjść komunikatów, które je obsługują, zawierają “Komunikaty referencyjne” na stronie 820 i “Uruchamianie przykładowych komunikatów referencyjnych” na stronie 1138 .

Wyjścia komunikatów mogą zwracać następujące odpowiedzi:

- Wyślij komunikat (wyjście GET). Komunikat mógł zostać zmieniony przez wyjście. (Powoduje to zwrócenie komunikatu MQXCC_OK).
- Umieść komunikat w kolejce (wyjście PUT). Komunikat mógł zostać zmieniony przez wyjście. (Powoduje to zwrócenie komunikatu MQXCC_OK).
- Nie przetwarzaj komunikatu. Komunikat jest umieszczany w kolejce niedostarczonych komunikatów (kolejce niedostarczonych komunikatów) przez agent MCA.
- Zamknij kanał.
- Błędny kod powrotu, który powoduje nieprawidłowe zakończenie agenta MCA.

**Uwaga:**

1. Wyjścia komunikatów są wywoływane raz dla każdego przesyłanego kompletnego komunikatu, nawet jeśli komunikat jest podzielony na części.
2. Linux AIX Jeśli w systemie AIX lub Linuxzostanie podane wyjście komunikatu, automatyczna konwersja identyfikatorów użytkowników na małe litery (opisana tutaj) nie będzie działać.
3. Wyjście jest uruchamiane w tym samym wątku, co sam agent MCA. Działa również w tej samej jednostce pracy (UOW) co agent MCA, ponieważ używa tego samego uchwytu połączenia. Dlatego wszystkie wywołania wykonane w punkcie synchronizacji są zatwierdzane lub wycofywane przez kanał na końcu zadania wsadowego. Na przykład jeden program obsługi wyjścia komunikatów kanału może wysyłać komunikaty powiadomień do innego, a te komunikaty są zatwierdzane w kolejce tylko wtedy, gdy zatwierdzane jest zadanie wsadowe zawierające oryginalny komunikat.

   Dlatego można wywołać wywołania MQI punktu synchronizacji z programu obsługi wyjścia komunikatów kanału.

*Konwersja komunikatu poza wyjściem komunikatu*
Przed wywołaniem wyjścia komunikatu odbierający agent MCA wykonuje konwersję komunikatu. W tej sekcji opisano algorytmy używane do wykonywania konwersji.

## Które nagłówki są przetwarzane

Procedura konwersji jest uruchamiana w MCA odbiornika przed wywołaniem wyjścia komunikatu. Procedura konwersji rozpoczyna się od nagłówka MQXQH na początku komunikatu. Następnie procedura konwersji przetwarza nagłówki w łańcuchu, które następują po MQXQH, wykonując w razie potrzeby konwersję. Nagłówki w łańcuchu mogą wykraczać poza przesunięcie zawarte w parametrze HeaderLength danych MQCXP przekazywanych do wyjścia komunikatu odbiornika. Następujące nagłówki są przekształcane w miejscu:

- MQXQH (nazwa formatu " MQXMIT ").
- MQMD (ten nagłówek jest częścią MQXQH i nie ma nazwy formatu)
- MQMDE (nazwa formatu " MQHMDE ").
- MQDH (nazwa formatu) MQHDIST ").
- MQWIH (nazwa formatu " MQHWIH ").

Następujące nagłówki nie są przekształcane, ale są wykonywane krokowe, gdy agent MCA kontynuuje przetwarzanie nagłówków połączonych w łańcuch:

- MQDLH (nazwa formatu " MQDEAD ").
- wszystkie nagłówki z nazwami formatów rozpoczynającymi się od trzech znaków MQH (na przykład " MQHRF "). które nie zostały wymienione w inny sposób

## Sposób przetwarzania nagłówków

Parametr Format każdego nagłówka IBM MQ jest odczytywany przez agent MCA. Parametr Format zawiera 8 bajtów w nagłówku, czyli 8 znaków jednobajtowych zawierających nazwę.

Agent MCA następnie interpretuje dane następujące po każdym nagłówku jako dane nazwanego typu. Jeśli format jest nazwą typu nagłówka kwalifikującego się do konwersji danych IBM MQ , jest on przekształcany. Jeśli jest to inna nazwa wskazująca dane produktu innego niżMQ (na przykład MQFMT_NONE lub MQFMT_STRING), agent MCA zatrzymuje przetwarzanie nagłówków.

## Czym jest HeaderLengthsystemu MQCXP?

Parametr HeaderLength w danych MQCXP dostarczonych do wyjścia komunikatu to łączna długość nagłówków MQXQH (w tym MQMD), MQMDE i MQDH na początku komunikatu. Te nagłówki są połączone w łańcuchy przy użyciu nazw i długości formatu.

## MQWIH

Połączone nagłówki mogą wykraczać poza nagłówek HeaderLength w obszarze danych użytkownika. Nagłówek MQWIH, jeśli istnieje, jest jednym z tych nagłówków, które są wyświetlane poza nagłówkiem HeaderLength.

Jeśli w nagłówkach dołączonych do łańcucha znajduje się nagłówek MQWIH, jest on przekształcany w miejscu przed wywołaniem wyjścia komunikatu odbiornika.

*Program obsługi wyjścia ponowienia komunikatu kanału*
Wyjście ponowienia komunikatu kanału jest wywoływane, gdy próba otwarcia kolejki docelowej nie powiedzie się. Za pomocą wyjścia można określić, w jakich okolicznościach należy ponowić próbę, ile razy i jak często.

To wyjście jest również wywoływane na odbierającym końcu kanału podczas inicjowania i kończenia MCA.

Do wyjścia ponowienia komunikatu kanału przekazywany jest bufor agenta zawierający nagłówek kolejki transmisji, MQXQH i tekst komunikatu aplikacji pobrany z kolejki. Format MQXQH jest podany w sekcji Przegląd MQXQH.

Wyjście jest wywoływane dla wszystkich kodów przyczyny; wyjście określa, dla których kodów przyczyny agent MCA ma ponowić próbę, ile razy i w jakich odstępach czasu. (Wartość licznika ponowień komunikatu ustawiona podczas definiowania kanału jest przekazywana do wyjścia w MQCD, ale wyjście może zignorować tę wartość).

Pole licznika MsgRetryw MQCXP jest zwiększane przez agent MCA przy każdym wywołaniu wyjścia i zwraca wartość MQXCC_OK z czasem oczekiwania zawartym w polu przedziału czasu MsgRetryw MQCXP lub MQXCC_SUPPRESS_FUNCTION. Ponowne próby będą kontynuowane w nieskończoność do momentu, gdy wyjście zwróci wartość MQXCC_SUPPRESS_FUNCTION w polu ExitResponse w MQCXP. Informacje na temat działania wykonywanego przez agent MCA dla tych kodów zakończenia zawiera sekcja MQCXP .

Jeśli wszystkie próby nie powiodą się, komunikat zostanie zapisany w kolejce niedostarczonych komunikatów. Jeśli nie ma dostępnej kolejki niedostarczonych komunikatów, kanał zostanie zatrzymany.

Jeśli dla kanału nie zostanie zdefiniowane wyjście dla ponowienia komunikatu i wystąpi awaria, która może być tymczasowa, na przykład MQRC_Q_FULL, agent MCA użyje licznika ponowień komunikatu i interwałów ponowień komunikatu ustawionych podczas definiowania kanału. Jeśli błąd jest bardziej trwały i nie zdefiniowano programu obsługi wyjścia do jego obsługi, komunikat jest zapisywany w kolejce niedostarczonych komunikatów.

*Program obsługi wyjścia automatycznej definicji kanału*
Wyjścia automatycznego definiowania kanału można użyć po odebraniu żądania uruchomienia kanału odbiorczego lub kanału połączenia z serwerem, ale dla tego kanału nie istnieje definicja (nie dla kanału IBM MQ for z/OS ). Można ją również wywoływać na wszystkich platformach dla kanałów wysyłających i odbierających klastry, aby umożliwić modyfikację definicji dla instancji kanału.

Wyjście automatycznej definicji kanału może być wywoływane na wszystkich platformach z wyjątkiem platformy z/OS , gdy zostanie odebrane żądanie uruchomienia kanału odbiornika lub kanału połączenia z serwerem, ale nie istnieje definicja kanału. Można go użyć do zmodyfikowania dostarczonej definicji domyślnej dla automatycznie zdefiniowanego kanału odbiorczego lub kanału połączenia z serwerem, SYSTEM.AUTO.RECEIVERlub SYSTEM.AUTO.SVRCON. Opis sposobu automatycznego tworzenia definicji kanałów zawiera sekcja Przygotowywanie kanałów .

Wyjście automatycznej definicji kanału może być również wywoływane po odebraniu żądania uruchomienia kanału wysyłającego klastry. Można ją wywołać dla kanałów wysyłających i odbierających klastry, aby umożliwić modyfikację definicji dla tej instancji kanału. W tym przypadku wyjście dotyczy również IBM MQ for z/OS. Typowym zastosowaniem wyjścia automatycznego definiowania kanału jest zmiana nazw wyjść komunikatów (MSGEXIT, RCVEXIT, SCYEXIT i SENDEXIT), ponieważ nazwy wyjść mają różne formaty na różnych platformach. Jeśli nie określono wyjścia automatycznej definicji kanału, domyślnym działaniem w systemie z/OS jest sprawdzenie nazwy rozproszonego wyjścia w postaci *[path]/libraryname(function)* i użycie do ośmiu znaków funkcji (jeśli istnieje) lub nazwy biblioteki. W systemie z/OSprogram obsługi wyjścia automatycznej definicji kanału musi zmienić pola adresowane przez pola MsgExitPtr, MsgUserDataPtr, SendExitPtr, SendUserDataPtr, ReceiveExitPtr i ReceiveUserDataPtr, a nie MsgExit, MsgUserData, SendExit, SendUserData, Same pola danych ReceiveExit i ReceiveUser.

Więcej informacji na ten temat zawiera sekcja Praca z kanałami definiowanymi automatycznie.

Podobnie jak w przypadku innych wyjść kanałów, lista parametrów jest następująca:

```
MQ_CHANNEL_AUTO_DEF_EXIT (ChannelExitParms, ChannelDefinition)
```

`ChannelExitParms` są opisane w sekcji MQCXP. `ChannelDefinition` został opisany w sekcji MQCD.

MQCD zawiera wartości, które są używane w domyślnej definicji kanału, jeśli nie zostały zmienione przez wyjście. Wyjście może modyfikować tylko podzbiór pól; patrz sekcja MQ_CHANNEL_AUTO_DEF_EXIT. Jednak próba zmiany innych pól nie powoduje błędu.

Wyjście automatycznej definicji kanału zwraca odpowiedź MQXCC_OK lub MQXCC_SUPPRESS_FUNCTION. Jeśli żadna z tych odpowiedzi nie zostanie zwrócona, agent MCA kontynuuje przetwarzanie tak, jakby została zwrócona wartość MQXCC_SUPPRESS_FUNCTION. Oznacza to, że definicja automatyczna jest porzucana, nie jest tworzona nowa definicja kanału i nie można uruchomić kanału.

## ALW *Kompilowanie programów obsługi wyjścia kanału w systemach AIX, Linux, and Windows*

Poniższe przykłady ułatwiają kompilowanie programów obsługi wyjścia kanału dla systemów AIX, Linux, and Windows .

## Windows

Windows

Komenda kompilatora i konsolidatora dla programów obsługi wyjścia kanału w systemie Windows:

```
cl.exe /Ic:\mqm\tools\c\include /nologo /c myexit.c
link.exe /nologo /dll myexit.obj /def:myexit.def /out:myexit.dll
```

## Systemy AIX and Linux

Linux AIX

W tych przykładach `exit` jest nazwą biblioteki, a `ChannelExit` jest nazwą funkcji. W systemie AIX plik eksportu ma nazwę `exit.exp`. Nazwy te są używane przez definicję kanału do odwoływania się do programu obsługi wyjścia przy użyciu formatu opisanego w sekcji Definicja kanału MQCD. Patrz także opis parametru MSGEXIT komendy DEFINE CHANNEL .

AIX Przykładowe komendy kompilatora i konsolidatora dla wyjść kanałów w systemie AIX:

```
$ xlc_r -q64 -e MQStart -bE:exit.exp -bM:SRE -o /var/mqm/exits64/exit
exit.c -I/usr/mqm/inc
```

Linux Przykładowe komendy kompilatora i konsolidatora dla wyjść kanałów w systemie Linux , w którym menedżer kolejek jest 32-bitowy:

```
$ gcc -shared -fPIC -o /var/mqm/exits/exit exit.c -I/opt/mqm/inc
```

Linux Przykładowe komendy kompilatora i konsolidatora dla wyjść kanałów w systemie Linux , w którym menedżer kolejek jest 64-bitowy:

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
```

Na kliencie można użyć 32-bitowego lub 64-bitowego wyjścia. To wyjście musi być połączone z mqic_r.

AIX W systemie AIXwszystkie funkcje wywoływane przez funkcję IBM MQ muszą zostać wyeksportowane. Przykładowy plik eksportu dla tego pliku make:

```
#
!channelExit
MQStart
```

### *Konfigurowanie wyjść kanałów*

Aby wywołać wyjście kanału, należy nadać mu nazwę w definicji kanału.

Wyjścia kanału muszą być nazwane w definicji kanału. Można to zrobić podczas pierwszego definiowania kanałów lub można dodać informacje później, na przykład za pomocą komendy MQSC ALTER CHANNEL. Można również podać nazwy wyjść kanału w strukturze danych kanału MQCD. Format nazwy wyjścia zależy od platformy IBM MQ . Informacje na ten temat można znaleźć w sekcji MQCD lub w sekcji Komendy MQSC .

Jeśli definicja kanału nie zawiera nazwy programu obsługi wyjścia użytkownika, program ten nie jest wywoływany.

Wyjście automatycznej definicji kanału jest właściwością menedżera kolejek, a nie pojedynczego kanału. Aby to wyjście mogło zostać wywołane, musi być nazwane w definicji menedżera kolejek. Aby zmienić definicję menedżera kolejek, użyj komendy MQSC ALTER QMGR.

## Zapisywanie wyjść konwersji danych

Ta kolekcja tematów zawiera informacje na temat zapisywania wyjść konwersji danych.

**Uwaga:** Nieobsługiwane w programie MQSeries dla VSE/ESA.

Podczas wykonywania operacji MQPUT aplikacja tworzy deskryptor komunikatu (MQMD) dla komunikatu. Ponieważ produkt IBM MQ musi być w stanie zrozumieć treść deskryptora MQMD niezależnie od platformy, na której został utworzony, jest on automatycznie przekształcany przez system.

Jednak dane aplikacji nie są przekształcane automatycznie. Jeśli dane znakowe są wymieniane między platformami, na których pola `CodedCharSetId` i `Encoding` są różne, na przykład między formatami ASCII i EBCDIC, aplikacja musi zorganizować konwersję komunikatu. Konwersja danych aplikacji może być wykonywana przez sam menedżer kolejek lub przez program użytkownika, nazywany *programem obsługi wyjścia konwersji danych*. Menedżer kolejek może sam przeprowadzić konwersję danych przy użyciu jednej z wbudowanych procedur konwersji, jeśli dane aplikacji są w jednym z wbudowanych formatów (takich jak MQFMT_STRING). Ta sekcja zawiera informacje na temat narzędzia obsługi wyjścia konwersji danych, które jest udostępniane przez produkt IBM MQ , gdy dane aplikacji nie są we wbudowanym formacie.

Sterowanie może zostać przekazane do wyjścia konwersji danych podczas wywołania MQGET. Pozwala to uniknąć konwersji między różnymi platformami przed osiągnięciem miejsca docelowego. Jeśli jednak ostateczne miejsce docelowe jest platformą, która nie obsługuje konwersji danych w MQGET, należy określić wartość CONVERT (YES) w kanale nadawczym, który wysyła dane do swojego końcowego miejsca docelowego. Dzięki temu program IBM MQ przekształca dane podczas transmisji. W takim przypadku wyjście konwersji danych musi znajdować się w systemie, w którym zdefiniowany jest kanał nadawczy.

Wywołanie MQGET jest wykonywane bezpośrednio przez aplikację. W polach `CodedCharSetId` i `Encoding` w strukturze MQMD ustaw wymagany zestaw znaków i kodowanie. Jeśli aplikacja używa tego samego zestawu znaków i kodowania co menedżer kolejek, należy ustawić parametr `CodedCharSetId` na wartość MQCCSI_Q_MGR, a parametr `Encoding` na wartość MQENC_NATIVE. Po zakończeniu wywołania MQGET te pola mają wartości odpowiednie dla zwracanych danych komunikatu. Mogą one różnić się od wartości wymaganych, jeśli konwersja nie powiodła się. Aplikacja powinna zresetować te pola do wartości wymaganych przed każdym wywołaniem MQGET.

Warunki wymagane do wywołania wyjścia konwersji danych są zdefiniowane dla wywołania MQGET w MQGET.

Opis parametrów przekazywanych do wyjścia konwersji danych oraz szczegółowe uwagi dotyczące użycia znajdują się w sekcji Konwersja danych dla wywołania komendy MQ_DATA_CONV_EXIT i struktury MQDXP.

Programy, które przekształcają dane aplikacji między różnymi kodowaniami maszynowymi i identyfikatorami CCSID, muszą być zgodne z interfejsem DCI (Data conversion interface) firmy IBM MQ .

W przypadku klientów rozsyłania grupowego wyjścia funkcji API i wyjścia konwersji danych muszą działać po stronie klienta, ponieważ niektóre komunikaty mogą nie przechodzić przez menedżer kolejek. Następujące biblioteki są częścią pakietów klienta oraz pakietów serwera:

*Tabela 143. Biblioteki, które znajdują się w pakietach klienta i serwera*

| System operacyjny | Biblioteki |
|---|---|
| AIX | Wersja 32-bitowa i 64-bitowa: libmqm.a & libmqm_r.a |
| IBM i | LIBMQM i LIBMQM_R |
| Linux | Wersja 32-bitowa i 64-bitowa: libmqm.so & libmqm_r.so |

| Tabela 143. Biblioteki, które znajdują się w pakietach klienta i serwera (kontynuacja) | |
|---|---|
| **System operacyjny** | **Biblioteki** |
| Windows | Wersja 32-bitowa i 64-bitowa: mqm.dll & mqm.pdb |

### *Wywoływanie wyjścia konwersji danych*

Wyjście konwersji danych to zapisane przez użytkownika wyjście, które odbiera sterowanie podczas przetwarzania wywołania MQGET.

Wyjście jest wywoływane, jeśli spełnione są następujące warunki:

- Opcja MQGMO_CONVERT jest określona w wywołaniu MQGET.
- Niektóre lub wszystkie dane komunikatu nie są w żądanym zestawie znaków lub kodowaniu.
- Pole *Format* w strukturze MQMD powiązanej z komunikatem nie ma wartości MQFMT_NONE.
- Wartość *BufferLength* określona w wywołaniu MQGET nie jest równa zero.
- Długość danych komunikatu jest różna od zera.
- Komunikat zawiera dane w formacie zdefiniowanym przez użytkownika. Format zdefiniowany przez użytkownika może zajmować cały komunikat lub być poprzedzony jednym lub kilkoma wbudowanymi formatami. Na przykład format zdefiniowany przez użytkownika może być poprzedzony formatem MQFMT_DEAD_LETTER_HEADER. Wyjście jest wywoływane w celu przekształcenia tylko formatu zdefiniowanego przez użytkownika; menedżer kolejek przekształca wszystkie wbudowane formaty, które poprzedzają format zdefiniowany przez użytkownika.

  Można również wywołać program zewnętrzny napisany przez użytkownika w celu przekształcenia formatu wbudowanego, ale dzieje się tak tylko wtedy, gdy wbudowane procedury konwersji nie mogą pomyślnie przekształcić formatu wbudowanego.

Istnieją inne warunki opisane w pełni w uwagach dotyczących użycia wywołania komendy MQ_DATA_CONV_EXIT w tabeli MQ_DATA_CONV_EXIT.

Szczegółowe informacje na temat wywołania MQGET zawiera sekcja MQGET . Wyjścia konwersji danych nie mogą używać wywołań MQI innych niż MQXCNVC.

Nowa kopia wyjścia jest ładowana, gdy aplikacja próbuje pobrać pierwszy komunikat, który używa tego komunikatu *Format* od czasu połączenia aplikacji z menedżerem kolejek. Nowa kopia może również zostać załadowana w innym czasie, jeśli menedżer kolejek odrzucił poprzednio załadowaną kopię.

Wyjście konwersji danych działa w środowisku podobnym do środowiska programu, który wywołał wywołanie MQGET. Oprócz aplikacji użytkownika programem może być agent MCA (agent kanału komunikatów) wysyłający komunikaty do docelowego menedżera kolejek, który nie obsługuje konwersji komunikatów. Środowisko obejmuje przestrzeń adresową i profil użytkownika, jeśli ma to zastosowanie. Wyjście nie może naruszyć integralności menedżera kolejek, ponieważ nie działa w środowisku menedżera kolejek.

## Konwersja danych w systemie z/OS

z/OS

W systemie z/OSnależy pamiętać o następujących sprawach:

- Programy obsługi wyjścia mogą być napisane tylko w języku asemblera.
- Programy obsługi wyjścia muszą być wielobieżne i mogą działać w dowolnym miejscu w pamięci.
- Programy obsługi wyjścia muszą odtworzyć środowisko przy wyjściu do tej pozycji i zwolnić każdą uzyskaną pamięć.
- Programy obsługi wyjścia nie mogą CZEKAĆ ani wystawiać ESTAE lub SPIEs.
- Programy obsługi wyjścia są zwykle wywoływane tak, jakby były wywoływane przez z/OS LINK w:
  - Nieautoryzowany stan programu problemu

  - Tryb sterowania podstawową przestrzenią adresową
  - Tryb inny niż międzypamięciowy
  - Tryb bez rejestracji dostępu
  - 31-bitowy tryb adresowania
  - Tryb TCB-PRB
- W przypadku użycia przez aplikację CICS wyjście jest wywoływane przez EXEC CICS LINK i musi być zgodne z konwencjami programistycznymi systemu CICS . Parametry są przekazywane przez wskaźniki (adresy) w obszarze komunikacyjnym CICS (COMMAREA).

  Chociaż nie jest to zalecane, programy użytkownika obsługi wyjścia mogą również korzystać z wywołań funkcji API języka CICS , z zachowaniem następującej ostrożności:

  - Nie należy wydawać punktów synchronizacji, ponieważ wyniki mogą mieć wpływ na jednostki pracy zadeklarowane przez agent MCA.
  - Nie należy aktualizować żadnych zasobów sterowanych przez menedżer zasobów inny niż IBM MQ for z/OS, w tym zasobów sterowanych przez serwer CICS Transaction Server.

  W przypadku kanałów z wartością CONVERT = YES wyjście jest ładowane z zestawu danych, do którego odwołuje się instrukcja DD CSQXLIB. Dostarczone z produktem MQwyjścia CSQCBDCI i CSQCBDCO dla mostu IBM MQ CICS znajdują się w SCSQAUTH.

## *Pisanie programu obsługi wyjścia konwersji danych dla systemu IBM i*

Informacje o krokach, które należy wziąć pod uwagę podczas pisania programów obsługi wyjścia konwersji danych produktu MQ dla systemu IBM i.

Wykonaj następujące kroki:

1. Podaj nazwę formatu komunikatu. Nazwa musi mieścić się w polu *Format* deskryptora MQMD. Nazwa *Format* nie może zawierać początkowych odstępów wewnętrznych, a końcowe odstępy są ignorowane. Nazwa obiektu nie może zawierać więcej niż osiem niepustych znaków, ponieważ *Format* ma tylko osiem znaków długości. Pamiętaj, aby używać tej nazwy za każdym razem, gdy wysyłasz wiadomość (w naszym przykładzie używana jest nazwa Format).
2. Utwórz strukturę reprezentującą komunikat. Przykład można znaleźć w sekcji Poprawna składnia .
3. Tę strukturę należy uruchomić za pomocą komendy CVTMQMDTA, aby utworzyć fragment kodu dla wyjścia konwersji danych.

   Funkcje wygenerowane przez komendę CVTMQMDTA używają makr, które są dostarczane w pliku QMQM/H (AMQSVMHA). Makra te są zapisywane przy założeniu, że wszystkie struktury są spakowane; w przeciwnym razie należy je zmienić.
4. Utwórz kopię dostarczonego szkieletu zbioru źródłowego QMQMSAMP/QCSRC (AMQSVFC4) i zmień jego nazwę. (W naszym przykładzie używana jest nazwa EXIT_MOD.)
5. Znajdź następujące pola komentarza w pliku źródłowym i wstaw kod zgodnie z opisem:
   a. Pod koniec pliku źródłowego, pole komentarza zaczyna się od:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   W tym miejscu należy wstawić fragment kodu wygenerowany w kroku “3” na stronie 1015.

   b. W pobliżu środka pliku źródłowego pole komentarza rozpoczyna się od:

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   Po nim następuje przekształcone w komentarz wywołanie funkcji `ConverttagSTRUCT`.

   Zmień nazwę funkcji na nazwę funkcji, która została dodana w kroku “5.a” na stronie 1015. Usuń znaki komentarza, aby aktywować funkcję. Jeśli istnieje kilka funkcji, utwórz wywołania dla każdej z nich.

c. Obok początku pliku źródłowego, pole komentarza rozpoczyna się od:

```
/* Insert the function prototypes for the functions produced by */
```

W tym miejscu należy wstawić instrukcje prototypu funkcji dla funkcji dodanych w kroku “5.a” na stronie 1015.

Jeśli komunikat zawiera dane znakowe, wygenerowany kod wywołuje MQXCNVC. Można to rozwiązać, wiążąc program usługowy QMQM/LIBMQM.

6. Skompiluj moduł źródłowy EXIT_MOD w następujący sposób:

```
CRTCMOD MODULE(library/EXIT_MOD) +
SRCFILE(QCSRC) +
TERASPACE(*YES *TSIFC)
```

7. Utwórz/dowiąż program.

W przypadku aplikacji niewielowątkowych należy użyć następujących informacji:

```
CRTPGM PGM(library/Format)  +
   MODULE(library/EXIT_MOD)  +
   BNDSRVPGM(QMQM/LIBMQM)  +
   ACTGRP(QMQM)  +
   USRPRF(*USER)
```

Oprócz utworzenia wyjścia konwersji danych dla środowiska podstawowego, w środowisku wielowątkowym wymagane jest inne wyjście. Po tym ładowanym obiekcie musi następować _R. Użyj biblioteki LIBMQM_R, aby rozstrzygnąć wywołania do MQXCNVC. Oba ładowalne obiekty są wymagane dla środowiska wielowątkowego.

```
CRTPGM PGM(library/Format_R)  +
   MODULE(library/EXIT_MOD)  +
   BNDSRVPGM(QMQM/LIBMQM_R)  +
   ACTGRP(QMQM)  +
   USRPRF(*USER)
```

8. Umieść dane wyjściowe na liście bibliotek dla zadania IBM MQ . Zaleca się, aby w środowisku produkcyjnym programy obsługi wyjścia konwersji danych były przechowywane w bibliotece QSYS.

**Uwaga:**

1. Jeśli komenda CVTMQMDTA używa spakowanych struktur, wszystkie aplikacje IBM MQ muszą używać kwalifikatora _Packed.
2. Programy obsługi wyjścia konwersji danych muszą być wielobieżne.
3. MQXCNVC jest jedynym wywołaniem MQI, które można wywołać z wyjścia konwersji danych.
4. Skompiluj program obsługi wyjścia z opcją kompilatora profilu użytkownika ustawioną na *USER, aby program ten działał z uprawnieniami użytkownika.
5. Obsługa pamięci teraprzestrzeni jest wymagana dla wszystkich programów zewnętrznych w systemie IBM MQ for IBM i ; Określ parametr TERASPACE (*YES *TSIFC) w komendach CRTCMOD i CRTBNDC.

## *Pisanie programu obsługi wyjścia konwersji danych dla systemu IBM MQ for z/OS*

Informacje o krokach, które należy wziąć pod uwagę podczas pisania programów obsługi wyjścia konwersji danych dla systemu IBM MQ for z/OS.

Wykonaj następujące kroki:

1. Pobierz dostarczony szkielet źródłowy CSQ4BAX9 (w przypadku środowisk innych niżCICS ) lub CSQ4CAX9 (w przypadku systemu CICS ). jako punkt wyjścia.
2. Uruchom program narzędziowy CSQUCVX.

3. Postępuj zgodnie z instrukcjami w prologu CSQ4BAX9 lub CSQ4CAX9 , aby uwzględnić procedury wygenerowane przez program narzędziowy CSQUCVX w kolejności, w jakiej struktury występują w komunikacie, który ma zostać przekształcony.
4. Program narzędziowy zakłada, że struktury danych nie są spakowane, że niejawne wyrównanie danych jest uwzględniane oraz że struktury rozpoczynają się od granicy pełnego słowa, a bajty są pomijane zgodnie z wymaganiami (tak jak w przypadku identyfikatorów i wersji w przykładzie w sekcji Poprawna składnia ). Jeśli struktury są spakowane, pomiń wygenerowane makra CMQXCALA. Dlatego należy rozważyć zadeklarowanie struktur w taki sposób, aby wszystkie pola były nazwane i nie były pomijane żadne bajty. W przykładzie z sekcji Poprawna składnianależy dodać pole "MQBYTE DUMMY" między identyfikatorem i wersją.
5. Podane wyjście zwraca błąd, jeśli bufor wejściowy jest krótszy niż format komunikatu, który ma zostać przekształcony. Mimo że wyjście przekształca tyle pól, ile jest to możliwe, błąd powoduje zwrócenie nieprzekształconego komunikatu do aplikacji. Aby umożliwić jak największą konwersję krótkich buforów wejściowych, w tym pól częściowych, należy zmienić wartość TRUNC= w makrze CSQXCDFA na YES: nie jest zwracany żaden błąd, więc aplikacja otrzymuje przekształcony komunikat. Aplikacja musi obsługiwać obcięcie.
6. Dodaj dowolny inny wymagany specjalny kod przetwarzania.
7. Zmień nazwę programu na nazwę formatu danych.
8. Skompiluj i dowiąż program jako wsadowy program użytkowy (chyba że jest przeznaczony do użytku z aplikacjami CICS ). Makra w kodzie wygenerowanym przez program narzędziowy znajdują się w bibliotece **thlqual**.SCSQMACS.

   Jeśli komunikat zawiera dane znakowe, wygenerowany kod wywołuje MQXCNVC. Jeśli wyjście używa tego wywołania, należy je połączyć z kodem pośredniczącym programu obsługi wyjścia CSQASTUB. Kod pośredniczący jest niezależny od języka i środowiska. Alternatywnie można załadować kod pośredniczący dynamicznie przy użyciu nazwy wywołania dynamicznego CSQXCNVC. Więcej informacji zawiera sekcja “Dynamiczne wywoływanie kodu pośredniczącego IBM MQ” na stronie 1060.

   Umieść moduł edytowany przez odsyłacz w bibliotece dynamicznej aplikacji i w zestawie danych, do którego odwołuje się instrukcja CSQXLIB DD procedury zadania uruchomionej przez inicjatora kanału.
9. Jeśli wyjście jest przeznaczone dla aplikacji CICS , należy je skompilować i skonsolidować w taki sam sposób, jak aplikację CICS , w tym CSQASTUB, jeśli jest to wymagane. Umieść go w bibliotece aplikacji CICS . Zdefiniuj program dla CICS w typowy sposób, określając parametr EXECKEY ( CICS ) w definicji.

**Uwaga:** Chociaż biblioteki środowiska wykonawczego LE/370 są potrzebne do uruchomienia programu narzędziowego CSQUCVX (patrz krok “2” na stronie 1016 ), nie są one potrzebne do konsolidacji ani do uruchomienia samego wyjścia konwersji danych (patrz kroki “8” na stronie 1017 i “9” na stronie 1017 ).

Więcej informacji na temat konwersji danych w obrębie mostu IBM MQ - IMS zawiera sekcja “Pisanie aplikacji mostu IMS” na stronie 76 .

## Linux AIX *Zapisywanie wyjścia konwersji danych dla systemów IBM MQ for AIX or Linux*

Informacje o krokach, które należy wziąć pod uwagę podczas zapisywania programów obsługi wyjścia konwersji danych dla systemów IBM MQ for AIX or Linux .

Wykonaj następujące kroki:

1. Podaj nazwę formatu komunikatu. Nazwa musi mieścić się w polu *Format* deskryptora MQMD i musi być zapisana wielkimi literami, na przykład: MYFORMAT. Nazwa *Format* nie może zawierać odstępów początkowych. Odstępy końcowe są ignorowane. Nazwa obiektu nie może zawierać więcej niż osiem niepustych znaków, ponieważ *Format* ma tylko osiem znaków długości. Należy pamiętać, aby używać tej nazwy za każdym razem, gdy wysyłany jest komunikat.

   Jeśli wyjście konwersji danych jest używane w środowisku wielowątkowym, po ładowanym obiekcie musi następować _r, aby wskazać, że jest to wersja wielowątkowa.
2. Utwórz strukturę reprezentującą komunikat. Przykład można znaleźć w sekcji Poprawna składnia .

3. Tę strukturę należy uruchomić za pomocą komendy `crtmqcvx` w celu utworzenia fragmentu kodu dla wyjścia konwersji danych.

   Funkcje generowane przez komendę `crtmqcvx` używają makr, które zakładają, że wszystkie struktury są spakowane. Jeśli tak nie jest, popraw je.

4. Skopiuj dostarczony plik źródłowy szkieletu, zmieniając jego nazwę na nazwę w formacie komunikatu ustawionym w kroku “1” na stronie 1017. Plik źródłowy szkieletu i kopia są tylko do odczytu.

   Szkielet pliku źródłowego nosi nazwę amqsvfc0.c.

5. W systemie IBM MQ for AIXdostarczany jest również plik eksportu szkieletu o nazwie amqsvfc.exp . Skopiuj ten plik, zmieniając jego nazwę na MYFORMAT.EXP.

6. Szkielet zawiera przykładowy plik nagłówkowy amqsvmha.hw katalogu *MQ_INSTALLATION_PATH*/inc, gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym zainstalowano produkt IBM MQ . Upewnij się, że ścieżka włączania wskazuje ten katalog, aby pobrać ten plik.

   Plik amqsvmha.h zawiera makra używane w kodzie wygenerowanym przez komendę `crtmqcvx` . Jeśli struktura do konwersji zawiera dane znakowe, te makra wywołują MQXCNVC.

7. Znajdź następujące pola komentarza w pliku źródłowym i wstaw kod zgodnie z opisem:

   a. Pod koniec pliku źródłowego, pole komentarza zaczyna się od:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   W tym miejscu należy wstawić fragment kodu wygenerowany w kroku “3” na stronie 1018.

   b. W pobliżu środka pliku źródłowego pole komentarza rozpoczyna się od:

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   Po nim następuje przekształcone w komentarz wywołanie funkcji `ConverttagSTRUCT`.

   Zmień nazwę funkcji na nazwę funkcji, która została dodana w kroku “7.a” na stronie 1018. Usuń znaki komentarza, aby aktywować funkcję. Jeśli istnieje kilka funkcji, utwórz wywołania dla każdej z nich.

   c. Obok początku pliku źródłowego, pole komentarza rozpoczyna się od:

   ```
   /* Insert the function prototypes for the functions produced by */
   ```

   W tym miejscu należy wstawić instrukcje prototypu funkcji dla funkcji dodanych w kroku “3” na stronie 1018.

8. Skompiluj wyjście jako bibliotekę współużytkowaną, używając MQStart jako punktu wejścia. W tym celu należy zapoznać się z sekcją “Kompilowanie wyjść konwersji danych w systemach AIX and Linux” na stronie 1018.

9. Umieść dane wyjściowe w katalogu wyjścia. Domyślnym katalogiem wyjścia jest `/var/mqm/exits` dla systemów 32-bitowych i `/var/mqm/exits64`dla systemów 64-bitowych. Katalogi te można zmienić w pliku qm.ini lub w pliku mqclient.ini . Tę ścieżkę można ustawić dla każdego menedżera kolejek, a wyjście jest wyszukiwany tylko w tej ścieżce lub ścieżkach.

**Uwaga:**

1. Jeśli produkt `crtmqcvx` używa spakowanych struktur, wszystkie aplikacje IBM MQ muszą zostać skompilowane w ten sposób.
2. Programy obsługi wyjścia konwersji danych muszą być wielobieżne.
3. MQXCNVC jest jedynym wywołaniem MQI, które można wywołać z wyjścia konwersji danych.

Linux AIX *Kompilowanie wyjść konwersji danych w systemach AIX and Linux*

Przykłady kompilowania wyjścia konwersji danych w systemach AIX and Linux .

Na wszystkich platformach punktem wejścia do modułu jest MQStart.

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

## AIX

AIX

Skompiluj kod źródłowy wyjścia, wydając jedną z następujących komend:

**Aplikacje 32-bitowe**

**Niewątkowy**

```
cc -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

**Wątkowy**

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

**Aplikacje 64-bitowe**

**Niewątkowy**

```
cc -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT \
     MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

**Wątkowy**

```
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

Skompiluj kod źródłowy wyjścia, wydając jedną z następujących komend:

**Aplikacje 31-bitowe**

**Niewątkowy**

```
 gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c \
      -I  MQ_INSTALLATION_PATH/inc
```

**Wątkowy**

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

**Aplikacje 32-bitowe**

**Niewątkowy**

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

**Wątkowy**

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

**Aplikacje 64-bitowe**

**Niewątkowy**

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

**Wątkowy**

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

## Windows *Zapisywanie wyjścia konwersji danych dla IBM MQ for Windows*

Informacje o krokach, które należy wziąć pod uwagę podczas pisania programów obsługi wyjścia konwersji danych dla systemu IBM MQ for Windows.

Wykonaj następujące kroki:

1. Podaj nazwę formatu komunikatu. Nazwa musi mieścić się w polu *Format* deskryptora MQMD. Nazwa *Format* nie może zawierać odstępów początkowych. Odstępy końcowe są ignorowane. Nazwa obiektu nie może zawierać więcej niż osiem niepustych znaków, ponieważ *Format* ma tylko osiem znaków długości.

   Plik .DEF o nazwie amqsvfcn.def jest również dostarczany w katalogu przykładów *MQ_INSTALLATION_PATH*\Tools\C\Samples. *MQ_INSTALLATION_PATH* to katalog, w którym zainstalowano produkt IBM MQ . Utwórz kopię tego pliku i zmień jej nazwę, na przykład na MYFORMAT.DEF. Upewnij się, że nazwa tworzonej biblioteki DLL i nazwa podana w formacie MYFORMAT.DEF są takie same. Zastąp nazwę FORMAT1 w formacie MYFORMAT.DEF z nową nazwą formatu.

   Należy pamiętać, aby używać tej nazwy za każdym razem, gdy wysyłany jest komunikat.
2. Utwórz strukturę reprezentującą komunikat. Przykład można znaleźć w sekcji Poprawna składnia .
3. Tę strukturę należy uruchomić za pomocą komendy `crtmqcvx` w celu utworzenia fragmentu kodu dla wyjścia konwersji danych.

   Funkcje generowane przez komendę CRTMQCVX używają makr, które są zapisywane przy założeniu, że wszystkie struktury są spakowane. Jeśli tak nie jest, popraw je.
4. Skopiuj dostarczony plik źródłowy szkieletu, amqsvfc0.c, zmieniając jego nazwę na nazwę w formacie komunikatu ustawionym w kroku “1” na stronie 1020.

   amqsvfc0.c znajduje się w katalogu *MQ_INSTALLATION_PATH*\Tools\C\Samples , gdzie *MQ_INSTALLATION_PATH* jest katalogiem, w którym zainstalowano produkt IBM MQ . (Domyślnym katalogiem instalacyjnym jest C:\Program Files\IBM\MQ).

Szkielet zawiera przykładowy plik nagłówkowy amqsvmha.h w katalogu *MQ_INSTALLATION_PATH*\Tools\C\include . Upewnij się, że ścieżka włączania wskazuje ten katalog, aby pobrać ten plik.

Plik amqsvmha.h zawiera makra używane przez kod wygenerowany przez komendę CRTMQCVX. Jeśli struktura do konwersji zawiera dane znakowe, te makra wywołują MQXCNVC.

5. Znajdź następujące pola komentarza w pliku źródłowym i wstaw kod zgodnie z opisem:

   a. Pod koniec pliku źródłowego, pole komentarza zaczyna się od:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   W tym miejscu należy wstawić fragment kodu wygenerowany w kroku “3” na stronie 1020.

   b. W pobliżu środka pliku źródłowego pole komentarza rozpoczyna się od:

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   Po nim następuje przekształcone w komentarz wywołanie funkcji ConverttagSTRUCT.

   Zmień nazwę funkcji na nazwę funkcji, która została dodana w kroku “5.a” na stronie 1021. Usuń znaki komentarza, aby aktywować funkcję. Jeśli istnieje kilka funkcji, utwórz wywołania dla każdej z nich.

   c. Obok początku pliku źródłowego, pole komentarza rozpoczyna się od:

   ```
   /* Insert the function prototypes for the functions produced by */
   ```

   W tym miejscu należy wstawić instrukcje prototypu funkcji dla funkcji dodanych w kroku “3” na stronie 1020.

6. Utwórz następujący plik komend:

   ```
   cl -I  MQ_INSTALLATION_PATH\Tools\C\Include -Tp   \
   MYFORMAT.C


   MYFORMAT.DEF
   ```

   gdzie *MQ_INSTALLATION_PATH* to katalog, w którym zainstalowano produkt IBM MQ .

7. Uruchom plik komend, aby skompilować wyjście jako plik DLL.
8. Umieść dane wyjściowe w podkatalogu wyjścia poniżej katalogu danych IBM MQ . Domyślnym katalogiem używanym do instalowania wyjść w systemach 32-bitowych jest *MQ_DATA_PATH*\Exits , a w systemach 64-bitowych jest *MQ_DATA_PATH*\Exits64 .

   Ścieżka używana do wyszukiwania wyjść konwersji danych jest podana w rejestrze. Folder rejestru to:

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere
   MQ\Installation\MQ_INSTALLATION_NAME\Configuration\ClientExitPath\
   ```

   a klucz rejestru to: ExitsDefaultPath. Tę ścieżkę można ustawić dla każdego menedżera kolejek, a wyjście jest wyszukiwany tylko w tej ścieżce lub ścieżkach.

**Uwaga:**

1. Jeśli komenda CRTMQCVX używa struktur spakowanych, wszystkie aplikacje IBM MQ muszą zostać skompilowane w ten sposób.
2. Programy obsługi wyjścia konwersji danych muszą być wielobieżne.
3. MQXCNVC jest jedynym wywołaniem MQI, które można wywołać z wyjścia konwersji danych.

### Windows *Wyjdź i przełącz pliki ładowania w systemach operacyjnych Windows*

Procesy menedżera kolejek systemu IBM WebSphere MQ for Windows 7.5 są 32-bitowe. W związku z tym w przypadku korzystania z aplikacji 64-bitowych niektóre typy plików wyjścia i ładowania przełącznika XA również muszą mieć dostępną wersję 32-bitową do użycia przez menedżer kolejek. Jeśli 32-bitowa wersja pliku ładowania wyjścia lub przełącznika XA jest wymagana i nie jest dostępna, odpowiednie wywołanie funkcji API lub komenda nie powiedzie się.

W pliku `qm.ini file` dla *ExitPath*obsługiwane są dwa atrybuty. Są to `ExitsDefaultPath=` *MQ_INSTALLATION_PATH*`\exits` i `ExitsDefaultPath64=` *MQ_INSTALLATION_PATH*`\exits64`. *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ . Dzięki temu można znaleźć odpowiednią bibliotekę. Jeśli wyjście jest używane w klastrze IBM MQ , zapewnia to również znalezienie odpowiedniej biblioteki w systemie zdalnym.

Poniższa tabela zawiera listę różnych typów plików ładowania produktu Exit i Switch oraz informacje o tym, czy wymagane są wersje 32-bitowe, 64-bitowe, czy też obie te wersje, w zależności od tego, czy używane są aplikacje 32-bitowe, czy 64-bitowe:

| **Typy plików** | **Aplikacje 32-bitowe** | **Aplikacje 64-bitowe** |
|---|---|---|
| wyjście funkcji API | 32-i 64-bitowe | 64-bitowe |
| Wyjście konwersji danych | 32 bity | 64-bitowe |
| Wyjścia kanału serwera (wszystkie typy) | 64-bitowe | 64-bitowe |
| Wyjścia kanału klienta (wszystkie typy) | 32 bity | 64-bitowe |
| Wyjście instalowalnej usługi | 64-bitowe | 64-bitowe |
| Wyjście menedżera WLM klastra | 64-bitowe | 64-bitowe |
| Wyjście routingu publikowania/ subskrypcji | 64-bitowe | 64-bitowe |
| Pliki ładowania przełącznika bazy danych | 32-i 64-bitowe | 64-bitowe |
| Biblioteki AX zewnętrznego menedżera transakcji | 32 bity | 64-bitowe |
| Wyjście przed połączeniem | 32 bity | 64-bitowe |

## Odwoływanie się do definicji połączeń przy użyciu wyjścia przed nawiązaniem połączenia z repozytorium

Produkt IBM MQ MQI clients można skonfigurować w taki sposób, aby odszukać repozytorium w celu uzyskania definicji połączenia przy użyciu biblioteki wyjścia przed nawiązaniem połączenia.

### Wprowadzenie

Aplikacja kliencka może nawiązać połączenie z menedżerem kolejek przy użyciu tabel definicji kanału klienta (CCDT). Ogólnie rzecz biorąc, plik CCDT znajduje się na centralnym serwerze plików sieciowych i klienty odwołują się do niego. Ponieważ zarządzanie i administrowanie różnymi aplikacjami klienckimi odwołującymi się do pliku CCDT jest trudne, elastycznym podejściem jest przechowywanie definicji klientów w repozytorium globalnym, takim jak katalog LDAP, rejestr WebSphere i repozytorium lub inne repozytorium. Przechowywanie definicji połączeń klienckich w repozytorium ułatwia zarządzanie definicjami połączeń klienckich, a aplikacje mogą uzyskiwać dostęp do poprawnych i najbardziej aktualnych definicji połączeń klienckich.

Podczas wykonywania wywołania MQCONN/X IBM MQ MQI client ładuje określoną przez aplikację bibliotekę wyjścia przed nawiązaniem połączenia i wywołuje funkcję wyjścia w celu pobrania definicji

połączenia. Pobrane definicje połączeń są następnie używane do nawiązania połączenia z menedżerem kolejek. Szczegóły biblioteki wyjścia i funkcji do wywołania są określone w pliku konfiguracyjnym mqclient.ini .

## Składnia

```
void MQ_PRECONNECT_EXIT (pExitParms, pQMgrNazwa, ppConnectOpts, pCompppComp, pReason);
```

## Parametry

**Parametry pExit**

Typ: wejście/wyjście PMQNXP

Struktura parametru wyjścia **PreConnection** .

Struktura jest przydzielana i obsługiwana przez program wywołujący wyjście.

**pQMgrNazwa**

Typ: PMQCHAR input/output

Nazwa menedżera kolejek.

W przypadku danych wejściowych ten parametr jest łańcuchem filtru dostarczanym do wywołania funkcji API MQCONN za pośrednictwem parametru **QMgrName** . To pole może być puste, jawne lub zawierać znaki wieloznaczne. Pole jest zmieniane przez wyjście. Parametr ma wartość NULL, gdy wyjście jest wywoływane z opcją MQXR_TERM.

**Opcje ppConnect**

Typ: ppConnectOpcja wejścia/wyjścia

Opcje sterujące działaniem komendy MQCONNX.

Jest to wskaźnik do struktury opcji połączenia MQCNO, która steruje działaniem wywołania API MQCONN. Parametr ma wartość NULL, gdy wyjście jest wywoływane z opcją MQXR_TERM. Klient MQI zawsze udostępnia na potrzeby wyjścia strukturę MQCNO, nawet jeśli nie została ona pierwotnie udostępniona przez aplikację. Jeśli aplikacja udostępnia strukturę MQCNO, klient tworzy duplikat, aby przekazać go do wyjścia, w którym został zmodyfikowany. Klient zachowuje prawo własności do obiektu MQCNO.

Dysk MQCD, do którego odwołuje się obiekt MQCNO, ma pierwszeństwo przed każdą definicją połączenia udostępnioną przez tablicę. Klient używa struktury MQCNO do nawiązania połączenia z menedżerem kolejek, a pozostałe są ignorowane.

**Kod pComp**

Typ: wejście/wyjście PMQLONG

Kod zakończenia.

Wskaźnik do obiektu MQLONG, który odbiera kod zakończenia wyjść. Musi to być jedna z następujących wartości:

- MQCC_OK -pomyślne zakończenie
- MQCC_WARNING -ostrzeżenie (częściowe zakończenie)
- MQCC_FAILED -Wywołanie nie powiodło się

**pReason**

Typ: wejście/wyjście PMQLONG

Przyczyna kwalifikowania kodu pComp.

Wskaźnik do obiektu MQLONG, który odbiera kod przyczyny wyjścia. Jeśli kod zakończenia to MQCC_OK, jedyną poprawną wartością jest:

- MQRC_NONE-(0, x '000') Brak powodu do raportowania.

Jeśli kod zakończenia to MQCC_FAILED lub MQCC_WARNING, funkcja wyjścia może ustawić w polu kodu przyczyny dowolną poprawną wartość MQRC_ *.

### Wywołanie C

```
void MQ_PRECONNECT_EXIT (&ExitParms, &QMgrName, &pConnectOpts, &CompCode, &Reason);
```

**Parameter**

```
PMQNXP  pExitParms   /*PreConnect exit parameter structure*/
PMQCHAR pQMgrName    /*Name of the queue manager*/
PPMQCNO ppConnectOpts/*Options controlling the action of MQCONNX*/
PMQLONG pCompCode    /*Completion code*/
PMQLONG pReason      /*Reason qualifying pCompCode*/
```

## Zapisywanie i kompilowanie wyjść publikowania

Istnieje możliwość skonfigurowania wyjścia publikowania w menedżerze kolejek w celu zmiany treści opublikowanego komunikatu przed jego odebraniem przez subskrybentów. Można również zmienić nagłówek komunikatu lub nie można dostarczyć komunikatu do subskrypcji.

**Uwaga:** Procedury zewnętrzne publikowania nie są obsługiwane w systemie z/OS.

Za pomocą wyjścia publikowania można sprawdzać i zmieniać komunikaty dostarczane do subskrybentów:

- Sprawdzanie treści komunikatu opublikowanego dla każdego subskrybenta
- Modyfikowanie treści komunikatu opublikowanego dla każdego subskrybenta
- Zmiana kolejki, w której umieszczany jest komunikat
- Zatrzymaj dostarczanie komunikatu do subskrybenta

### Zapisywanie wyjścia publikowania

Wykonaj kroki opisane w sekcji “Zapisywanie wyjść i usług instalowalnych w systemie AIX, Linux, and Windows” na stronie 965, aby pomóc w napisaniu i skompilowaniu wyjścia.

Dostawca wyjścia publikowania definiuje działanie wyjścia. Jednak wyjście musi być zgodne z regułami zdefiniowanymi w MQPSXP.

Produkt IBM MQ nie udostępnia implementacji punktu wejścia MQ_PUBLISH_EXIT. Zawiera on deklarację definicji typu języka C. Użyj parametru typedef, aby poprawnie zadeklarować parametry w zapisanym przez użytkownika wyjściu. W poniższym przykładzie przedstawiono sposób użycia deklaracji typedef:

```
#include "cmqec.h"

MQ_PUBLISH_EXIT MyPublishExit;

void MQENTRY MyPublishExit( PMQPSXP pExitParms,
                            PMQPBC  pPubContext,
                            PMQSBC  pSubContext )
{
/* C language statements to perform the function of the exit */
}
```

Wyjście publikowania jest uruchamiane w procesie menedżera kolejek w wyniku następujących operacji:

- Operacja publikowania, w której komunikat jest dostarczany do jednego lub większej liczby subskrybentów
- Operacja subskrypcji, w której dostarczany jest co najmniej jeden zachowany komunikat
- Operacja żądania subskrypcji, w której dostarczany jest co najmniej jeden zachowany komunikat

Jeśli wyjście publikowania jest wywoływane dla połączenia, przy pierwszym jego wywołaniu ustawiany jest kod *ExitReason* o wartości MQXR_INIT . Przed rozłączeniem połączenia po użyciu wyjścia publikowania wyjście jest wywoływane z kodem *ExitReason* MQXR_TERM.

Jeśli wyjście publikowania jest skonfigurowane, ale nie można go załadować podczas uruchamiania menedżera kolejek, operacje komunikatów publikowania/subskrypcji są zablokowane dla menedżera kolejek. Należy rozwiązać ten problem lub zrestartować menedżer kolejek przed ponownym włączeniem przesyłania komunikatów w trybie publikowania/subskrypcji.

Każde połączenie IBM MQ , które wymaga wyjścia publikowania, może nie załadować lub zainicjować wyjścia. Jeśli ładowanie lub inicjowanie wyjścia nie powiedzie się, operacje publikowania/subskrypcji, które wymagają wyjścia publikowania, są wyłączone dla tego połączenia. Operacje kończą się niepowodzeniem z IBM MQ kodem przyczyny `MQRC_PUBLISH_EXIT_ERROR`.

Kontekst, w którym wywoływane jest wyjście publikowania, jest połączeniem aplikacji z menedżerem kolejek. Obszar danych użytkownika jest obsługiwany przez menedżer kolejek dla każdego połączenia, które wykonuje operacje publikowania. Wyjście może przechowywać informacje w obszarze danych użytkownika dla każdego połączenia.

Wyjście publikowania może korzystać z niektórych wywołań MQI. Może on używać tylko tych wywołań MQI, które manipulują właściwościami komunikatu. Wywołania są następujące:

- MQBUFMH
- MQCRTMH
- MQDLTMH
- Komenda MQDLTMP
- MQMHBUF
- MQINQMP
- Komenda MQSETMP

Jeśli wyjście publikowania zmienia nazwę docelowego menedżera kolejek lub nazwę kolejki, nie jest wykonywane nowe sprawdzanie uprawnień.

## Kompilowanie wyjścia publikowania

Wyjście publikowania jest biblioteką ładowaną dynamicznie; można je traktować jako wyjście kanału. Informacje na temat kompilowania wyjść zawiera sekcja “Zapisywanie wyjść i usług instalowalnych w systemie AIX, Linux, and Windows” na stronie 965.

## Przykładowe wyjście publikowania

Przykładowy program obsługi wyjścia nosi nazwę `amqspse0.c`. Zapisuje on inny komunikat w pliku dziennika w zależności od tego, czy wyjście zostało wywołane dla operacji inicjowania, publikowania lub zakończenia. Zademonstrowano również użycie pola obszaru użytkownika programu zewnętrznego w celu odpowiedniego przydzielenia i zwolnienia pamięci.

### *Konfigurowanie wyjść publikowania*

Aby skonfigurować wyjście publikowania, należy zdefiniować pewne atrybuty.

W systemach Windows i Linux do zdefiniowania atrybutów można użyć eksploratora IBM MQ . Atrybuty są definiowane na stronie właściwości menedżera kolejek w obszarze Publikowanie/subskrypcja.

Aby skonfigurować wyjście publikowania w pliku `qm.ini` w systemach AIX and Linux , utwórz sekcję o nazwie `PublishSubscribe`. Sekcja `PublishSubscribe` ma następujące atrybuty:

**PublishExitPath = [ ścieżka] |*nazwa_modułu***
Nazwa modułu i ścieżka zawierająca kod wyjścia publikowania. Maksymalna długość tego pola to `MQ_EXIT_NAME_LENGTH`. Wartością domyślną jest brak wyjścia publikowania.

**PublishExitFunction = *nazwa_funkcji***
Nazwa punktu wejścia funkcji w module, który zawiera kod wyjścia publikowania. Maksymalna długość tego pola to `MQ_EXIT_NAME_LENGTH`.

IBM i W systemie IBM i, jeśli używany jest program, pomiń `PublishExitFunction`.

**PublishExitData = *łańcuch***

Jeśli menedżer kolejek wywołuje wyjście publikowania, przekazuje strukturę MQPSXP jako dane wejściowe. Dane określone za pomocą atrybutu **PublishExitData** są udostępniane w polu *ExitData* struktury. Łańcuch może mieć długość do MQ_EXIT_DATA_LENGTH znaków. Wartością domyślną są 32 puste znaki.

## Zapisywanie i kompilowanie wyjść obciążenia klastra

Napisz program obsługi wyjścia obciążenia klastra, aby dostosować zarządzanie obciążeniem klastrów. Podczas kierowania komunikatów można wziąć pod uwagę koszt używania kanału o różnych porach dnia lub treść wiadomości. Są to czynniki, które nie są uwzględniane przez standardowy algorytm zarządzania obciążeniem.

W większości przypadków algorytm zarządzania obciążeniem jest wystarczający do potrzeb użytkownika. Jednak aby można było udostępnić własny program obsługi wyjścia użytkownika w celu dostosowania zarządzania obciążeniem, produkt IBM MQ zawiera program obsługi wyjścia użytkownika, czyli program obsługi wyjścia obciążenia klastra.

Mogą być dostępne konkretne informacje o sieci lub komunikaty, które mogą wpłynąć na równoważenie obciążenia. Użytkownik może wiedzieć, które z nich są kanałami o dużej pojemności lub tanimi trasami sieciowymi, lub może chcieć kierować komunikaty w zależności od ich treści. Można napisać program obsługi wyjścia obciążenia klastra lub użyć programu dostarczonego przez inną firmę.

Wyjście obciążenia klastra jest wywoływane podczas uzyskiwania dostępu do kolejki klastra. Jest wywoływana przez MQOPEN, MQPUT1 i MQPUT.

Docelowy menedżer kolejek wybrany o godzinie MQOPEN jest stały, jeśli określono parametr MQOO_BIND_ON_OPEN . W takim przypadku wyjście jest uruchamiane tylko raz.

Jeśli docelowy menedżer kolejek nie jest ustalony w czasie MQOPEN , docelowy menedżer kolejek jest wybierany w czasie wywołania funkcji MQPUT . Jeśli docelowy menedżer kolejek nie jest dostępny lub jego działanie kończy się niepowodzeniem, gdy komunikat nadal znajduje się w kolejce transmisji, wyjście jest ponownie wywoływane. Wybrano nowy docelowy menedżer kolejek. Jeśli kanał komunikatów zakończy się niepowodzeniem podczas przesyłania komunikatu i komunikat zostanie wycofany, zostanie wybrany nowy docelowy menedżer kolejek.

Multi W systemie Wiele platformmenedżer kolejek ładuje nowe wyjście obciążenia klastra przy następnym uruchomieniu menedżera kolejek.

Jeśli definicja menedżera kolejek nie zawiera nazwy programu obsługi wyjścia obciążenia klastra, program obsługi wyjścia obciążenia klastra nie jest wywoływany.

Różne dane są przekazywane do wyjścia obciążenia klastra w strukturze parametru wyjścia ( MQWXP):

- Struktura definicji komunikatu MQMD.
- Parametr długości komunikatu.
- Kopia komunikatu lub jego część.

Na platformach innych niżz/OS , jeśli używany jest system CLWLMode=FAST, każdy proces systemu operacyjnego ładuje własną kopię wyjścia. Różne połączenia z menedżerem kolejek mogą spowodować wywołanie różnych kopii wyjścia. Jeśli wyjście jest uruchamiane w domyślnym trybie bezpiecznym ( CLWLMode=SAFE), pojedyncza kopia wyjścia jest uruchamiana w oddzielnym procesie.

## Zapisywanie wyjść obciążenia klastra

z/OS Informacje na temat zapisywania wyjść obciążenia klastra dla produktu z/OSzawiera sekcja “Programowanie wyjścia obciążenia klastra dla systemu IBM MQ for z/OS” na stronie 1028.

W produkcie IBM MQ 9.1.0wyjścia obciążenia klastra są uruchamiane w przestrzeni adresowej inicjatora kanału zamiast w przestrzeni adresowej menedżera kolejek. Jeśli istnieje wyjście obciążenia klastra, należy usunąć instrukcję CSQXLIB DD z procedury uruchomionego zadania menedżera kolejek

i dodać zestaw danych zawierający wyjście obciążenia klastra do konkatenacji CSQXLIB w procedurze uruchomionego zadania inicjatora kanału.

Multi W przypadku wielu platform wyjścia obciążenia klastra nie mogą używać wywołań MQI. Pod innymi względami reguły zapisywania i kompilowania programów obsługi wyjścia obciążenia klastra są podobne do reguł mających zastosowanie do programów obsługi wyjścia kanału. Wykonaj kroki opisane w sekcji “Zapisywanie wyjść i usług instalowalnych w systemie AIX, Linux, and Windows” na stronie 965i użyj przykładowego programu “Przykładowe wyjście obciążenia klastra” na stronie 1027 , aby napisać i skompilować wyjście.

Więcej informacji na temat wyjść kanałów zawiera sekcja “Zapisywanie programów obsługi wyjścia kanału” na stronie 993.

## Konfigurowanie wyjść obciążenia klastra

Wyjścia obciążenia klastra można nazwać w definicji menedżera kolejek, podając atrybut wyjścia obciążenia klastra w komendzie `ALTER QMGR` . Na przykład:

```
ALTER QMGR CLWLEXIT(myexit)
```

**Odsyłacze pokrewne**

Wywołania wyjścia obciążenia klastra i struktury danych

### *Przykładowe wyjście obciążenia klastra*

IBM MQ zawiera przykładowy program obsługi wyjścia obciążenia klastra. Można skopiować przykład i użyć go jako podstawy dla własnych programów.

z/OS **IBM MQ for z/OS**

Przykładowy program obsługi wyjścia obciążenia klastra jest dostarczany w asemblerze i w programie C. Wersja asemblera nosi nazwę CSQ4BAF1 i znajduje się w bibliotece `thlqual.SCSQASMS`. Wersja bazy danych C nosi nazwę CSQ4BCF1 i znajduje się w bibliotece `thlqual.SCSQC37S`. `thlqual` jest kwalifikatorem wysokiego poziomu biblioteki docelowej dla zestawów danych IBM MQ w danej instalacji.

Multi **IBM MQ for Multiplatforms**

Przykładowy program obsługi wyjścia obciążenia klastra jest dostarczany w języku C i nosi nazwę `amqswlm0.c`. Można go znaleźć w:

*Tabela 144. Przykładowe położenie programu obsługi wyjścia obciążenia klastra dla wielu platform*

| **Platforma** | **Ścieżka do pliku** |
|---|---|
| AIX AIX | *MQ_INSTALLATION_PATH*`/samp` |
| Windows Windows | *MQ_INSTALLATION_PATH* `\Tools\c\Samples` |
| IBM i IBM i | Biblioteka `qmqm` |

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

To przykładowe wyjście kieruje wszystkie komunikaty do konkretnego menedżera kolejek, chyba że ten menedżer kolejek stanie się niedostępny. Reaguje on na awarię menedżera kolejek, kierując komunikaty do innego menedżera kolejek.

Wskaż, do którego menedżera kolejek mają być wysyłane komunikaty. Podaj nazwę kanału odbiorczego klastra w atrybucie CLWLDATA w definicji menedżera kolejek. Na przykład:

```
ALTER QMGR CLWLDATA(' my-cluster-name. my-queue-manager ')
```

Aby włączyć wyjście, należy podać jego pełną ścieżkę i nazwę w atrybucie CLWLEXIT :

Linux AIX W systemie AIX and Linux:

```
ALTER QMGR CLWLEXIT(' path /amqswlm(clwlFunction)')
```

Windows W systemie Windows:

```
ALTER QMGR CLWLEXIT(' path \amqswlm(clwlFunction)')
```

z/OS W systemie z/OS:

```
ALTER QMGR CLWLEXIT(CSQ4BxF1)
```

gdzie x to 'A' lub 'C', w zależności od używanego języka programowania.

IBM i W systemie IBM iużyj jednej z następujących komend:

- Użyj komendy MQSC:

  ```
  ALTER QMGR CLWLEXIT('AMQSWLM       library       ')
  ```

  Zarówno nazwa programu, jak i nazwa biblioteki zajmują 10 znaków i w razie potrzeby są dopełniane odstępem w prawo.
- Użyj komendy CL:

  ```
  CHGMQM MQMNAME( qmgrname ) CLWLEXIT(' library /AMQSWLM')
  ```

Teraz zamiast używać dostarczonego algorytmu zarządzania obciążeniem, program IBM MQ wywołuje to wyjście w celu skierowania wszystkich komunikatów do wybranego menedżera kolejek.

### z/OS *Programowanie wyjścia obciążenia klastra dla systemu IBM MQ for z/OS*

Wyjścia obciążenia klastra są wywoływane tak, jak za pomocą komendy z/OS **LINK** . Wyjścia podlegają wielu rygorystycznym zasadom programowania. Należy unikać używania większości komend SVC, które wymagają oczekiwania, lub używania STAE lub ESTAE w wyjściu obciążenia.

Wyjścia obciążenia klastra są wywoływane przez z/OS **LINK** w następujący sposób:

- Nieautoryzowany stan programu problemu
- Tryb sterowania podstawową przestrzenią adresową
- Tryb inny niż międzypamięciowy
- Tryb rejestru bez dostępu
- 31-bitowy tryb adresowania
- Klucz pamięci 8
- Maska klucza programu 8
- Klucz TCB 8

Umieść moduły edytowane przez łącze w zestawie danych określonym przez instrukcję CSQXLIB DD procedury uruchomionego zadania inicjatora kanału. Nazwy modułów ładowania są określone jako nazwy wyjść obciążenia w definicji menedżera kolejek.

Podczas zapisywania wyjść obciążenia dla IBM MQ for z/OSobowiązują następujące reguły:

- Należy zapisać wyjścia w asemblerze lub C. Jeśli używany jest kod w języku C, musi on być zgodny ze środowiskiem programistycznym systemu C dla wyjść systemowych opisanym w podręczniku *z/OS C/C++ Programming Guide*, SC09-4765.
- Jeśli używane jest wywołanie MQXCLWLN , należy połączyć edycję z wartością CSQMFCLW, która jest podana w pliku *thlqual*.SCSQLOAD.
- Wyjścia są ładowane z nieautoryzowanych bibliotek zdefiniowanych przez instrukcję CSQXLIB DD . Jeśli właściwość CSQXLIB ma wartość DISP=SHR, wyjścia mogą być aktualizowane podczas działania menedżera kolejek przy użyciu nowej wersji używanej w następnym wątku MQCONN uruchamianym przez menedżer kolejek.
- Wyjścia muszą być wielobieżne i mogą działać w dowolnym miejscu wirtualnej pamięci masowej.
- Wyjścia muszą resetować środowisko po powrocie do pozycji.
- Wyjścia muszą zwolnić każdą uzyskaną pamięć lub zapewnić, że pamięć zostanie zwolniona przez kolejne wywołanie wyjścia.
- Wywołania MQI nie są dozwolone.
- Wyjścia nie mogą używać żadnych usług systemowych, które mogą spowodować oczekiwanie, ponieważ oczekiwanie poważnie obniża wydajność menedżera kolejek. W związku z tym należy unikać urządzeń SVC, PC lub we/wy.
- Wyjścia nie mogą wystawiać ESTAE ani SPIE, z wyjątkiem przyłączanych podzadań.

**Uwaga:** Nie ma bezwzględnych ograniczeń co do tego, co można zrobić w wyjściu. Jednak większość kanałów SVC wymaga oczekiwania, dlatego należy ich unikać, z wyjątkiem następujących komend:

- **GETMAIN** / **FREEMAIN**
- **LOAD** / **DELETE**

Nie należy używać ESTAE i ESPIEs, ponieważ ich obsługa błędów może kolidować z obsługą błędów wykonywaną przez IBM MQ. Program IBM MQ może nie być w stanie naprawić błędu lub program obsługi wyjścia może nie otrzymać wszystkich informacji o błędzie.

Parametr systemowy EXITLIM ogranicza czas, przez jaki może działać wyjście. Wartością domyślną parametru EXITLIM jest 30 sekund. Jeśli zostanie wyświetlony kod powrotu MQRC_CLUSTER_EXIT_ERROR, 2266 X'8DA' , wyjście może być zapętlone. Jeśli zakończenie wyjścia wymaga więcej niż 30 sekund, zwiększ wartość EXITLIM.

# Budowanie aplikacji proceduralnej

Użytkownik może napisać aplikację IBM MQ w jednym z kilku języków proceduralnych i uruchomić ją na kilku różnych platformach.

## AIX Budowanie aplikacji proceduralnej w systemie AIX

Publikacje AIX zawierają opis sposobu budowania aplikacji wykonywalnych na podstawie napisanych programów.

W tym temacie opisano dodatkowe zadania oraz zmiany w zadaniach standardowych, które należy wykonać podczas budowania aplikacji IBM MQ for AIX w celu ich uruchomienia w systemie AIX. Obsługiwane są języki C, C + + i COBOL. Informacje na temat przygotowywania programów w języku C++ zawiera sekcja Korzystanie z języka C++.

Zadania, które należy wykonać, aby utworzyć aplikację wykonywalną za pomocą programu IBM MQ for AIX , różnią się w zależności od języka programowania, w którym napisany jest kod źródłowy. Oprócz kodowania wywołań MQI w kodzie źródłowym należy dodać odpowiednie instrukcje języka, aby

uwzględnić pliki włączane IBM MQ for AIX dla używanego języka. Zapoznaj się z zawartością tych plików. Pełny opis znajduje się w sekcji “Pliki definicji danych IBM MQ” na stronie 741 .

Podczas uruchamiania wielowątkowych aplikacji serwera lub wielowątkowych aplikacji klienckich należy ustawić zmienną środowiskową AIXTHREAD_SCOPE = S.

### AIX *Przygotowywanie programów w języku C w systemie AIX*

Ten temat zawiera informacje o dowiązywaniu bibliotek niezbędnych do przygotowania programów w języku C w systemie AIX.

Prekompilowane programy w języku C są dostarczane w katalogu *MQ_INSTALLATION_PATH*/samp/bin . Użyj kompilatora ANSI i uruchom następujące komendy. Więcej informacji na temat programowania aplikacji 64-bitowych zawiera sekcja Standardy kodowania na platformach 64-bitowych.

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Dla aplikacji 32-bitowych:

```
$ xlc_r -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm
```

gdzie amqsput0 jest programem przykładowym.

Dla aplikacji 64-bitowych:

```
$ xlc_r -q64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm
```

gdzie amqsput0 jest programem przykładowym.

V 9.3.5 Dla aplikacji 32-bitowych używających kompilatora XLC 17:

```
$ ibm-clang -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-lmqm
```

gdzie amqsput0 jest programem przykładowym.

V 9.3.5 W przypadku aplikacji 64-bitowych korzystających z kompilatora XLC 17:

```
$ ibm-clang -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/
lib64 -lmqm
```

gdzie amqsput0 jest programem przykładowym.

Jeśli używany jest kompilator VisualAge C/C++ dla programów w języku C + +, należy dołączyć opcję -q namemangling=v5 , aby podczas dowiązywania bibliotek zostały przetłumaczone wszystkie symbole IBM MQ .

Jeśli chcesz używać programów na komputerze, na którym jest zainstalowany tylko system IBM MQ MQI client for AIX , zrekompiluj programy, aby połączyć je z biblioteką klienta (-lmqic).

## Dowiązywanie bibliotek

Potrzebne są następujące biblioteki:

- Połącz programy z odpowiednią biblioteką udostępnioną przez IBM MQ.

  W środowisku niewielowątkowym należy utworzyć odsyłacz do jednej z następujących bibliotek:

| Plik biblioteki | Typ programu/wyjścia |
|---|---|
| libmqm.a | Serwer dla języka C |
| libmqic.a & libmqm.a | Klient dla języka C |

W środowisku wielowątkowym należy utworzyć odsyłacz do jednej z następujących bibliotek:

| Plik biblioteki | Typ programu/wyjścia |
|---|---|
| libmqm_r.a | Serwer dla języka C |
| libmqic_r.a & libmqm_r.a | Klient dla języka C |

Na przykład, aby zbudować prostą wielowątkową aplikację IBM MQ na podstawie pojedynczej jednostki kompilacji, należy uruchomić następujące komendy.

Dla aplikacji 32-bitowych:

```
$ xlc_r -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm_r
```

gdzie `amqsput0` jest programem przykładowym.

Dla aplikacji 64-bitowych:

```
$ xlc_r -q64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -lmqm_r
```

gdzie `amqsput0` jest programem przykładowym.

V 9.3.5 Dla aplikacji 32-bitowych używających kompilatora XLC 17:

```
$ ibm-clang_r -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib -lmqm_r
```

gdzie `amqsput0` jest programem przykładowym.

V 9.3.5 W przypadku aplikacji 64-bitowych korzystających z kompilatora XLC 17:

```
$ ibm-clang_r -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib64 -lmqm_r
```

gdzie `amqsput0` jest programem przykładowym.

Jeśli chcesz używać programów na komputerze, na którym jest zainstalowany tylko system IBM MQ MQI client for AIX , zrekompiluj programy, aby połączyć je z biblioteką klienta (`-lmqic`).

**Uwaga:**

1. Nie można utworzyć odsyłacza do więcej niż jednej biblioteki. Oznacza to, że nie można jednocześnie utworzyć dowiązania do biblioteki wielowątkowej i niewielowątkowej.
2. W przypadku tworzenia instalowalnej usługi (więcej informacji na ten temat zawiera publikacja Administrowanie programem IBM MQ ) należy utworzyć odsyłacz do biblioteki `libmqmzf.a` w aplikacji niewielowątkowej oraz do biblioteki `libmqmzf_r.a` w aplikacji wielowątkowej.
3. Jeśli aplikacja jest generowana na potrzeby koordynacji zewnętrznej przez menedżer transakcji zgodny z interfejsem XA, taki jak IBM TXSeries, Encina lub BEA Tuxedo, należy połączyć się z serwerem `libmqmxa.a` (lub `libmqmxa64.a` , jeśli menedżer transakcji traktuje typ "long" jako 64-bitowy) i bibliotekami `libmqz.a` w aplikacji niewielowątkowej oraz z serwerem `libmqmxa_r.a` (lub `libmqmxa64_r.a` ). i bibliotek `libmqz_r.a` w aplikacji wielowątkowej.
4. Należy połączyć zaufane aplikacje z wątkowymi bibliotekami IBM MQ . W tym samym czasie można jednak połączyć tylko jeden wątek w zaufanej aplikacji w systemach IBM MQ for AIX or Linux .
5. Należy dowiązać biblioteki IBM MQ przed wszystkimi innymi bibliotekami produktu.

## AIX *Przygotowywanie programów w języku COBOL w systemie AIX*

Tych informacji należy użyć podczas przygotowywania programów w języku COBOL w systemie AIX przy użyciu zestawów IBM COBOL i Micro Focus COBOL.

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

- 32-bitowe książki kopii w języku COBOL są instalowane w następującym katalogu:

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

  i dowiązania symboliczne są tworzone w:

```
MQ_INSTALLATION_PATH/inc
```

- 64-bitowe książki w języku COBOL są instalowane w następującym katalogu:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

W poniższych przykładach należy ustawić zmienną środowiskową **COBCPY** na wartość:

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

dla aplikacji 32-bitowych oraz:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

dla aplikacji 64-bitowych.

Należy połączyć program z jednym z następujących plików biblioteki:

| Plik biblioteki | Typ programu/wyjścia |
|---|---|
| libmqmcb.a | Server for COBOL (aplikacja niewątkowa) |
| libmqmcb_r.a | Server for COBOL (aplikacja wielowątkowa) |
| libmqicb.a | Client for COBOL (aplikacja niewątkowa) |
| libmqicb_r.a | Client for COBOL (aplikacja wielowątkowa) |

W zależności od programu można użyć kompilatora IBM COBOL Set lub kompilatora Micro Focus COBOL:

- Programy rozpoczynające się od `amqm` są odpowiednie dla kompilatora Micro Focus COBOL oraz
- Programy rozpoczynające się od `amq0` są odpowiednie dla każdego kompilatora.

## Przygotowywanie programów w języku COBOL za pomocą programu IBM COBOL Set for AIX

Przykładowe programy w języku COBOL są dostarczane z produktem IBM MQ. Aby skompilować taki program, należy wprowadzić odpowiednią komendę z następującej listy:

**32-bitowa aplikacja serwera niewielowątkowego**

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb -qLIB \
-ICOBCPY_VALUE
```

**32-bitowa niewątkowa aplikacja kliencka**

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-ICOBCPY_VALUE
```

**32-bitowa aplikacja serwera z wątkami**

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

**32-bitowa aplikacja kliencka z wątkami**

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

**64-bitowa aplikacja serwera niewielowątkowego**

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqmcb \
-qLIB -ICOBCPY_VALUE
```

**64-bitowa niewątkowa aplikacja kliencka**

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqicb \
-qLIB -ICOBCPY_VALUE
```

**Aplikacja serwera z wątkami 64-bitowymi**

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

**Aplikacja kliencka z wątkami 64-bitowymi**

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

## Przygotowywanie programów w języku COBOL przy użyciu programu Micro Focus COBOL

Przed skompilowaniem programu ustaw zmienne środowiskowe w następujący sposób:

```
export COBCPY=COBCPY_VALUE
export LIBPATH=MQ_INSTALLATION_PATH/lib:$LIBPATH
```

Aby skompilować 32-bitowy program w języku COBOL przy użyciu programu Micro Focus COBOL, należy wpisać:

- Serwer dla języka COBOL

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb
```

- Klient dla języka COBOL

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb
```

- Serwer wielowątkowy dla języka COBOL

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r
```

- Wątkowy klient dla języka COBOL

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r
```

Aby skompilować 64-bitowy program w języku COBOL przy użyciu programu Micro Focus COBOL, należy wpisać:

- Serwer dla języka COBOL

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb
```

- Klient dla języka COBOL

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb
```

- Serwer wielowątkowy dla języka COBOL

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r
```

- Wątkowy klient dla języka COBOL

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r
```

gdzie `amqminqx` jest programem przykładowym

Opis zmiennych środowiskowych, które należy skonfigurować, można znaleźć w dokumentacji języka Micro Focus COBOL.

### AIX *Przygotowywanie aplikacji CICS w systemie AIX*

Te informacje są przydatne podczas przygotowywania programów CICS w systemie AIX.

Użyj modułów *przełącznika XA* , aby połączyć produkt CICS z produktem IBM MQ. Więcej informacji na temat struktury przełącznika XA zawiera sekcja Struktury przełącznika XA.

Udostępniono przykładowy plik kodu źródłowego, który umożliwia tworzenie przełączników XA dla innych komunikatów transakcji. Nazwa udostępnionego modułu ładowalnego przełącznika jest wymieniona w sekcjiTabela 145 na stronie 1034.

*Tabela 145. Podstawowy kod dla aplikacji CICS w systemie AIX: procedura inicjowania XA*

| **Opis** | **C (źródło)** | **C (exec)-dodaj do XAD.Stanza** |
|---|---|---|
| Procedura inicjowania XA | amqzscix.c | *amqzsc* - CICS dla systemu AIX |

Należy użyć wstępnie zbudowanej wersji pliku ładowania przełącznika IBM MQ *amqzsc*, który jest dostarczany z produktem.

Zawsze należy dowiązywać transakcje C z wątkowo bezpieczną IBM MQ biblioteką *libmqm_r.a.*, i transakcje w języku COBOL z biblioteką języka COBOL *libmqmcb_r.a.*.

Więcej informacji na temat obsługi transakcji CICS można znaleźć w podręczniku Administrowanie programem IBM MQ IBM MQ System Administration Guide.

#### AIX *Wsparcie dla produktu TXSeries CICS*

Produkt IBM MQ w systemie AIX obsługuje produkt TXSeries CICS przy użyciu interfejsu XA. Upewnij się, że aplikacje CICS są połączone z wątkową wersją bibliotek IBM MQ .

Programy CICS można uruchamiać za pomocą IBM COBOL Set for AIX lub Micro Focus COBOL. W poniższych sekcjach opisano różnicę między uruchamianiem programów CICS w języku IBM COBOL Set for AIX i Micro Focus COBOL.

Napisz programy IBM MQ , które są ładowane do tego samego regionu CICS w języku C lub COBOL. Nie można tworzyć kombinacji wywołań MQI języka C i COBOL do tego samego regionu CICS . Większość wywołań MQI w drugim używanym języku kończy się niepowodzeniem z kodem przyczyny MQRC_HOBJ_ERROR.

## Przygotowywanie programów w języku CICS COBOL za pomocą programu IBM COBOL Set for AIX

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Aby użyć języka COBOL IBM , wykonaj następujące kroki:

1. Wyeksportuj następującą zmienną środowiskową:

   ```
   export LDFLAGS="-qLIB -bI:/usr/lpp/cics/lib/cicsprIBMCOB.exp \
                   -I MQ_INSTALLATION_PATH/inc -I/usr/lpp/cics/include \
                   -e _iwz_cobol_main  \
   ```

   gdzie LIB jest dyrektywą kompilatora.
2. Przetłumacz, skompiluj i dowiąż program, wpisując:

   ```
   cicstcl -l IBMCOB yourprog.ccp
   ```

## Przygotowywanie programów w języku COBOL dla systemu CICS przy użyciu programu Micro Focus COBOL

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Aby użyć programu Micro Focus COBOL, należy wykonać następujące czynności:

1. Dodaj moduł biblioteki środowiska wykonawczego COBOL IBM MQ do biblioteki środowiska wykonawczego, używając następującej komendy:

   ```
   cicsmkcobol -L/usr/lib/dce -L MQ_INSTALLATION_PATH/lib \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
   ```

   **Uwaga:** W systemie `cicsmkcobol`produkt IBM MQ nie zezwala na wykonywanie wywołań MQI w języku programowania C z aplikacji w języku COBOL.

   Jeśli istniejące aplikacje mają takie wywołania, zaleca się przeniesienie tych funkcji z aplikacji w języku COBOL do własnej biblioteki, na przykład `myMQ.so`. Po przeniesieniu funkcji nie należy uwzględniać IBM MQ biblioteki `libmqmcbrt.o` podczas budowania aplikacji w języku COBOL dla systemu CICS.

   Ponadto, jeśli aplikacja w języku COBOL nie wykonuje żadnego wywołania MQI języka COBOL, nie należy łączyć produktu `libmqmz_r` z produktem `cicsmkcobol`.

   Spowoduje to utworzenie pliku metody języka Micro Focus COBOL i umożliwi wywoływanie systemów IBM MQ for AIX or Linux przez bibliotekę wykonawczą języka COBOL CICS .

   **Uwaga:** Uruchom program `cicsmkcobol` tylko wtedy, gdy instalowany jest jeden z następujących produktów:

   - Nowa wersja lub wydanie programu Micro Focus COBOL
   - Nowa wersja lub wydanie produktu CICS for AIX
   - Nowa wersja lub wydanie dowolnego obsługiwanego produktu bazodanowego (tylko dla transakcji COBOL)
   - Nowa wersja lub wydanie systemu IBM MQ

2. Wyeksportuj następującą zmienną środowiskową:

```
COBCPY= MQ_INSTALLATION_PATH/inc export COBCPY
```

3. Przetłumacz, skompiluj i dowiąż program, wpisując:

```
cicstcl -l COBOL -e yourprog.ccp
```

### Przygotowywanie programów w języku C dla systemu CICS

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Budowanie programów w języku C dla systemu CICS przy użyciu standardowych narzędzi systemu CICS :

1. Wyeksportuj ***jedną*** z następujących zmiennych środowiskowych:
   - LDFLAGS = "-L/ *MQ_INSTALLATION_PATH* lib -lmqm_r" export LDFLAGS
   - USERLIB = "-L *MQ_INSTALLATION_PATH* lib -lmqm_r" export USERLIB
2. Przetłumacz, skompiluj i dowiąż program, wpisując:

```
cicstcl -l C amqscic0.ccs
```

#### CICS Przykładowa transakcja C

Przykładowe źródło w języku C dla transakcji produktu AIX IBM MQ jest udostępniane przez produkt AMQSCIC0.CCS. Transakcja odczytuje komunikaty z kolejki transmisji SYSTEM.SAMPLE.CICS.WORKQUEUE w domyślnym menedżerze kolejek i umieszcza je w kolejce lokalnej z nazwą kolejki, która jest zawarta w nagłówku transmisji komunikatu. Wszystkie niepowodzenia są wysyłane do kolejki SYSTEM.SAMPLE.CICS.DLQ. Użyj przykładowego skryptu MQSC AMQSCIC0.TST , aby utworzyć te kolejki i przykładowe kolejki wejściowe.

## IBM i Budowanie aplikacji proceduralnej w systemie IBM i

Publikacje IBM i zawierają opis sposobu budowania aplikacji wykonywalnych na podstawie programów napisanych przez użytkownika, w celu ich uruchomienia w systemie IBM i w systemie iSeries lub System i .

W tym temacie opisano dodatkowe zadania oraz zmiany w zadaniach standardowych, które należy wykonać podczas budowania aplikacji proceduralnych IBM MQ for IBM i uruchamianych w systemach IBM i . Obsługiwane są języki programowania COBOL, C + +, Java i RPG. Informacje na temat przygotowywania programów w języku C++ zawiera sekcja Korzystanie z języka C++. Informacje na temat przygotowywania programów Java zawiera sekcja Korzystanie z programu IBM MQ classes for Java.

Zadania, które należy wykonać, aby utworzyć wykonywalną aplikację IBM MQ for IBM i , zależą od języka programowania, w którym napisany jest kod źródłowy. Oprócz kodowania wywołań MQI w kodzie źródłowym należy dodać odpowiednie instrukcje języka, aby uwzględnić pliki definicji danych programu IBM MQ for IBM i dla używanego języka. Zapoznaj się z zawartością tych plików. Pełny opis znajduje się w sekcji “Pliki definicji danych IBM MQ” na stronie 741 .

### IBM i *Przygotowywanie programów w języku C w systemie IBM i*

Produkt IBM MQ for IBM i obsługuje komunikaty o wielkości do 100 MB. Aplikacje napisane w języku ILE C, obsługujące komunikaty IBM MQ większe niż 16 MB, muszą użyć opcji kompilatora `Teraspace` , aby przydzielić wystarczającą ilość pamięci dla tych komunikatów.

Więcej informacji na temat opcji kompilatora C zawiera publikacja *WebSphere Development Studio ILE C/C++ Programmer's Guide*.

Aby skompilować moduł C, można użyć IBM i komendy **CRTCMOD**. Upewnij się, że biblioteka zawierająca zbiory włączane (QMQM) znajduje się na liście bibliotek podczas kompilacji.

Następnie należy powiązać dane wyjściowe kompilatora z programem usługowym za pomocą komendy **CRTPGM** .

*Tabela 146. Przykład komendy CRTPGM w środowiskach niewielowątkowych i wielowątkowych*

| Typ środowiska | Komenda | Typ programu/wyjścia |
|---|---|---|
| Środowisko niewątkowe | `CRTPGM PGM( pgmname ) MODULE( pgmname ) BNDSRVPGM(QMQM/LIBMQM)` | Serwer lub klient dla języka C |
| Środowisko wielowątkowe | `CRTPGM PGM( pgmname ) MODULE( pgmname ) BNDSRVPGM(QMQM/LIBMQM_R)` | Serwer lub klient dla języka C |

gdzie *pgmname* jest nazwą programu.

Tabela 147 na stronie 1037 zawiera listę bibliotek, które są potrzebne podczas przygotowywania programów w języku C w systemie IBM i w środowisku niewielowątkowym i środowisku wielowątkowym.

*Tabela 147. Biblioteki wymagane dla środowisk niewielowątkowych i wielowątkowych*

| Typ środowiska | Plik biblioteki | Typ programu/wyjścia |
|---|---|---|
| Środowisko niewątkowe | LIBMQM, | Serwer dla języka C |
| | LIBMQIC i LIBMQM | Klient dla języka C |
| Środowisko wielowątkowe | LIBMQM_R, | Serwer dla języka C |
| | LIBMQIC_R & LIBMQM_R | Klient dla języka C |

## IBM i *Przygotowywanie programów w języku COBOL w systemie IBM i*

Sekcja zawiera informacje o przygotowywaniu programów w języku COBOL w produkcie IBM i oraz o metodzie uzyskiwania dostępu do interfejsu MQI z poziomu programu w języku COBOL.

### O tym zadaniu

Aby uzyskać dostęp do interfejsu MQI z poziomu programów w języku COBOL, produkt IBM MQ for IBM i udostępnia skonsolidowany interfejs proceduralny wywoływania udostępniany przez programy usługowe. Zapewnia to dostęp do wszystkich funkcji MQI w produkcie IBM MQ for IBM i oraz obsługę aplikacji wielowątkowych. Ten interfejs może być używany tylko z kompilatorem języka ILE COBOL.

Do uzyskania dostępu do funkcji MQI używana jest standardowa składnia COBOL CALL.

Pliki kopii języka COBOL zawierające stałe nazwane i definicje struktur używane z MQI są zawarte w źródłowym zbiorze fizycznym QMQM/QCBLLESRC.

Pliki kopii w języku COBOL używają znaku pojedynczego cudzysłowu (') jako separatora łańcuchów. Kompilatory języka COBOL IBM i zakładają, że separatorem jest znak cudzysłowu ("). Aby zapobiec generowaniu komunikatów ostrzegawczych przez kompilatory, należy podać parametr `OPTION (*APOST)` w komendach **CRTCBLPGM**, **CRTBNDCBL**lub **CRTCBLMOD**.

Aby kompilator akceptował pojedynczy znak cudzysłowu (') jako ogranicznik łańcucha w plikach kopii COBOL, należy użyć opcji kompilatora \APOST.

**Uwaga:** Interfejs wywołań dynamicznych nie jest udostępniany w wersji IBM MQ 9.0 lub nowszej.

Aby użyć interfejsu wywołania procedury powiązanej, wykonaj następujące kroki.

## Procedura

1. Utwórz moduł przy użyciu kompilatora **CRTCBLMOD** , określając parametr:

```
LINKLIT(*PRC)
```

2. Użyj komendy **CRTPGM** , aby utworzyć obiekt programu, podając odpowiedni parametr:

   W przypadku aplikacji niewielowątkowych:

```
BNDSRVPGM(QMQM/AMQ0STUB)        Server for COBOL for non-threaded applications
BNDSRVPGM(QMQM/AMQCSTUB)        Client for COBOL for non-threaded applications
```

   W przypadku aplikacji wielowątkowych:

```
BNDSRVPGM(QMQM/AMQ0STUB_R)        Server for COBOL for threaded applications
BNDSRVPGM(QMQM/AMQCSTUB_R)        Client for COBOL for threaded applications
```

   **Uwaga:** Z wyjątkiem programów utworzonych za pomocą kompilatora ILE COBOL V4R4 i zawierających opcję THREAD (SERIALIZE) w instrukcji PROCESS, programy w języku COBOL nie mogą używać bibliotek IBM MQ z wątkami. Nawet jeśli program w języku COBOL został w ten sposób zabezpieczony wątkowo, należy zachować ostrożność podczas projektowania aplikacji, ponieważ THREAD (SERIALIZE) wymusza serializację procedur w języku COBOL na poziomie modułu i może mieć wpływ na ogólną wydajność.

   Więcej informacji na ten temat zawiera podręcznik *WebSphere Development Studio: ILE COBOL Programmer's Guide* oraz podręcznik *WebSphere Development Studio: ILE COBOL Reference* .

   Więcej informacji na temat kompilowania aplikacji CICS zawiera podręcznik *CICS for IBM i Application Programming Guide*, SC41-5454.

### IBM i *Przygotowywanie programów CICS w systemie IBM i*

Sekcja zawiera informacje o krokach wymaganych podczas przygotowywania programów CICS w systemie IBM i.

Aby utworzyć program zawierający instrukcje EXEC CICS i wywołania MQI, wykonaj następujące kroki:

1. W razie potrzeby przygotuj mapy za pomocą komendy CRTCICSMAP.
2. Przetłumacz komendy EXEC CICS na instrukcje w języku rodzimym. Dla programu w języku C należy użyć komendy CRTCICSC. Użyj komendy CRTCICSCBL dla programu w języku COBOL.

   Uwzględnij `CICSOPT(*NOGEN)` w komendzie CBL CRTCICSC lub CRTCICS. Powoduje to zatrzymanie przetwarzania w celu uwzględnienia odpowiednich programów usługowych CICS i IBM MQ . Komenda ta domyślnie umieszcza kod w katalogu QTEMP/QACYCICS.
3. Skompiluj kod źródłowy przy użyciu komendy CRTCMOD (dla programu w języku C) lub komendy CRTCBLMOD (dla programu w języku COBOL).
4. Użyj komendy CRTPGM, aby połączyć skompilowany kod z odpowiednimi programami usługowymi CICS i IBM MQ . Spowoduje to utworzenie programu wykonywalnego.

Poniżej przedstawiono przykład takiego kodu (kompiluje on dostarczony program przykładowy CICS ):

```
CRTCICSC  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 *Przygotowywanie programów RPG w systemie IBM i*

Jeśli używany jest język IBM MQ for IBM i, można pisać aplikacje w języku RPG.

Więcej informacji na ten temat zawiera sekcja “Kodowanie programów IBM MQ w języku RPG (tylko w systemieIBM i )” na stronie 1088oraz podręcznik IBM i Application Programming Reference (ILE/RPG).

### IBM i *Uwagi dotyczące programowania w języku SQL dla systemu IBM i*

Sekcja zawiera informacje na temat kroków wymaganych podczas budowania aplikacji w systemie IBM i za pomocą języka SQL.

Jeśli program zawiera instrukcje SQL EXEC i wywołania MQI, wykonaj następujące kroki:

1. Przetłumacz komendy SQL EXEC na instrukcje w języku rodzimym. Użyj komendy CRTSQLCI dla programu w języku C. Użyj komendy CRTSQLCBLI dla programu w języku COBOL.

   Uwzględnij `OPTION(*NOGEN)` w komendzie CRTSQLCI lub CRTSQLCBLI. Powoduje to zatrzymanie przetwarzania w celu uwzględnienia odpowiednich programów usługowych IBM MQ . Komenda ta domyślnie umieszcza kod w QTEMP/QSQLTEMP.
2. Skompiluj kod źródłowy przy użyciu komendy CRTCMOD (dla programu w języku C) lub komendy CRTCBLMOD (dla programu w języku COBOL).
3. Użyj komendy CRTPGM, aby połączyć skompilowany kod z odpowiednimi programami usługowymi systemu IBM MQ . Spowoduje to utworzenie programu wykonywalnego.

Przykład takiego kodu jest następujący (kompiluje program, SQLTEST, w bibliotece, SQLUSER):

```
CRTSQLCI  OBJ(MQTEST/SQLTEST) SRCFILE(SQLUSER/QCSRC) +
     SRCMBR(SQLTEST) OUTPUT(*PRINT) OPTION(*NOGEN)
CRTCMOD   MODULE(MQTEST/SQLTEST) +
     SRCFILE(QTEMP/QSQLTEMP) OUTPUT(*PRINT)
CRTPGM   PGM(MQTEST/SQLTEST) +
     BNDSRVPGM(QMQM/LIBMQIC)
```

## Linux Budowanie aplikacji proceduralnej w systemie Linux

Te informacje opisują dodatkowe zadania oraz zmiany w zadaniach standardowych, które należy wykonać podczas budowania aplikacji IBM MQ for Linux w celu ich uruchomienia.

Obsługiwane są środowiska C i C + +. Informacje na temat przygotowywania programów w języku C++ zawiera sekcja Korzystanie z języka C++.

### Linux *Przygotowywanie programów w języku C w systemie Linux*

Prekompilowane programy w języku C są dostarczane w katalogu *MQ_INSTALLATION_PATH*`/samp/bin` . Aby zbudować przykład z kodu źródłowego, należy użyć kompilatora `gcc` .

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Praca w normalnym środowisku. Więcej informacji na temat programowania aplikacji 64-bitowych zawiera sekcja Standardy kodowania na platformach 64-bitowych.

## Dowiązywanie bibliotek

Poniższa tabela zawiera listę bibliotek, które są potrzebne podczas przygotowywania programów w języku C w systemie Linux.

- Należy połączyć programy z odpowiednią biblioteką udostępnioną przez IBM MQ.

  W środowisku niewielowątkowym należy utworzyć odsyłacz tylko do jednej z następujących bibliotek:

| Plik biblioteki | Typ programu/wyjścia |
|---|---|
| libmqm.so | Serwer dla języka C |
| libmqic.so & libmqm.so | Klient dla języka C |

W środowisku wielowątkowym należy utworzyć odsyłacz tylko do jednej z następujących bibliotek:

| Plik biblioteki | Typ programu/wyjścia |
|---|---|
| libmqm_r.so | Serwer dla języka C |
| libmqic_r.so & libmqm_r.so | Klient dla języka C |

**Uwaga:**

1. Nie można utworzyć odsyłacza do więcej niż jednej biblioteki. Oznacza to, że nie można jednocześnie utworzyć dowiązania do biblioteki wielowątkowej i niewielowątkowej.
2. W przypadku pisania instalowalnej usługi (więcej informacji na ten temat zawiera publikacja Administrowanie programem IBM MQ ), należy utworzyć odsyłacz do biblioteki `libmqmzf.so` .
3. Jeśli aplikacja jest generowana na potrzeby koordynacji zewnętrznej przez menedżer transakcji zgodny z interfejsem XA, taki jak IBM TXSeries Encina lub BEA Tuxedo, należy połączyć się z serwerem `libmqmxa.so` (lub `libmqmxa64.so` , jeśli menedżer transakcji traktuje typ "long" jako 64-bitowy) i bibliotekami `libmqz.so` w aplikacji niewielowątkowej oraz z serwerem `libmqmxa_r.so` (lub `libmqmxa64_r.so` ). i bibliotek `libmqz_r.so` w aplikacji wielowątkowej.
4. Należy dowiązać biblioteki IBM MQ przed wszystkimi innymi bibliotekami produktu.

Linux *Budowanie aplikacji 31-bitowych*

Ten temat zawiera przykłady komend używanych do budowania 31-bitowych programów w różnych środowiskach.

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

**Aplikacja kliencka C, 31-bitowa, niewielowątkowa**

```
gcc -m31 -o famqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

**Aplikacja kliencka C, 31-bitowa, wielowątkowa**

```
gcc -m31 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

**Aplikacja serwera C, 31-bitowa, niewielowątkowa**

```
gcc -m31 -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
```

**Aplikacja serwera C, 31-bitowa, wielowątkowa**

```
gcc -m31 -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
```

**Aplikacja kliencka C + +, 31-bitowa, niewielowątkowa**

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

**Aplikacja kliencka C + +, 31-bitowa, wielowątkowa**

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.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
```

**Aplikacja serwera C + +, 31-bitowa, niewielowątkowa**

```
g++ -m31 -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
```

**Aplikacja serwera C + +, 31-bitowa, wielowątkowa**

```
g++ -m31 -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
```

**Wyjście klienta C, 31-bitowe, niewielowątkowe**

```
gcc -m31 -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
```

**Wyjście klienta C, 31-bitowe, wielowątkowe**

```
gcc -m31 -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
```

**Wyjście serwera C, 31-bitowe, niewątkowe**

```
gcc -m31 -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
```

**Wyjście serwera C, 31-bitowe, wielowątkowe**

```
gcc -m31 -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 *Budowanie aplikacji 32-bitowych*

Ten temat zawiera przykłady komend używanych do budowania programów 32-bitowych w różnych środowiskach.

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

**Aplikacja kliencka C, 32-bitowa, niewielowątkowa**

```
gcc -m32 -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

**Aplikacja kliencka C, 32-bitowa, wielowątkowa**

```
gcc -m32 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

**Aplikacja serwera C, 32-bitowa, niewielowątkowa**

```
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
```

**Aplikacja serwera C, 32-bitowa, wielowątkowa**

```
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
```

**Aplikacja kliencka C + +, 32-bitowa, niewielowątkowa**

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

**Aplikacja kliencka C + +, 32-bitowa, wielowątkowa**

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.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
```

**Aplikacja serwera C + +, 32-bitowa, niewielowątkowa**

```
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
```

**Aplikacja serwera C + +, 32-bitowa, wielowątkowa**

```
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
```

**Wyjście klienta C, 32-bitowe, niewielowątkowe**

```
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
```

**Wyjście klienta C, 32-bitowe, wielowątkowe**

```
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
```

**Wyjście serwera C, 32-bitowe, niewątkowe**

```
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
```

**Wyjście serwera C, 32-bitowe, wielowątkowe**

```
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 *Budowanie aplikacji 64-bitowych*

Ten temat zawiera przykłady komend używanych do budowania programów 64-bitowych w różnych środowiskach.

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

**Aplikacja kliencka C, 64-bitowa, niewielowątkowa**

```
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
```

**Aplikacja kliencka C, 64-bitowa, wielowątkowa**

```
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
```

**Aplikacja serwera C, 64-bitowa, niewielowątkowa**

```
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
```

**Aplikacja serwera C, 64-bitowa, wielowątkowa**

```
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
```

**Aplikacja kliencka C + +, 64-bitowa, niewielowątkowa**

```
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
```

**Aplikacja kliencka C + +, 64-bitowa, wielowątkowa**

```
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
```

**Aplikacja serwera C + +, 64-bitowa, niewielowątkowa**

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

**Aplikacja serwera C + +, 64-bitowa, wielowątkowa**

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.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
```

**Wyjście klienta C, 64-bitowe, niewielowątkowe**

```
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
```

**Wyjście klienta C, 64-bitowe, wielowątkowe**

```
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
```

**Wyjście serwera C, 64-bitowe, niewielowątkowe**

```
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
```

**Wyjście serwera C, 64-bitowe, wielowątkowe**

```
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 *Przygotowywanie programów w języku COBOL w systemie Linux*

Sekcja zawiera informacje na temat przygotowywania programów w języku COBOL w systemie Linux oraz przygotowywania programów w języku COBOL za pomocą oprogramowania IBM COBOL for Linux na platformach x86 i Micro Focus COBOL.

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

1. 32-bitowe książki kopii COBOL są instalowane w następującym katalogu:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   i dowiązania symboliczne są tworzone w:

   ```
   MQ_INSTALLATION_PATH/inc
   ```

2. Na platformach 64-bitowych 64-bitowe książki w języku COBOL są instalowane w następującym katalogu:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

3. W poniższych przykładach należy ustawić parametr COBCPY na wartość:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   dla aplikacji 32-bitowych oraz:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

dla aplikacji 64-bitowych.

Należy połączyć program z jednym z następujących elementów:

| Plik biblioteki | Typ programu/wyjścia |
|---|---|
| libmqmcb.so | Serwer dla języka COBOL |
| libmqicb.so | Klient dla języka COBOL |
| libmqmcb_r.so | Server for COBOL (aplikacja wielowątkowa) |
| libmqicb_r.so | Client for COBOL (aplikacja wielowątkowa) |

## Przygotowywanie programów w języku COBOL przy użyciu systemu IBM COBOL for Linux na platformie x86

Przykładowe programy w języku COBOL są dostarczane z produktem IBM MQ. Aby skompilować taki program, należy wprowadzić odpowiednią komendę z następującej listy:

**32-bitowa, niewątkowa aplikacja serwera**

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqmcb -ICOBCPY_VALUE
```

**32-bitowa niewątkowa aplikacja kliencka**

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqicb -ICOBCPY_VALUE
```

**32-bitowa aplikacja serwera z wątkami**

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqmcb_r -ICOBCPY_VALUE
```

**32-bitowa wątkowa aplikacja kliencka**

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqicb_r -ICOBCPY_VALUE
```

## Przygotowywanie programów w języku COBOL przy użyciu programu Micro Focus COBOL

Przed skompilowaniem programu ustaw zmienne środowiskowe w następujący sposób:

```
export COBCPY=COBCPY_VALUE
export LIB= MQ_INSTALLATION_PATH lib:$LIB
```

Aby skompilować 32-bitowy program w języku COBOL, jeśli jest on obsługiwany, przy użyciu programu Micro Focus COBOL, należy wprowadzić komendę:

```
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb Server for COBOL
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb Client for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r Threaded Server for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r Threaded Client for COBOL
```

Aby skompilować 64-bitowy program w języku COBOL przy użyciu programu Micro Focus COBOL, należy wpisać:

```
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb Server for COBOL
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb Client for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r Threaded Server for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r Threaded Client for COBOL
```

gdzie `amqsput` jest programem przykładowym

Opis potrzebnych zmiennych środowiskowych można znaleźć w dokumentacji języka Micro Focus COBOL.

## Windows Budowanie aplikacji proceduralnej w systemie Windows

Publikacje dotyczące systemów Windows opisują sposób budowania aplikacji wykonywalnych na podstawie programów napisanych przez użytkownika.

W tym temacie opisano dodatkowe zadania oraz zmiany w zadaniach standardowych, które należy wykonać podczas budowania aplikacji IBM MQ for Windows w celu ich uruchomienia w systemach Windows . Obsługiwane są języki programowania C, C + +, COBOL i Visual Basic. Informacje na temat przygotowywania programów w języku C++ zawiera sekcja Korzystanie z języka C++.

Zadania, które należy wykonać, aby utworzyć aplikację wykonywalną za pomocą programu IBM MQ for Windows , różnią się w zależności od języka programowania, w którym napisany jest kod źródłowy. Oprócz kodowania wywołań MQI w kodzie źródłowym należy dodać odpowiednie instrukcje języka, aby uwzględnić pliki włączane IBM MQ for Windows dla używanego języka. Zapoznaj się z zawartością tych plików. Pełny opis znajduje się w sekcji “Pliki definicji danych IBM MQ” na stronie 741 .

### Windows *Budowanie aplikacji 64-bitowych w systemie Windows*

W systemie IBM MQ for Windowsobsługiwane są zarówno aplikacje 32-bitowe, jak i 64-bitowe. Pliki wykonywalne i biblioteki IBM MQ są dostarczane zarówno w formacie 32-, jak i 64-bitowym. W zależności od używanej aplikacji należy użyć odpowiedniej wersji.

#### Pliki wykonywalne i biblioteki

Zarówno 32-bitowe, jak i 64-bitowe wersje bibliotek IBM MQ są dostarczane w następujących miejscach:

*Tabela 148. Położenie bibliotek IBM MQ*

| **Wersja biblioteki** | **Katalog zawierający pliki bibliotek** |
|---|---|
| 32 bity | *MQ_INSTALLATION_PATH* \Tools\Lib |
| 64-bitowe | *MQ_INSTALLATION_PATH* \Tools\Lib64 |

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Aplikacje 32-bitowe nadal działają normalnie po migracji. Pliki 32-bitowe znajdują się w tym samym katalogu, co w poprzednich wersjach produktu.

Aby utworzyć wersję 64-bitową, należy upewnić się, że środowisko jest skonfigurowane do używania plików bibliotek w produkcie *MQ_INSTALLATION_PATH* \Tools\Lib64. Upewnij się, że zmienna środowiskowa LIB nie jest ustawiona do wyszukiwania w folderze zawierającym biblioteki 32-bitowe.

### Windows *Przygotowywanie programów w języku C w systemie Windows*

Praca w typowym środowisku Windows ; IBM MQ for Windows nie wymaga żadnych specjalnych działań.

Więcej informacji na temat programowania aplikacji 64-bitowych zawiera sekcja Standardy kodowania na platformach 64-bitowych.

- Powiąż programy z odpowiednimi bibliotekami udostępnionymi przez IBM MQ:

| **Plik biblioteki** | **Typ programu/wyjścia** |
|---|---|
| *MQ_INSTALLATION_PATH* \Tools\Lib\mqm.lib | serwer dla 32-bitowego C |
| *MQ_INSTALLATION_PATH* \Tools\Lib\mqic.lib | klient dla 32-bitowego języka C |

| Plik biblioteki | Typ programu/wyjścia |
| --- | --- |
| *MQ_INSTALLATION_PATH*<br>\Tools\Lib\mqicxa.lib | Klient dla 32-bitowego języka C z koordynacją transakcji |
| *MQ_INSTALLATION_PATH*<br>\Tools\Lib64\mqm.lib | Serwer dla 64-bitowego języka C |
| *MQ_INSTALLATION_PATH*<br>\Tools\Lib64\mqic.lib | Klient dla 64-bitowego języka C |
| *MQ_INSTALLATION_PATH*<br>\Tools\Lib64\mqicxa.lib | klient dla 64-bitowego języka C z koordynacją transakcji |

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Poniższa komenda przedstawia przykład kompilowania przykładowego programu amqsget0 (przy użyciu kompilatora Microsoft Visual C++).

Dla aplikacji 32-bitowych:

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib
```

Dla aplikacji 64-bitowych:

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib
```

**Uwaga:**

- W przypadku pisania instalowalnej usługi (więcej informacji na ten temat zawiera plik Administrowanie programem IBM MQ ) należy utworzyć odsyłacz do biblioteki mqmzf.lib .
- Jeśli aplikacja jest generowana na potrzeby koordynacji zewnętrznej przez menedżer transakcji zgodny z interfejsem XA, taki jak IBM TXSeries Encina lub BEA Tuxedo, należy utworzyć odsyłacz do biblioteki mqmxa.lib lub mqmxa.lib .
- Jeśli piszesz wyjście CICS , utwórz odsyłacz do biblioteki mqmcics4.lib .
- Należy dowiązać biblioteki IBM MQ przed wszystkimi innymi bibliotekami produktu.

- Biblioteki DLL muszą znajdować się w podanej ścieżce (PATH).
- Jeśli w miarę możliwości używane są małe litery, można przejść z systemu IBM MQ for Windows do systemu IBM MQ for AIX or Linux , w którym wymagane jest użycie małych liter.

## Przygotowywanie programów CICS i Transaction Server

Przykładowe źródło w języku C dla transakcji produktu CICS IBM MQ jest udostępniane przez produkt AMQSCIC0.CCS. Jest on budowany przy użyciu standardowych narzędzi systemu CICS . Na przykład w przypadku bazy danych TXSeries for Windows 2000:

1. Ustaw zmienną środowiskową (wprowadź następujący kod w jednym wierszu):

```
set CICS_IBMC_FLAGS=-I MQ_INSTALLATION_PATH\Tools\C\Include;
%CICS_IBMC_FLAGS%
```

2. Ustaw zmienną środowiskową USERLIB:

```
set USERLIB=MQM.LIB;%USERLIB%
```

3. Przetłumacz, skompiluj i dowiąż przykładowy program:

```
cicstcl -l IBMC amqscic0.ccs
```

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Zostało to opisane w publikacji *Transaction Server for Windows NT Application Programming Guide ( CICS ) V4*.

Więcej informacji na temat obsługi transakcji CICS można znaleźć w publikacji Administrowanie programem IBM MQ.

### Windows *Przygotowywanie programów w języku COBOL w systemie Windows*

Te informacje umożliwiają przygotowanie programów w języku COBOL w systemie Windowsoraz przygotowanie programów CICS i Transaction Server.

1. 32-bitowe książki kopii w języku COBOL są instalowane w następującym katalogu: *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook.
2. 64-bitowe książki kopii w języku COBOL są instalowane w następującym katalogu: *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook64
3. W poniższych przykładach należy ustawić parametr CopyBook na wartość:

   ```
   CopyBook
   ```

   dla aplikacji 32-bitowych oraz:

   ```
   CopyBook64
   ```

   dla aplikacji 64-bitowych.

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Aby przygotować programy w języku COBOL w systemach Windows , należy dowiązać program do jednej z następujących bibliotek udostępnianych przez IBM MQ:

| Plik biblioteki | Program lub typ wyjścia |
|---|---|
| *MQ_INSTALLATION_PATH* \Tools\Lib\mqmcb | 32-bitowy serwer dla Micro Focus COBOL |
| *MQ_INSTALLATION_PATH* \Tools\Lib\mqiccb | 32-bitowy klient dla Micro Focus COBOL |
| *MQ_INSTALLATION_PATH* \Tools\Lib64\mqmcb | 64-bitowy serwer dla Micro Focus COBOL |
| *MQ_INSTALLATION_PATH* \Tools\Lib64\mqiccb | 64-bitowy klient dla Micro Focus COBOL |

Podczas uruchamiania programu w środowisku klienta MQI należy upewnić się, że biblioteka DOSCALLS jest wyświetlana przed jakąkolwiek biblioteką COBOL lub IBM MQ .

## Przygotowywanie programów w języku COBOL przy użyciu programu Micro Focus COBOL

Ponownie pokonsoliduj wszystkie istniejące 32-bitowe programy IBM MQ Micro Focus COBOL za pomocą biblioteki mqmcb.lib lub mqiccb.lib, a nie biblioteki mqmcbb i mqiccbb .

Aby skompilować przykładowy program amq0put0przy użyciu języka Micro Focus COBOL:

1. Ustaw zmienną środowiskową COBCPY tak, aby wskazywała na struktury copybook języka COBOL IBM MQ (wprowadź następujący kod w jednym wierszu):

```
set COBCPY= MQ_INSTALLATION_PATH\
Tools\Cobol\Copybook
```

2. Skompiluj program, aby uzyskać plik wynikowy:

```
cobol amq0put0 LITLINK
```

3. Dowiąż plik obiektu do systemu środowiska wykonawczego.
   - Ustaw zmienną środowiskową LIB tak, aby wskazywała biblioteki COBOL kompilatora.
   - Dowiąż plik wynikowy do użycia na serwerze IBM MQ :

     ```
     cbllink amq0put0.obj mqmcb.lib
     ```

   - Lub dowiąż plik wynikowy do użycia na kliencie IBM MQ :

     ```
     cbllink amq0put0.obj mqiccb.lib
     ```

## Przygotowywanie programów CICS i Transaction Server

Aby skompilować i skonsolidować program TXSeries for Windows NT, V5.1 przy użyciu języka COBOL IBM VisualAge :

1. Ustaw zmienną środowiskową (wprowadź następujący kod w jednym wierszu):

```
set CICS_IBMCOB_FLAGS= MQ_INSTALLATION_PATH\
Cobol\Copybook\VAcobol;%CICS_IBMCOB_FLAGS%
```

2. Ustaw zmienną środowiskową USERLIB:

```
set USERLIB=MQMCBB.LIB
```

3. Przetłumacz, skompiluj i dowiąż program:

```
cicstcl -l IBMCOB myprog.ccp
```

Zostało to opisane w publikacji *Transaction Server for Windows NT, V4 Application Programming* Guide.

Aby skompilować i skonsolidować program CICS for Windows V5 przy użyciu języka Micro Focus COBOL:

- Ustaw zmienną INCLUDE:

```
set
INCLUDE=drive:\programname\ibm\websphere\tools\c\include;
        drive:\opt\cics\include;%INCLUDE%
```

- Ustaw zmienną środowiskową COBCPY:

```
setCOBCPY=drive:\programname\ibm\websphere\tools\cobol\copybook;
          drive:\opt\cics\include
```

- Ustaw opcje języka COBOL:
  - set

– COBOPTS=/LITLINK /NOTRUNC

i uruchom następujący kod:

```
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 *Przygotowywanie programów Visual Basic w systemie Windows*

Informacje, które należy wziąć pod uwagę podczas używania programów Microsoft Visual Basic w systemie Windows.

Deprecated Począwszy od wersji IBM MQ 9.0, obsługa języka Microsoft Visual Basic 6.0 jest nieaktualna. Zalecaną technologią zastępczą są klasy IBM MQ dla .NET. Aby uzyskać więcej informacji, zapoznaj się z sekcją: Tworzenie aplikacji .NET.

**Uwaga:** Nie są dostarczane 64-bitowe wersje plików modułu Visual Basic .

Aby przygotować programy Visual Basic w systemie Windows:

1. Utwórz nowy projekt.
2. Dodaj dostarczony plik modułu CMQB.BASdla projektu.
3. Dodaj inne dostarczone pliki modułów, jeśli są potrzebne:
   - CMQBB.BAS: obsługa MQAI
   - CMQCFB.BAS:
   - CMQXB.BAS: Obsługa wyjść kanałów
   - CMQPSB.BAS: publikowanie/subskrypcja

Informacje na temat używania wywołania MQCONNXAny z poziomu produktu Visual Basiczawiera sekcja “Kodowanie w języku Visual Basic” na stronie 1084 .

Przed wykonaniem jakichkolwiek wywołań MQI w kodzie projektu należy wywołać procedurę MQ_SETDEFAULTS. Ta procedura służy do konfigurowania struktur domyślnych wymaganych przez wywołania MQI.

Określ, czy tworzony jest serwer lub klient IBM MQ przed skompilowaniem lub uruchomieniem projektu, ustawiając warunkową zmienną kompilacji *MqType*. Ustaw wartość *MqType* w projekcie Visual Basic na 1 dla serwera lub 2 dla klienta w następujący sposób:

1. Wybierz menu Projekt.
2. Wybierz opcję *Name* Właściwości (gdzie *Name* jest nazwą bieżącego projektu).
3. W oknie dialogowym wybierz kartę Make (Utwórz).
4. W polu Argumenty kompilacji warunkowej wprowadź następującą wartość dla serwera:

   ```
   MqType=1
   ```

   lub w przypadku klienta:

   ```
   MqType=2
   ```

**Pojęcia pokrewne**

“Kodowanie w języku Visual Basic” na stronie 1084
Informacje, które należy wziąć pod uwagę podczas kodowania programów IBM MQ w języku Microsoft Visual Basic. Parametr Visual Basic jest obsługiwany tylko w systemie Windows.

**Odsyłacze pokrewne**

“Łączenie aplikacji Visual Basic z kodem IBM MQ MQI client” na stronie 949

Aplikacje Microsoft Visual Basic można połączyć za pomocą kodu IBM MQ MQI client w systemie Windows.

### Windows *Wyjście zabezpieczeń SPIP*

Program IBM MQ for Windows udostępnia wyjście zabezpieczeń zarówno dla serwera IBM MQ MQI client , jak i serwera IBM MQ . Jest to program obsługi wyjścia kanału, który umożliwia uwierzytelnianie kanałów systemu IBM MQ za pomocą interfejsu SSPI (Security Services Programming Interface). Interfejs SSPI udostępnia zintegrowane narzędzia bezpieczeństwa systemów Windows .

Pakiety zabezpieczeń są ładowane z biblioteki security.dll lub secur32.dll. Te biblioteki DLL są dostarczane z systemem operacyjnym.

Uwierzytelnianie jednokierunkowe jest udostępniane przy użyciu usług uwierzytelniania NTLM. Uwierzytelnianie dwukierunkowe jest udostępniane przy użyciu usług uwierzytelniania Kerberos .

Program obsługi wyjścia zabezpieczeń jest dostarczany w formacie źródłowym i obiektowym. Można użyć kodu obiektu w takim stanie, w jakim jest, lub użyć kodu źródłowego jako punktu początkowego do utworzenia własnych programów użytkownika.

Patrz także “Korzystanie z wyjścia zabezpieczeń SSPI w systemie Windows” na stronie 1167.

## Wprowadzenie do wyjść bezpieczeństwa

Wyjście zabezpieczeń tworzy bezpieczne połączenie między dwoma programami z wyjściami zabezpieczeń, z których jeden jest przeznaczony dla wysyłającego agenta kanału komunikatów (Message channel agent - MCA), a drugi dla odbierającego agenta MCA.

Program, który inicjuje bezpieczne połączenie, czyli pierwszy program, który ma przejąć kontrolę po ustanowieniu sesji MCA, jest nazywany *inicjatorem kontekstu*. Program partnerski jest znany jako *akceptor kontekstu*.

W poniższej tabeli przedstawiono niektóre typy kanałów, które są inicjatorami kontekstu i powiązanymi z nimi akceptorami kontekstu.

*Tabela 149. Inicjatory kontekstu i powiązane z nimi akceptory kontekstu*

| **Inicjator kontekstu** | **Akceptor kontekstu** |
|---|---|
| MQCHT_CLNTCONN | MQCHT_SVRCONN |
| MQCHT_RECEIVER | MQCHT_SENDER |
| MQCHT_CLUSRCVR | MQCHT_CLUSSDR |

Program obsługi wyjścia zabezpieczeń ma dwa punkty wejścia:

- **SCY_NTLM,**

  Używane są usługi uwierzytelniania NTLM, które zapewniają uwierzytelnianie jednokierunkowe. Protokół NTLM umożliwia serwerom weryfikowanie tożsamości ich klientów. Nie pozwala to klientom na weryfikowanie tożsamości serwera ani jednemu serwerowi na weryfikowanie tożsamości innego serwera. Uwierzytelnianie NTLM zostało zaprojektowane dla środowiska sieciowego, w którym zakłada się, że serwery są oryginalne.

- **SCY_KERBEROS**

  Używane są usługi wzajemnego uwierzytelniania Kerberos . Protokół Kerberos nie zakłada, że serwery w środowisku sieciowym są autentyczne. Podmioty na obu końcach połączenia sieciowego mogą weryfikować tożsamość drugiej strony. Oznacza to, że serwery mogą weryfikować tożsamość klientów i innych serwerów, a klienty mogą weryfikować tożsamość serwera.

## Co robi wyjście zabezpieczeń

W tej sekcji opisano działania programów obsługi wyjścia kanału SSPI.

Dostarczone programy obsługi wyjścia kanału udostępniają uwierzytelnianie jednokierunkowe lub dwukierunkowe (wzajemne) systemu partnerskiego podczas uruchamiania sesji. Dla konkretnego kanału każdy program obsługi wyjścia ma powiązaną *nazwę użytkownika* (podobną do identyfikatora użytkownika, patrz sekcja "IBM MQ kontrola dostępu i Windows nazwy użytkowników" na stronie 1052 ). Połączenie między dwoma programami obsługi wyjścia jest powiązaniem między dwiema nazwami użytkowników.

Po ustanowieniu sesji bazowej nawiązywane jest bezpieczne połączenie między dwoma bezpiecznymi programami obsługi wyjścia (jednym dla wysyłającego agenta MCA i jednym dla odbierającego agenta MCA). Sekwencja operacji jest następująca:

1. Każdy program jest powiązany z określoną nazwą użytkownika, na przykład w wyniku jawnej operacji logowania.
2. Inicjator kontekstu żąda bezpiecznego połączenia z partnerem z pakietu zabezpieczeń (dla nazwanego partnera Kerberos) i odbiera token (o nazwie token1). Token jest wysyłany do programu partnerskiego przy użyciu sesji bazowej, która została już ustanowiona.
3. Program partnerski (akceptor kontekstu) przekazuje token1 do pakietu zabezpieczeń, który sprawdza, czy inicjator kontekstu jest autentyczny. W przypadku protokołu NTLM połączenie jest teraz nawiązywane.
4. Dla wyjścia zabezpieczeń dostarczonego przez Kerberos(na potrzeby uwierzytelniania wzajemnego) pakiet zabezpieczeń generuje również drugi znacznik (nazywany token2), który jest zwracany przez akceptor kontekstu do inicjatora kontekstu przy użyciu sesji bazowej.
5. Inicjator kontekstu używa znacznika token2 do sprawdzenia, czy akceptor kontekstu jest autentyczny.
6. Na tym etapie, jeśli obie aplikacje są zadowolone z autentyczności tokenu partnera, nawiązywane jest bezpieczne (uwierzytelnione) połączenie.

## IBM MQ kontrola dostępu i Windows nazwy użytkowników

Kontrola dostępu zapewniana przez program IBM MQ jest oparta na użytkowniku i grupie. Uwierzytelnianie, które jest udostępniane przez Windows , jest oparte na nazwach użytkowników, takich jak użytkownik i servicePrincipalName (SPN). W przypadku nazwy servicePrincipalwiele z nich może być powiązanych z pojedynczym użytkownikiem.

Wyjście zabezpieczeń SSPI używa odpowiednich nazw użytkowników systemu Windows do uwierzytelniania. Jeśli uwierzytelnianie w systemie Windows powiedzie się, wyjście przekazuje identyfikator użytkownika, który jest powiązany z nazwą użytkownika Windows , do systemu IBM MQ w celu kontroli dostępu.

Nazwy użytkowników Windows , które są istotne dla uwierzytelniania, różnią się w zależności od typu używanego uwierzytelniania.

- W przypadku uwierzytelniania NTLM nazwą użytkownika Windows dla inicjatora kontekstu jest ID użytkownika powiązany z uruchomionym procesem. Ponieważ to uwierzytelnianie jest jednokierunkowe, nazwa użytkownika powiązana z akceptorem kontekstu jest nieistotna.
- W przypadku uwierzytelniania Kerberos w kanałach CLNTCONN nazwą użytkownika Windows jest identyfikator użytkownika powiązany z uruchomionym procesem. W przeciwnym razie nazwa użytkownika Windows jest nazwą servicePrincipalutworzoną przez dodanie następującego przedrostka do nazwy QueueManager.

```
ibmMQSeries/
```

## z/OS Budowanie aplikacji proceduralnej w systemie z/OS

Publikacje CICS, IMSi z/OS opisują sposób budowania aplikacji działających w tych środowiskach.

W tej kolekcji tematów opisano dodatkowe zadania oraz zmiany w zadaniach standardowych, które należy wykonać podczas budowania aplikacji IBM MQ for z/OS dla tych środowisk. Obsługiwane są języki programowania COBOL, C + +, Assembler i PL/I. (Informacje na temat budowania aplikacji C++ zawiera sekcja Korzystanie z języka C++).

Zadania, które należy wykonać w celu utworzenia wykonywalnej aplikacji IBM MQ for z/OS , zależą zarówno od języka programowania, w którym program jest napisany, jak i od środowiska, w którym aplikacja będzie uruchamiana.

Oprócz kodowania wywołań MQI w programie należy dodać odpowiednie instrukcje języka, aby dołączyć plik definicji danych IBM MQ for z/OS dla używanego języka. Zapoznaj się z zawartością tych plików. Pełny opis znajduje się w sekcji “Pliki definicji danych IBM MQ” na stronie 741 .

## Uwaga

Nazwa **thlqual** jest kwalifikatorem wysokiego poziomu biblioteki instalacji w systemie z/OS.

### z/OS *Przygotowywanie programu do uruchomienia*

Po napisaniu programu dla aplikacji IBM MQ w celu utworzenia aplikacji wykonywalnej należy ją skompilować lub złożyć, a następnie skonsolidować wynikowy kod obiektu z programem pośredniczącym dostarczanym przez IBM MQ for z/OS dla każdego obsługiwanego środowiska.

Sposób przygotowania programu zależy zarówno od środowiska (wsadowego, CICS, IMS(BMP lub MPP), Linux lub z/OS UNIX System Services), w którym działa aplikacja, jak i od struktury zestawów danych w instalacji z/OS .

W sekcji “Dynamiczne wywoływanie kodu pośredniczącego IBM MQ” na stronie 1060 opisano alternatywną metodę tworzenia wywołań MQI w programach, dzięki czemu nie ma potrzeby konsolidacji kodu pośredniczącego IBM MQ . Ta metoda nie jest dostępna we wszystkich językach i środowiskach.

Nie należy dowiązywać-edytować wyższego poziomu programu pośredniczącego niż wersja produktu IBM MQ for z/OS , na której jest uruchomiony program. Na przykład program działający w systemie MQSeries for OS/390 V5.2 nie może być połączony z programem pośredniczącym dostarczanym z produktem IBM MQ for z/OS V7.

#### z/OS *Budowanie 64-bitowych aplikacji w języku C*

W systemie z/OS64-bitowe aplikacje w języku C są budowane przy użyciu kompilatora LP64 i opcji konsolidatora. Plik nagłówkowy IBM MQ for z/OS *cmqc.h* rozpoznaje, gdy ta opcja zostanie udostępniona kompilatorowi, i generuje typy danych i struktury IBM MQ odpowiednie dla działania 64-bitowego.

Kod w języku C utworzony za pomocą tej opcji musi zostać zbudowany w taki sposób, aby używane były biblioteki DLL (dynamic-link libraries) odpowiednie dla wymaganej semantyki koordynacji. W tym celu należy powiązać skompilowany kod z odpowiednim pokładem bocznym zdefiniowanym w poniższej tabeli:

*Tabela 150. Nazwa pokładu bocznego wymagana dla każdej semantyki koordynacji*

| **spójnik** | **Nazwa pokładu bocznego** |
|---|---|
| MQI zatwierdzania jednofazowego | CSQBMQ2X |
| Zatwierdzanie dwufazowe z koordynacją RRS przy użyciu komend RRS | CSQBRR2X |
| Zatwierdzanie dwufazowe z koordynacją RRS przy użyciu komend MQI | CSQBRI2X |

**Uwaga:** W przypadku 31-bitowych aplikacji w języku C należy również ustawić opcje kompilatora dla interfejsu wywołującego (Language Environment lub XPLINK), zgodnie z opisem w sekcji “Budowanie aplikacji wsadowych systemu z/OS przy użyciu 31-bitowego środowiska językowego lub kodu XPLINK” na stronie 1055. W przypadku 64-bitowych aplikacji w języku C nie należy określać interfejsu wywołującego, ponieważ jedynym obsługiwanym powiązaniem jest XPLINK(XPLINK).

Użyj procedury EDCQCB JCL dostarczanej z produktem *z/OS XL C/C++*, aby zbudować program zatwierdzania jednofazowego IBM MQ jako zadanie wsadowe w następujący sposób:

```
//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
```

Aby zbudować program koordynowany przez usługi RRS w produkcie z/OS UNIX System Services, należy skompilować i utworzyć dowiązanie w następujący sposób:

```
cc -o mqsamp -W c,LP64,DLL -W l,DYNAM=DLL,LP64 -I"//'thlqual.SCSQC370'" "//'thlqual.SCSQDEFS(CSQBRR2X)'" mqsamp.c
```

z/OS *Budowanie aplikacji wsadowych z/OS*

Ta sekcja zawiera informacje na temat budowania aplikacji wsadowych systemu z/OS oraz kroków, które należy wziąć pod uwagę podczas wykonywania tej czynności.

Aby zbudować aplikację dla systemu IBM MQ for z/OS , która działa w ramach zadania wsadowego systemu z/OS , należy utworzyć kod JCL (Job Control Language), który wykonuje następujące zadania:

1. Skompiluj (lub złóż) program, aby utworzyć kod wynikowy. Kod JCL dla kompilacji musi zawierać instrukcje SYSLIB, które udostępniają kompilatorowi pliki definicji danych produktu. Definicje danych są dostarczane w następujących bibliotekach IBM MQ for z/OS :
   - W przypadku języka COBOL: **thlqual**.SCSQCOBC
   - Dla języka asemblera: **thlqual**.SCSQMACS
   - Dla języka C: **thlqual**.SCSQC370
   - W przypadku języka PL/I: **thlqual**.SCSQPLIC
2. W przypadku aplikacji w języku C należy wstępnie dowiązać kod obiektu utworzony w kroku “1” na stronie 1054.
3. W przypadku aplikacji w języku PL/I należy użyć opcji kompilatora EXTRN (SHORT).
4. Połącz-zmodyfikuj kod obiektu utworzony w kroku “1” na stronie 1054 (lub kroku “2” na stronie 1054 dla aplikacji w języku C), aby utworzyć moduł ładujący. Podczas konsolidacji kodu należy dołączyć jeden z wsadowych programów pośredniczących IBM MQ for z/OS (CSQBSTUB lub jeden z programów pośredniczących RRS: CSQBRRSI lub CSQBRSTB).

   **CSQBSTUB**
   zatwierdzanie jednofazowe udostępniane przez IBM MQ for z/OS

   **CSQBRRSI**
   dwufazowe zatwierdzanie udostępniane przez usługi RRS za pomocą interfejsu MQI

   **CSQBRSTB**
   Zatwierdzanie dwufazowe udostępniane bezpośrednio przez usługi RRS

   **Uwagi:**

   a. Jeśli używana jest baza danych CSQBRSTB, należy również połączyć aplikację z ATRSCSS z systemu SYS1.CSSLIB. Rysunek 113 na stronie 1055 i Rysunek 114 na stronie 1055 przedstawiają fragmenty kodu JCL w celu wykonania tej czynności. Kody pośredniczące są niezależne od języka i są dostarczane w bibliotece **thlqual**.SCSQLOAD.

b. Jeśli aplikacja działa w środowisku językowym, należy upewnić się, że zamiast tego została użyta biblioteka DLL środowiska językowego, zgodnie z opisem w sekcji “Budowanie aplikacji wsadowych systemu z/OS przy użyciu 31-bitowego środowiska językowego lub kodu XPLINK” na stronie 1055.

5. Zapisz moduł ładujący w bibliotece dynamicznej aplikacji.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//SYSIN     DD *
  INCLUDE CSQSTUB(CSQBSTUB)
⋮
/*
```

*Rysunek 113. Fragmenty kodu JCL do konsolidacji-edycja modułu obiektu w środowisku wsadowym przy użyciu zatwierdzania jednofazowego*

```
⋮
//*
//* WEBSPHERE 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)
⋮
/*
```

*Rysunek 114. Fragmenty kodu JCL do konsolidacji-edycja modułu obiektu w środowisku wsadowym przy użyciu zatwierdzania dwufazowego*

Aby uruchomić program wsadowy lub RRS, należy dołączyć biblioteki **thlqual**.SCSQAUTH i **thlqual**.SCSQLOAD do konkatenacji zestawu danych STEPLIB lub JOBLIB.

Aby uruchomić program TSO, należy dołączyć biblioteki **thlqual**.SCSQAUTH i **thlqual**.SCSQLOAD do biblioteki STEPLIB używanej przez sesję TSO.

Aby uruchomić program wsadowy z powłoki z/OS UNIX System Services , dodaj biblioteki **thlqual**.SCSQAUTH i **thlqual**.SCSQLOAD do specyfikacji STEPLIB w pliku $HOME? .profile w następujący sposób:

```
STEPLIB= thlqual.SCSQAUTH: thlqual.SCSQLOAD
export STEPLIB
```

z/OS *Budowanie aplikacji wsadowych systemu z/OS przy użyciu 31-bitowego środowiska językowego lub kodu XPLINK*

Produkt IBM MQ for z/OS udostępnia zestaw bibliotek dołączanych dynamicznie (DLL), które muszą być używane podczas konsolidacji aplikacji.

Istnieją dwa warianty bibliotek, które umożliwiają aplikacji użycie jednego z następujących interfejsów wywołujących:

- Interfejs wywołania 31-bitowego środowiska językowego.
- 31-bitowy interfejs wywołujący XPLINK. z/OS XPLINK jest wysoce wydajną konwencją wywoływania dostępną dla aplikacji w języku C. Patrz sekcja XPLINK | NOXPLINK w dokumentacji systemu z/OS 2.2 .

Aby użyć bibliotek DLL, aplikacja jest powiązana lub powiązana z tzw. *sidedecks*, zamiast z kodami pośredniczącymi udostępnionymi we wcześniejszych wersjach. Elementy sidedecks znajdują się w bibliotece SCSQDEFS (zamiast biblioteki SCSQLOAD).

*Tabela 151. Warianty bibliotek dołączanych dynamicznie*

| Zatwierdzanie | 31-bitowa biblioteka DLL środowiska językowego | 31-bitowa biblioteka DLL XPLINK | Nazwa równoważnego kodu pośredniczącego |
|---|---|---|---|
| Biblioteki MQI zatwierdzania jednofazowego | CSQBMQ1 | CSQBMQ1X | CSQBSTUB |
| Zatwierdzanie dwufazowe z koordynacją RRS przy użyciu komend sterowania transakcjami RRS | CSQBRR1 | CSQBRR1X | CSQBRSTB |
| Zatwierdzanie dwufazowe z koordynacją RRS przy użyciu komend sterowania transakcjami MQI | CSQBRI1 | CSQBRI1X | CSQBRRSI |

**Uwaga:** Wszystkie strony zawierają definicję punktu wejścia konwersji danych (MQXCNVC), która została wcześniej rozstrzygnięta przez dołączenie CSQASTUB.

Najczęstsze problemy:

- Jeśli aplikacja używa asynchronicznego korzystania z komunikatów (wywołania MQCB, MQCTL lub MQSUB), a poprzedni interfejs DLL nie jest używany, w protokole zadania pojawia się następujący komunikat:

  ```
  CSQB001E Programy środowiska językowego działające w trybie wsadowym systemu z/OS lub z/OS UNIX
  System Services muszą używać interfejsu DLL do IBM MQ
  ```

  Rozwiązanie: Odbuduj aplikację przy użyciu boczków zamiast kodów pośredniczących, zgodnie z wcześniejszym opisem.

- Podczas budowania programu wyświetlany jest następujący komunikat

  ```
  IEW2469E Atrybuty odwołania do MQAPI-NAME z sekcji kod_użytkownika nie są zgodne z atrybutami
  symbol docelowy
  ```

  Przyczyna: oznacza to, że program XPLINK został skompilowany przy użyciu programu V701 (lub nowszego) w wersji cmqc.h, ale nie jest on powiązany z bodedecks.

  Rozwiązanie: Należy zmienić plik budowania programu w taki sposób, aby był powiązany z odpowiednim kodem bocznikowym z SCSQDEFS zamiast z kodem pośredniczącym z SCSQLOAD.

Poniższy przykładowy kod JCL demonstruje sposób kompilowania i konsolidacji programu w języku C w celu użycia 31-bitowego interfejsu wywoływania biblioteki DLL środowiska językowego:

```
//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
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
```

```
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1)
 NAME MYPROGAM(R)
//
```

**Uwaga:** Kompilacja korzysta z opcji **DLL** . Funkcja konsolidacji używa opcji **DYNAM=DLL** i odwołuje się do biblioteki **CSQBMQ1** .

Poniższy przykładowy kod JCL demonstruje sposób kompilowania i konsolidacji programu w języku C w celu użycia 31-bitowego interfejsu wywoływania biblioteki DLL XPLINK:

```
//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
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1X)
 NAME MYPROGAM(R)
//
```

**Uwaga:** Kompilacja korzysta z opcji **XPLINK** i **DLL** . Funkcja konsolidacji używa opcji **DYNAM=DLL** i odwołuje się do biblioteki **CSQBMQ1X** .

Upewnij się, że do każdego programu w module została dodana biblioteka DLL opcji kompilacji. Komunikaty takie jak IEW2456E 9207 SYMBOL CSQ1BAK UNRESOLVED wskazują, że należy sprawdzić, czy wszystkie programy zostały skompilowane przy użyciu opcji DLL.

z/OS *Budowanie aplikacji CICS w produkcie z/OS*

Te informacje są przydatne podczas budowania aplikacji CICS w produkcie z/OS.

Aby zbudować aplikację dla produktu IBM MQ for z/OS , która działa w systemie CICS, należy wykonać następujące czynności:

- Przetłumacz komendy CICS w programie na język, w którym napisano resztę programu.
- Skompiluj lub złóż dane wyjściowe z translatora, aby utworzyć kod obiektu.
  - W przypadku programów w języku PL/I należy użyć opcji kompilatora EXTRN (SHORT).
  - W przypadku aplikacji w języku C, jeśli aplikacja nie używa kodu XPLINK, należy użyć opcji kompilatora DEFINE (MQ_OS_LINKAGE=1).
- Połącz-edytuj kod obiektu, aby utworzyć moduł ładowania.

CICS udostępnia procedurę wykonywania tych kroków w kolejności dla każdego obsługiwanego języka programowania.

- W przypadku produktu CICS Transaction Server for z/OSpodręcznik *CICS Transaction Server for z/OS System Definition Guide* opisuje sposób korzystania z tych procedur, a publikacja *CICS/ESA Application Programming Guide* zawiera więcej informacji na temat procesu tłumaczenia.

Należy dołączyć:

- W instrukcji SYSLIB etapu kompilacji (lub asemblacji): instrukcje, które udostępniają kompilatorowi pliki definicji danych produktu. Definicje danych są dostarczane w następujących bibliotekach IBM MQ for z/OS :
  - W przypadku języka COBOL: **thlqual**.SCSQCOBC
  - Dla języka asemblera: **thlqual**.SCSQMACS
  - Dla języka C: **thlqual**.SCSQC370
  - W przypadku języka PL/I: **thlqual**.SCSQPLIC
- W kodzie JCL konsolidacji: program kodu pośredniczącego IBM MQ for z/OS CICS (CSQCSTUB). Rysunek 115 na stronie 1058 pokazuje fragmenty kodu JCL w tym celu. Kod pośredniczący jest niezależny od języka i jest dostarczany w bibliotece **thlqual**.SCSQLOAD.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING CICS STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQCSTUB)
  ⋮
/*
```

*Rysunek 115. Fragmenty kodu JCL do połączenia-edycja modułu obiektu w środowisku CICS*

- W przypadku wersji produktu CICS nowszych niż CICS TS 3.2lub, jeśli mają być używane interfejsy API właściwości komunikatu produktu IBM MQ lub interfejsy API produktu IBM MQ MQCB, MQCTL, MQSTAT, MQSUB lub MQSUBR, należy dokonać konsolidacji kodu obiektu z dostarczonym przez CICS kodem pośredniczącym, DFHMQSTB, a nie z dostarczonym przez IBM MQ kodem pośredniczącym CSQCSTUB. Więcej informacji na temat budowania programów IBM MQ dla produktu CICSzawiera sekcja Program pośredniczący interfejsu API do uzyskiwania dostępu do wywołań MQI produktu IBM MQ w dokumentacji produktu CICS .

Po wykonaniu tych kroków należy zapisać moduł ładujący w bibliotece dynamicznej aplikacji i zdefiniować program w języku CICS w zwykły sposób.

Przed uruchomieniem programu CICS administrator systemu musi zdefiniować go jako CICS program i transakcję IBM MQ . Następnie można go uruchomić w typowy sposób.

z/OS *Budowanie aplikacji IMS (BMP lub MPP)*

Te informacje są przydatne podczas budowania aplikacji IMS (BMP lub MPP).

W przypadku budowania programów wsadowych DL/I należy zapoznać się z sekcją “Budowanie aplikacji wsadowych z/OS” na stronie 1054. Aby zbudować inne aplikacje działające w systemie IMS (jako BMP lub MPP), należy utworzyć zadanie JCL wykonujące następujące zadania:

1. Skompiluj (lub złóż) program, aby utworzyć kod wynikowy. Kod JCL dla kompilacji musi zawierać instrukcje SYSLIB, które udostępniają kompilatorowi pliki definicji danych produktu. Definicje danych są dostarczane w następujących bibliotekach IBM MQ for z/OS :
   - W przypadku języka COBOL: **thlqual**.SCSQCOBC
   - W przypadku języka asemblera: **thlqual**.SCSQMACS
   - Dla języka C: **thlqual**.SCSQC370
   - W przypadku języka PL/I: **thlqual**.SCSQPLIC
2. W przypadku aplikacji w języku C należy wstępnie dowiązać moduł obiektu utworzony w kroku “1” na stronie 1058.
3. W przypadku programów w języku PL/I należy użyć opcji kompilatora EXTRN (SHORT).
4. W przypadku aplikacji w języku C, jeśli aplikacja nie używa kodu XPLINK, należy użyć opcji kompilatora DEFINE (MQ_OS_LINKAGE=1).

5. Przeprowadź edycję kodu obiektu utworzonego w kroku “1” na stronie 1058 (lub w kroku “2” na stronie 1058 dla aplikacji C/370 ), aby utworzyć moduł ładujący:
   a. Dołącz moduł interfejsu języka IMS (DFSLI000).
   b. Dołącz program pośredniczący produktu IBM MQ for z/OS IMS (CSQQSTUB). Rysunek 116 na stronie 1059 pokazuje fragmenty kodu JCL w celu wykonania tej czynności. Kod pośredniczący jest niezależny od języka i jest dostarczany w bibliotece **thlqual**.SCSQLOAD.

      **Uwaga:** Jeśli używany jest język COBOL, należy wybrać opcję kompilatora NODYNAM, aby umożliwić programowi łączącemu rozstrzyganie odwołań do kodu pośredniczącego CSQQSTUB, chyba że planowane jest użycie łączenia dynamicznego zgodnie z opisem w sekcji “Dynamiczne wywoływanie kodu pośredniczącego IBM MQ” na stronie 1060.
6. Zapisz moduł ładujący w bibliotece dynamicznej aplikacji.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING IMS STUB
//*
//CSQSTUB   DD  DSN=thlqual.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQQSTUB)
  ⋮
/*
```

*Rysunek 116. Fragmenty kodu JCL do połączenia-edycja modułu obiektu w środowisku IMS*

Przed uruchomieniem programu IMS administrator systemu musi zdefiniować program IMS jako program IBM MQ i transakcję: można go uruchomić w typowy sposób.

z/OS *Tworzenie aplikacji z/OS UNIX System Services*

Te informacje są przydatne podczas budowania aplikacji z/OS UNIX System Services .

Aby zbudować aplikację w języku C dla systemu IBM MQ for z/OS , która działa w systemie z/OS UNIX System Services, skompiluj i powiąż aplikację w następujący sposób:

```
cc -o mqsamp -W c,DLL -I "//' thlqual.SCSQC370'" mqsamp.c "//' thlqual.SCSQDEFS(CSQBMQ1)'"
```

gdzie **thlqual** jest kwalifikatorem wysokiego poziomu używanym w danej instalacji.

Aby uruchomić program w języku C, należy dodać następujący wpis do pliku `.profile` . Plik ten powinien znajdować się w katalogu głównym:

```
STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

Należy zauważyć, że aby zmiana została rozpoznana, należy wyjść z programu z/OS UNIX System Servicesi ponownie wprowadzić komendę z/OS UNIX System Services .

Aby uruchomić wiele powłok, dodaj słowo export na początku wiersza, czyli:

```
export STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

Po pomyślnym zakończeniu tej czynności można połączyć wywołania CSQBSTUB i IBM MQ .

W sekcji “Dynamiczne wywoływanie kodu pośredniczącego IBM MQ” na stronie 1060 opisano alternatywną metodę tworzenia wywołań MQI w programach, dzięki czemu nie ma potrzeby konsolidacji kodu pośredniczącego IBM MQ . Ta metoda nie jest dostępna we wszystkich językach i środowiskach.

Nie należy dowiązywać-edytować wyższego poziomu programu pośredniczącego niż wersja produktu IBM MQ for z/OS , na której jest uruchomiony program. Na przykład program działający w systemie IBM

WebSphere MQ for z/OS 7.1 nie może być edytowany za pomocą odsyłacza do kodu pośredniczącego dostarczanego z produktem IBM MQ for z/OS 8.0.

### z/OS *Dynamiczne wywoływanie kodu pośredniczącego IBM MQ*

Zamiast konsolidacji programu pośredniczącego IBM MQ za pomocą kodu obiektu, można dynamicznie wywołać kod pośredniczący z poziomu programu.

Można to zrobić w środowiskach wsadowych, IMSi CICS . Ta funkcja nie jest obsługiwana w środowisku RRS. Jeśli aplikacja używa usługi RRS do koordynowania aktualizacji, należy zapoznać się z sekcją "Uwagi dotyczące usługi RRS" na stronie 1064.

Jednak ta metoda:

- Zwiększa złożoność programów
- Zwiększa ilość pamięci wymaganej przez programy w czasie wykonywania
- Zmniejsza wydajność programów
- Oznacza, że nie można używać tych samych programów w innych środowiskach.

Jeśli kod pośredniczący jest wywoływany dynamicznie, odpowiedni program pośredniczący i jego aliasy muszą być dostępne w czasie wykonywania. Aby to zapewnić, należy uwzględnić SCSQLOAD zestawu danych IBM MQ for z/OS :

- W przypadku zadań wsadowych i IMSw konkatenacji STEPLIB JCL.
- W przypadku systemu CICS-w konkatenacji CICS DFHRPL.

W przypadku produktu IMSnależy upewnić się, że biblioteka zawierająca dynamiczny kod pośredniczący (zbudowana zgodnie z opisem zawartym w informacjach dotyczących instalowania adaptera IMS w sekcji Konfigurowanie adaptera IMS ) jest przed zestawem danych SCSQLOAD w konkatenacji STEPLIB kodu JCL regionu.

Podczas dynamicznego wywoływania kodu pośredniczącego należy używać nazw wyświetlanych w pliku Tabela 152 na stronie 1060 . W języku PL/I deklaruj tylko nazwy wywołań używane w programie.

*Tabela 152. Nazwy wywołań dla połączeń dynamicznych*

| **Wywołanie MQI** | **Nazwy dynamicznych wywołań wsadowych (innych niż RRS)** | **Nazwy wywołań dynamicznych CICS** | **Nazwy wywołań dynamicznych IMS** |
|---|---|---|---|
| **MQBACK** | KSQBBACK | nieobsługiwane | Nieobsługiwane |
| **MQBUFMH** | CSQBBFMH | CSQCBFMH [1] | MQBUFMH |
| **Baza MQCB** | CSQBCB | CSQCCB [1] | Nieobsługiwane |
| **MQCLOSE** | KSQBKLOS | KSQCCLOS | MQCLOSE |
| **MQCMIT (MQCMIT)** | CSQBCOMM, | nieobsługiwane | Nieobsługiwane |
| **MQCONN** | CSQBCONN | CSQCCONN | ZMQCONN |
| **MQCONNX (usługa MQCONNX)** | CSQBCONX | CSQCCONX | MQCONNX |
| **MQCRTMH** | CSQBCTMH | CSQCCTMH [1] | MQCRTMH |
| **MQCTL (MQCTL)** | CSQBCTL | CSQCCTL [1] | Nieobsługiwane |
| **MQDISC** | CSQBDISC | CSQCDISC | MQDISC |
| **MQDLTMH** | CSQBDTMH | CSQCDTMH [1] | MQDLTMH |
| **MQDLTMP** | CSQBDTMP | CSQCDTMP [1] | Komenda MQDLTMP |
| **MQGet** | CSQBGET | CSQCGET | MQGET |

| *Tabela 152. Nazwy wywołań dla połączeń dynamicznych (kontynuacja)* | | | |
|---|---|---|---|
| **Wywołanie MQI** | **Nazwy dynamicznych wywołań wsadowych (innych niż RRS)** | **Nazwy wywołań dynamicznych CICS** | **Nazwy wywołań dynamicznych IMS** |
| **MQINQ** | KSQBINQ | CSQCINQ | MQINQ |
| **MQINQMP** | CSQBIKMP | CSQCIQMP [1] | MQINQMP |
| **MQMHBUF** | CSQBMHBF, | CSQCMHBF [1] | MQMHBUF |
| **MQOPEN** | KSQBOPEN | KSQKOPEN | MQOPEN |
| **MQPUT** | KSQBPUT | CSQCPUT | MQPUT |
| **MQPUT1** | CSQBPUT1 | CSQCPUT1 | MQPUT1 |
| **MQSET** | CSQBSET, | CSQCSET, | MQSET |
| **MQSETMP (komenda MQSETMP)** | CSQBSTMP | CSQCSTMP [1] | Komenda MQSETMP |
| **MQSTAT (tabela MQSTAT)** | CSQBSTAT | CSQCSTAT [1] | MQSTAT |
| **MQSUB (MQSUB)** | CSQBSUB | CSQCSUB [1] | MQSUB |
| **MQSUBRQ (MQSUBRQ)** | KSQBSUBR | CSQCSUBR [1] | MQSUBRQ |

**Uwaga:** 1. Te wywołania interfejsu API są dostępne tylko wtedy, gdy używany jest produkt CICS TS 3.2 lub nowszy, a kod CSQCSTUB dostarczany z produktem CICS musi być używany. W przypadku systemu CICS TS 3.2należy zastosować poprawkę APAR PK66866 . W przypadku systemu CICS TS 4.1należy zastosować poprawkę APAR PK89844 .

Przykłady użycia tej techniki znajdują się na poniższych rysunkach:

- Zadania wsadowe i COBOL: patrz Rysunek 117 na stronie 1062
- CICS i COBOL: patrz Rysunek 118 na stronie 1062
- IMS i COBOL: patrz Rysunek 119 na stronie 1062
- Zadanie wsadowe i asembler: patrz Rysunek 120 na stronie 1063
- CICS i asembler: patrz Rysunek 121 na stronie 1063
- IMS i asembler: patrz Rysunek 122 na stronie 1063
- Partia i C: Rysunek 123 na stronie 1063
- CICS i C: patrz Rysunek 124 na stronie 1063
- IMS i C: patrz Rysunek 125 na stronie 1064
- Batch i PL/I: patrz Rysunek 126 na stronie 1064
- IMS i PL/I: patrz Rysunek 127 na stronie 1064

```
...
       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.
...
```

*Rysunek 117. Dynamiczne łączenie przy użyciu języka COBOL w środowisku wsadowym*

```
...
       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.
...
```

*Rysunek 118. Dynamiczne łączenie przy użyciu języka COBOL w środowisku CICS*

```
...
       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.
...
```

*Rysunek 119. Dynamiczne łączenie przy użyciu języka COBOL w środowisku IMS*

```
...
         LOAD   EP=CSQBOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=CSQBOPEN
...
```

*Rysunek 120. Dynamiczne łączenie przy użyciu języka asemblacji w środowisku wsadowym*

```
...
         EXEC CICS LOAD PROGRAM('CSQCOPEN') ENTRY(R15)
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         EXEC CICS RELEASE PROGRAM('CSQCOPEN')
...
```

*Rysunek 121. Dynamiczne łączenie przy użyciu języka asemblacji w środowisku CICS*

```
...
         LOAD   EP=MQOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=MQOPEN
...
```

*Rysunek 122. Dynamiczne łączenie przy użyciu języka asemblacji w środowisku IMS*

```
...
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);
...
```

*Rysunek 123. Dynamiczne łączenie przy użyciu języka C w środowisku wsadowym*

```
...
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);
...
```

*Rysunek 124. Dynamiczne łączenie przy użyciu języka C w środowisku CICS*

```
...
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);
...
```

*Rysunek 125. Dynamiczne łączenie przy użyciu języka C w środowisku IMS*

```
...
      DCL CSQBOPEN ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH CSQBOPEN;

      CALL CSQBOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE CSQBOPEN;
```

*Rysunek 126. Dynamiczne łączenie za pomocą języka PL/I w środowisku wsadowym*

```
...
      DCL MQOPEN   ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH MQOPEN;

      CALL   MQOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE   MQOPEN;
```

*Rysunek 127. Dynamiczne łączenie za pomocą języka PL/I w środowisku IMS*

z/OS *Uwagi dotyczące usługi RRS*

Należy rozważyć użycie tych informacji, jeśli aplikacja używa usługi RRS do koordynowania aktualizacji.

Produkt IBM MQ udostępnia dwa różne kody pośredniczące dla programów wsadowych, które wymagają koordynacji RRS-patrz sekcja “Adapter zadania wsadowego RRS” na stronie 919. Różnica w zachowaniu późniejszych wywołań API jest określana w czasie MQCONN przez adapter wsadowy na podstawie informacji przekazywanych przez podprogram pośredniczący w interfejsie API MQCONN lub MQCONNX. Oznacza to, że dynamiczne wywołania API są dostępne dla programów wsadowych, które wymagają koordynacji RRS, pod warunkiem, że początkowe połączenie z produktem IBM MQ zostało nawiązane przy użyciu odpowiedniego kodu pośredniczącego. Ilustruje to następujący przykład:

```
       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 CSQBRRSI 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.
```

## z/OS *Debugowanie programów*

Te informacje umożliwiają zapoznanie się z informacjami na temat debugowania programów TSO i CICS oraz wglądu w dane śledzenia CICS .

Głównymi pomocami przy debugowaniu aplikacji IBM MQ for z/OS są kody przyczyny zwracane przez każde wywołanie funkcji API. Listę tych działań, wraz z pomysłami na działania naprawcze, można znaleźć pod adresem:

- Komunikaty systemu IBM MQ for z/OS , kody zakończenia i kody przyczyny dla IBM MQ for z/OS
- Komunikaty i kody przyczyny dla wszystkich innych platform IBM MQ

Ten temat zawiera również sugestie dotyczące innych narzędzi do debugowania, które mogą być używane w konkretnych środowiskach.

### Debugowanie programów TSO

Dla programów TSO dostępne są następujące interaktywne narzędzia debugowania:

- TEST, narzędzie
- Interaktywne narzędzie debugowania VS COBOL II
- Interaktywne narzędzie debugowania INSPECT dla programów w językach C i PL/I

### Debugowanie programów CICS

Do interaktywnego testowania programów CICS bez konieczności modyfikowania programu lub procedury przygotowania programu można użyć narzędzia CICS Execution Diagnostic Facility (CEDF).

Więcej informacji na temat EDF zawiera publikacja *CICS Transaction Server for z/OS CICS Application Programming Guide*.

### CICS ślad

Pomocne może być również użycie transakcji CICS Sterowanie śledzeniem (Trace Control Transaction-CETR) do sterowania działaniem śledzenia CICS .

Więcej informacji na temat programu CETR zawiera podręcznik *CICS Transaction Server for z/OS CICS-Suppl-Transactions* .

Aby określić, czy śledzenie CICS jest aktywne, wyświetl status połączenia za pomocą panelu CKQC. Na tym panelu wyświetlany jest również numer śledzenia.

Informacje na temat interpretowania pozycji śledzenia CICS zawiera sekcja Tabela 153 na stronie 1066.

Pozycja śledzenia CICS dla tych wartości to AP0 *xxx* (gdzie *xxx* jest numerem śledzenia określonym podczas włączania adaptera CICS ). Wszystkie pozycje śledzenia z wyjątkiem CSQCTEST są wysyłane przez CSQCTRUE. Komenda CSQCTEST jest wystawiana przez komendy CSQCRST i CSQCDSP.

*Tabela 153. Pozycje śledzenia adaptera CICS*

| Nazwa | Opis | Sekwencja śledzenia | Dane śledzenia |
|---|---|---|---|
| KSQCABNT | Nieprawidłowe zakończenie | Przed wysłaniem komendy END_THREAD ABNORMAL do IBM MQ. Jest to spowodowane zakończeniem zadania i niejawnym wycofaniem może być wykonane przez aplikację. W tym przypadku żądanie ROLLBACK jest zawarte w wywołaniu END_THREAD. | Informacje o jednostce pracy. Informacji tych można użyć podczas sprawdzania statusu pracy. (Na przykład można go sprawdzić względem danych wyjściowych wygenerowanych przez komendę DISPLAY THREAD lub program narzędziowy do drukowania dziennika IBM MQ for z/OS ). |
| CSQCBACK | Wycofanie punktu synchronizacji | Przed wysłaniem komendy BACKOUT do IBM MQ for z/OS. Jest to spowodowane jawnym żądaniem wycofania z aplikacji. | Informacje o jednostce pracy. |
| CSQCCCRC | Kod zakończenia i kod przyczyny | Po nieudanym powrocie z wywołania API. | Kod zakończenia i kod przyczyny. |
| CSQCCOMM, | Zatwierdzenie punktu synchronizacji | Przed wykonaniem instrukcji COMMIT dla komendy IBM MQ for z/OS. Może to być spowodowane żądaniem zatwierdzania jednofazowego lub drugą fazą żądania zatwierdzania dwufazowego. Żądanie jest spowodowane jawnym żądaniem punktu synchronizacji z aplikacji. | Informacje o jednostce pracy. |
| KSQCEXER | Wykonaj rozstrzyganie | Przed wydaniem komendy EXECUTE_RESOLVE do IBM MQ for z/OS. | Informacje o jednostce pracy wydającej komendę EXECUTE_RESOLVE. Jest to ostatnia wątpliwa jednostka pracy w procesie resynchronizacji. |
| CSQCGETW | Oczekiwanie GET | Przed wysłaniem komendy CICS należy poczekać. | Adres EBC, na który należy czekać. |
| CSQCGMGD | Dane komunikatu GET | Po pomyślnym powrocie z MQGET. | Do 40 bajtów danych komunikatu. |
| CSQCGMGH | Uchwyt komunikatu GET | Przed wysłaniem komendy MQGET do IBM MQ for z/OS. | Uchwyt obiektu. |
| CSQCGMGI | Pobierz identyfikator komunikatu | Po pomyślnym powrocie z MQGET. | Identyfikator komunikatu i identyfikator korelacji komunikatu. |
| CSQCINDLStencils | Lista wątpliwych | Po pomyślnym powrocie z drugiej operacji INQUIRE_INDOUBT. | Lista wątpliwych jednostek pracy. |
| CSQCINDOConan (I) | Tylko do użytku w systemie IBM | | |

*Tabela 153. Pozycje śledzenia adaptera CICS (kontynuacja)*

| Nazwa | Opis | Sekwencja śledzenia | Dane śledzenia |
|---|---|---|---|
| KSQCINDS | Wielkość listy wątpliwych | Po pomyślnym powrocie z pierwszej listy INQUIRE_INDOUBT lista ta nie jest pusta. | Długość listy. Podzielone przez 64 daje liczbę wątpliwych jednostek pracy. |
| CSQCINQH | Uchwyt INQ | Przed wysłaniem komendy MQINQ do IBM MQ for z/OS. | Uchwyt obiektu. |
| KSQCLOSH | Uchwyt CLOSE | Przed wysłaniem komendy MQCLOSE do IBM MQ for z/OS. | Uchwyt obiektu. |
| CSQCLOST@ item: inmenu | Dyspozycja utracona | Podczas procesu resynchronizacji CICS informuje adapter, że został zrestartowany, dlatego nie są dostępne informacje o resynchronizacji jednostki pracy. | Identyfikator jednostki pracy znany CICS dla resynchronizowanej jednostki pracy. |
| KSQCNIND | Dyspozycja nie jest wątpliwa | Podczas procesu resynchronizacji CICS informuje adapter, że jednostka pracy, która jest resynchronizowana, nie powinna być wątpliwa (być może nadal działa). | Identyfikator jednostki pracy znany CICS dla resynchronizowanej jednostki pracy. |
| KSQCNORT | Normalne zakończenie | Przed wysłaniem komendy END_THREAD NORMAL do IBM MQ for z/OS. Jest to spowodowane zakończeniem zadania i dlatego aplikacja może wykonać niejawne przekazanie do publikacji w punkcie synchronizacji. W tym przypadku do wywołania END_THREAD dołączane jest żądanie COMMIT. | Informacje o jednostce pracy. |
| CSQCOPNH | Uchwyt OPEN | Po pomyślnym powrocie z MQOPEN. | Uchwyt obiektu. |
| CSQCOPNO | OPEN, obiekt | Przed wysłaniem komendy MQOPEN do IBM MQ for z/OS. | Nazwa obiektu. |
| CSQCPMGD, | Dane komunikatu PUT | Przed wysłaniem MQPUT do IBM MQ for z/OS. | Do 40 bajtów danych komunikatu. |
| CSQCPMGH | Uchwyt komunikatu PUT | Przed wysłaniem MQPUT do IBM MQ for z/OS. | Uchwyt obiektu. |
| CSQCPMGI | Identyfikator komunikatu PUT | Po pomyślnym wykonaniu operacji MQPUT z produktu IBM MQ for z/OS. | Identyfikator komunikatu i identyfikator korelacji komunikatu. |

*Tabela 153. Pozycje śledzenia adaptera CICS (kontynuacja)*

| Nazwa | Opis | Sekwencja śledzenia | Dane śledzenia |
|---|---|---|---|
| CSQCPREP, | Przygotowanie punktu synchronizacji | Przed wydaniem komendy PREPARE dla IBM MQ for z/OS w pierwszej fazie zatwierdzania dwufazowego. To wywołanie można również wywołać z komponentu kolejkowania rozproszonego jako wywołanie interfejsu API. | Informacje o jednostce pracy. |
| CSQCP1MD | Dane komunikatu PUTONE | Przed wprowadzeniem komendy MQPUT1 do programu IBM MQ for z/OS. | Do 40 bajtów danych komunikatu. |
| CSQCP1MI | Identyfikator komunikatu PUTONE | Po pomyślnym powrocie z operacji MQPUT1. | Identyfikator komunikatu i identyfikator korelacji komunikatu. |
| CSQCP1ON | Nazwa obiektu PUTONE | Przed wprowadzeniem komendy MQPUT1 do programu IBM MQ for z/OS. | Nazwa obiektu. |
| KSQCRBAK | Rozwiązane wycofanie | Przed wykonaniem komendy RESOLVE_ROLLBACK na IBM MQ for z/OS. | Informacje o jednostce pracy. |
| CSQCRCMT | Rozstrzygnięte zatwierdzenie | Przed wydaniem komendy RESOLVE_COMMIT w systemie IBM MQ for z/OS. | Informacje o jednostce pracy. |
| CSQCRMIR | Odpowiedź RMI | Przed powrotem do interfejsu CICS RMI (resource manager interface) z konkretnego wywołania. | Wartość architected odpowiedzi RMI. Jego znaczenie zależy od typu wywołania. Te wartości są udokumentowane w podręczniku *CICS Transaction Server for z/OS Customization Guide*. Aby określić typ wywołania, należy przejrzeć poprzednie pozycje śledzenia wygenerowane przez komponent RMI produktu CICS . |
| CSQCRSYN | Ponowna synchronizacja | Przed uruchomieniem procesu resynchronizacji dla zadania. | Identyfikator jednostki pracy znany CICS dla resynchronizowanej jednostki pracy. |
| KSQCSETH | Uchwyt SET | Przed wysłaniem komendy MQSET do IBM MQ for z/OS. | Uchwyt obiektu. |
| KSQCTASE | Tylko do użytku w systemie IBM | | |
| CSQCTEST | Test śledzenia | Używany w wywołaniu EXEC CICS ENTER TRACE w celu zweryfikowania numeru śledzenia podanego przez użytkownika lub statusu śledzenia połączenia. | Brak danych. |

| Tabela 153. Pozycje śledzenia adaptera CICS (kontynuacja) | | | |
|---|---|---|---|
| **Nazwa** | **Opis** | **Sekwencja śledzenia** | **Dane śledzenia** |
| CSQCDCFF, | Tylko do użytku w systemie IBM | | |

# Obsługa błędów proceduralnych programu

Te informacje wyjaśniają błędy powiązane z wywołaniami MQI aplikacji podczas wykonywania wywołania lub po dostarczeniu komunikatu do miejsca docelowego.

Jeśli jest to możliwe, menedżer kolejek zwraca wszystkie błędy natychmiast po wywołaniu MQI. Są to *lokalnie określone błędy*.

Podczas wysyłania komunikatów do kolejki zdalnej błędy mogą nie być widoczne po wykonaniu wywołania MQI. W takim przypadku menedżer kolejek, który identyfikuje błędy, zgłasza je, wysyłając inny komunikat do programu źródłowego. Są to *błędy określane zdalnie*.

## Błędy określone lokalnie

Informacje o lokalnie określonych błędach, które obejmują: niepowodzenie wywołania MQI, przerwania systemowe i komunikaty zawierające niepoprawne dane.

Trzy najczęstsze przyczyny błędów, które menedżer kolejek może natychmiast zgłosić, to:

- Niepowodzenie wywołania MQI, na przykład z powodu zapełnienia kolejki
- Przerwanie działania części systemu, od której zależy aplikacja, na przykład menedżera kolejek.
- Komunikaty zawierające dane, których nie można pomyślnie przetworzyć

Jeśli używane jest asynchroniczne narzędzie umieszczania, błędy nie są zgłaszane natychmiast. Wywołanie MQSTAT służy do pobierania informacji o statusie poprzednich asynchronicznych operacji umieszczania.

### Niepowodzenie wywołania MQI

Menedżer kolejek może natychmiast zgłosić wszystkie błędy w kodowaniu wywołania MQI. W tym celu używany jest zestaw predefiniowanych kodów powrotu. Są one podzielone na kody zakończenia i kody przyczyny.

Aby pokazać, czy wywołanie zakończyło się pomyślnie, menedżer kolejek zwraca *kod zakończenia* po zakończeniu wywołania. Istnieją trzy kody zakończenia wskazujące powodzenie, częściowe zakończenie i niepowodzenie wywołania. Menedżer kolejek zwraca również *kod przyczyny* , który wskazuje przyczynę częściowego zakończenia lub niepowodzenia wywołania.

Kody zakończenia i przyczyny dla każdego wywołania są wymienione z opisem tego wywołania w sekcji Kody powrotu. Bardziej szczegółowe informacje, w tym pomysły na działania naprawcze, znajdują się w następujących sekcjach:

- z/OS Komunikaty systemu IBM MQ for z/OS , kody zakończenia i kody przyczyny dla IBM MQ for z/OS
- Komunikaty i kody przyczyny dla wszystkich innych platform IBM MQ

Zaprojektuj programy do obsługi wszystkich kodów powrotu, które mogą wynikać z każdego wywołania.

### System interruptions (liczba opcji)

Aplikacja może nie mieć informacji o przerwie, jeśli menedżer kolejek, z którym jest połączona, musi przeprowadzić odtwarzanie po awarii systemu. Należy jednak zaprojektować aplikację, aby upewnić się, że dane nie zostaną utracone w przypadku wystąpienia takiej przerwy.

Metody, których można użyć do zapewnienia spójności danych, zależą od platformy, na której działa menedżer kolejek:

**z/OS**

W środowiskach CICS i IMS można wykonywać wywołania MQPUT i MQGET w obrębie jednostek pracy, które są zarządzane przez produkt CICS lub IMS. W środowisku wsadowym można wykonywać wywołania MQPUT i MQGET w ten sam sposób, ale należy zadeklarować punkty synchronizacji przy użyciu:

- IBM MQ for z/OS Wywołania MQCMIT i MQBACK (patrz sekcja “Zatwierdzanie i wycofywanie jednostek pracy” na stronie 879 ) lub
- Usługi RRS ( z/OS Transaction Management and Recoverable Resource Manager Resource Services) zapewniają obsługę punktów synchronizacji dwufazowej. Usługa RRS umożliwia aktualizowanie zarówno zasobów produktu IBM MQ , jak i innych zasobów produktu obsługujących usługi RRS, takich jak zasoby procedury składowanej Db2 , w ramach pojedynczej logicznej jednostki pracy. Informacje na temat obsługi punktów synchronizacji RRS zawiera sekcja “Usługi zarządzania transakcjami i odtwarzalnego menedżera zasobów” na stronie 884.

**IBM i**

Wywołania MQPUT i MQGET można tworzyć w obrębie globalnych jednostek pracy, które są zarządzane przez kontrolę transakcji IBM i . Punkty synchronizacji można deklarować za pomocą rodzimych komend COMMIT i ROLLBACK systemu IBM i lub komend specyficznych dla języka. Lokalne jednostki pracy są zarządzane przez produkt IBM MQ przy użyciu wywołań MQCMIT i MQBACK.

**Systemy AIX, Linux, and Windows**

W tych środowiskach można wykonywać wywołania MQPUT i MQGET w zwykły sposób, ale należy zadeklarować punkty synchronizacji przy użyciu wywołań MQCMIT i MQBACK (patrz sekcja “Zatwierdzanie i wycofywanie jednostek pracy” na stronie 879 ). W środowisku CICS komendy MQCMIT i MQBACK są wyłączone, ponieważ można tworzyć wywołania MQPUT i MQGET w obrębie jednostek pracy, które są zarządzane przez produkt CICS.

Do przenoszenia wszystkich danych, których nie można stracić, należy używać trwałych komunikatów. Trwałe komunikaty są przywracane w kolejkach, jeśli menedżer kolejek musi wykonać odtwarzanie po awarii. ALW W przypadku produktu IBM MQ w systemie AIX, Linux, and Windowswywołanie MQGET lub MQPUT w aplikacji nie powiedzie się w momencie zapełnienia wszystkich plików dziennika. Zostanie wyświetlony komunikat MQRC_RESOURCE_PROBLEM. Więcej informacji na temat plików dziennika w systemie AIX, Linux, and Windowszawiera sekcja Administrowanie programem IBM MQ.
z/OS W przypadku systemu z/OS patrz sekcja Planowanie w systemie z/OS.

Jeśli menedżer kolejek jest zatrzymywany przez operatora podczas działania aplikacji, zwykle używana jest opcja wyciszania. Menedżer kolejek przechodzi w stan wyciszania, w którym aplikacje mogą kontynuować pracę, ale muszą zostać zakończone w dogodnym momencie. Małe, szybkie aplikacje mogą prawdopodobnie ignorować stan wyciszania i kontynuować działanie, dopóki nie zostaną zakończone w normalny sposób. Aplikacje działające dłużej, lub aplikacje oczekujące na nadejście komunikatów, powinny używać opcji *fail if quiescing* , gdy korzystają z wywołań MQOPEN, MQPUT, MQPUT1i MQGET. Te opcje oznaczają, że wywołania nie powiodą się po wyciszeniu menedżera kolejek, ale aplikacja może nadal mieć czas na czyste zakończenie przez wywołanie wywołań, które ignorują stan wyciszania. Takie aplikacje mogą również zatwierdzić lub wycofać zmiany, które dokonały, a następnie zakończyć działanie.

Jeśli zostanie wymuszone zatrzymanie menedżera kolejek (czyli zatrzymanie bez wyciszania), aplikacje otrzymają kod przyczyny MQRC_CONNECTION_BROKEN podczas wykonywania wywołań MQI. Wyjdź z aplikacji lub, alternatywnie, w systemach IBM i IBM MQ for IBM i, AIX, Linux, and Windows wywołaj MQDISC.

## Komunikaty zawierające niepoprawne dane

Jeśli w aplikacji używane są jednostki pracy, a program nie może pomyślnie przetworzyć komunikatu pobranego z kolejki, wywołanie MQGET zostaje wycofane.

Menedżer kolejek utrzymuje liczbę wystąpień (w polu *BackoutCount* deskryptora komunikatu). Ten licznik jest przechowywany w deskryptorze każdego komunikatu, którego dotyczy problem. Ta liczba może dostarczyć cennych informacji na temat wydajności aplikacji. Komunikaty z licznikami wycofanych komunikatów, które z biegiem czasu rosną, są wielokrotnie odrzucane. Należy zaprojektować aplikację w taki sposób, aby analizała przyczyny tego stanu i odpowiednio obsługiwała takie komunikaty.

z/OS W systemie IBM MQ for z/OS, aby liczba wycofanych komunikatów była w stanie przetrwać restarty menedżera kolejek, należy ustawić atrybut **HardenGetBackout** na wartość MQQA_BACKOUT_HARDENED; w przeciwnym razie, jeśli menedżer kolejek musi zostać zrestartowany, nie będzie zachowywał dokładnej liczby wycofań dla każdego komunikatu. Ustawienie atrybutu w ten sposób powoduje dodanie kary za dodatkowe przetwarzanie.

W systemach IBM MQ for IBM i IBM ii AIX, Linux, and Windows liczba wycofań zawsze przetrwa restarty menedżera kolejek.

z/OS Ponadto w systemie IBM MQ for z/OSpodczas usuwania komunikatów z kolejki w ramach jednostki pracy można oznaczyć jeden komunikat, aby nie był ponownie dostępny, jeśli jednostka pracy zostanie wycofana przez aplikację. Oznaczona wiadomość jest traktowana tak, jakby została pobrana w ramach nowej jednostki pracy. Komunikat, który ma zostać pominięty przy użyciu opcji MQGMO_MARK_SKIP_BACKOUT, jest oznaczany jako komunikat, który ma zostać pominięty.(w strukturze MQGMO), gdy używane jest wywołanie MQGET. Więcej informacji na temat tej techniki zawiera sekcja “Pomijanie wycofywania” na stronie 824 .

## Używanie komunikatów raportów do określania problemu

Menedżer kolejek zdalnych nie może zgłaszać błędów, takich jak niepowodzenie umieszczenia komunikatu w kolejce podczas wykonywania wywołania MQI, ale może wysłać komunikat raportu informujący, w jaki sposób komunikat został przetworzony.

W obrębie aplikacji można tworzyć komunikaty raportów (MQPUT), a także wybierać opcję ich odbierania (w takim przypadku są one wysyłane przez inną aplikację lub przez menedżer kolejek).

### Tworzenie komunikatów raportu

Komunikaty raportu umożliwiają aplikacji poinformowanie innej aplikacji, że nie może ona zająć się wysłanym komunikatem.

Jednak pole *Report* musi zostać wstępnie przeanalizowane w celu określenia, czy aplikacja, która wysłała komunikat, ma zostać poinformowana o ewentualnych problemach. Po ustaleniu, że komunikat raportu jest wymagany, należy podjąć decyzję:

- Określa, czy ma zostać dołączona cała oryginalna wiadomość, tylko pierwsze 100 bajtów danych, czy też nie ma być dołączona żadna oryginalna wiadomość.
- Co zrobić z oryginalnym komunikatem. Można go usunąć lub pozwolić mu przejść do kolejki niedostarczonych komunikatów.
- Określa, czy wymagana jest również zawartość pól *MsgId* i *CorrelId* .

Użyj pola *Feedback* , aby wskazać przyczynę wygenerowania komunikatu raportu. Umieść komunikaty raportu w kolejce odpowiedzi aplikacji. Więcej informacji na ten temat zawiera sekcja Opinia .

### Wysyłanie żądań i odbieranie komunikatów raportu (MQGET)

Po wysłaniu komunikatu do innej aplikacji użytkownik nie jest informowany o żadnych problemach, chyba że wypełni pole *Report* w celu wskazania wymaganej opinii. Dostępne opcje znajdują się w sekcji Struktura pola raportu .

Menedżery kolejek zawsze umieszczają komunikaty raportów w kolejce odpowiedzi aplikacji i zaleca się, aby własne aplikacje robiły to samo. Jeśli używane jest narzędzie raportowania komunikatów, należy określić nazwę kolejki odpowiedzi w deskryptorze komunikatu. W przeciwnym razie wywołanie MQPUT nie powiedzie się.

Aplikacja musi zawierać procedury, które monitorują kolejkę odpowiedzi i przetwarzają wszystkie komunikaty, które do niej docierają. Należy pamiętać, że komunikat raportu może zawierać cały oryginalny komunikat, pierwsze 100 bajtów oryginalnego komunikatu lub nie może zawierać żadnego oryginalnego komunikatu.

Menedżer kolejek ustawia pole *Feedback* komunikatu raportu, aby wskazać przyczynę błędu, na przykład kolejka docelowa nie istnieje. Programy powinny robić to samo.

Więcej informacji na temat komunikatów raportów zawiera sekcja "Komunikaty raportu" na stronie 21.

## Błędy określane zdalnie

Jeśli komunikaty są wysyłane do kolejki zdalnej, nawet jeśli lokalny menedżer kolejek przetworzył wywołanie MQI bez znalezienia błędu, inne czynniki mogą mieć wpływ na sposób obsługi komunikatu przez zdalny menedżer kolejek.

Na przykład kolejka, której celem jest użytkownik, może być pełna lub może nawet nie istnieć. Jeśli komunikat musi być obsługiwany przez inne pośrednie menedżery kolejek na trasie do kolejki docelowej, może to spowodować wystąpienie błędu.

### Problemy z dostarczeniem komunikatu

Jeśli wywołanie MQPUT nie powiedzie się, można ponownie umieścić komunikat w kolejce, zwrócić go do nadawcy lub umieścić w kolejce niedostarczonych komunikatów.

Każda opcja ma swoje zalety, ale może nie być potrzebna ponowna próba umieszczenia komunikatu, jeśli przyczyną niepowodzenia operacji MQPUT było zapełnienie kolejki docelowej. W tym przypadku umieszczenie go w kolejce niedostarczonych komunikatów umożliwia późniejsze dostarczenie go do poprawnej kolejki docelowej.

**Ponów próbę dostarczenia komunikatu**

Przed umieszczeniem komunikatu w kolejce niedostarczonych komunikatów menedżer kolejek zdalnych próbuje ponownie umieścić komunikat w kolejce, jeśli dla kanału zostały ustawione atrybuty *MsgRetryCount* i *MsgRetryInterval* lub jeśli istnieje dla niego program obsługi wyjścia ponawiania (którego nazwa jest przechowywana w polu atrybutu kanału *MsgRetryExitId* ).

Jeśli pole *MsgRetryExitId* jest puste, używane są wartości atrybutów *MsgRetryCount* i *MsgRetryInterval* .

Jeśli pole *MsgRetryExitId* nie jest puste, uruchamiany jest program obsługi wyjścia o tej nazwie. Więcej informacji na temat używania własnych programów obsługi wyjścia zawiera sekcja "Kanał-programy obsługi wyjścia dla kanałów przesyłania komunikatów" na stronie 990.

**Zwróć wiadomość do nadawcy**

Zwracasz komunikat do nadawcy, żądając wygenerowania komunikatu raportu w celu uwzględnienia wszystkich oryginalnych komunikatów.

Szczegółowe informacje na temat opcji komunikatów raportu zawiera sekcja "Komunikaty raportu" na stronie 21 .

### *Korzystanie z kolejki niedostarczonych komunikatów (niedostarczonych komunikatów)*

Jeśli menedżer kolejek nie może dostarczyć komunikatu, próbuje umieścić komunikat w swojej kolejce niedostarczonych komunikatów. Ta kolejka powinna zostać zdefiniowana podczas instalowania menedżera kolejek.

Programy mogą korzystać z kolejki niedostarczonych komunikatów w taki sam sposób, w jaki korzysta z niej menedżer kolejek. Nazwę kolejki niedostarczonych komunikatów można znaleźć, otwierając obiekt menedżera kolejek (za pomocą wywołania MQOPEN) i pytając o atrybut **DeadLetterQName** (za pomocą wywołania MQINQ).

Gdy menedżer kolejek umieszcza komunikat w tej kolejce, dodaje nagłówek do komunikatu, którego format jest opisany przez strukturę nagłówka niedostarczonego komunikatu (MQDLH). Więcej informacji

na ten temat zawiera sekcja MQDLH-Dead-letter header(Nagłówek niedostarczonego komunikatu). Ten nagłówek zawiera nazwę kolejki docelowej i przyczynę umieszczenia komunikatu w kolejce niedostarczonych komunikatów. Należy go usunąć i rozwiązać problem, zanim komunikat zostanie umieszczony w żądanej kolejce. Ponadto menedżer kolejek zmienia pole *Format* deskryptora komunikatu (MQMD), aby wskazać, że komunikat zawiera strukturę MQDLH.

## Struktura MQDLH

Zaleca się dodanie struktury MQDLH do wszystkich komunikatów umieszczanych w kolejce niedostarczonych komunikatów. Jeśli jednak ma być używana procedura obsługi niedostarczonych komunikatów udostępniana przez niektóre produkty IBM MQ , do komunikatów należy dodać strukturę MQDLH.

Dodanie nagłówka do komunikatu może spowodować, że komunikat będzie zbyt długi dla kolejki niedostarczonych komunikatów, dlatego zawsze należy się upewnić, że wielkość komunikatów jest mniejsza niż maksymalna dozwolona dla kolejki niedostarczonych komunikatów, o co najmniej wartość stałej MQ_MSG_HEADER_LENGTH. Maksymalna wielkość komunikatów dozwolonych w kolejce jest określana przez wartość atrybutu **MaxMsgLength** kolejki. W przypadku kolejki niedostarczonych komunikatów upewnij się, że ten atrybut jest ustawiony na wartość maksymalną dozwoloną przez menedżer kolejek. Jeśli aplikacja nie może dostarczyć komunikatu, a komunikat jest zbyt długi, aby można go było umieścić w kolejce niedostarczonych komunikatów, należy postępować zgodnie z zaleceniami podanymi w opisie struktury MQDLH.

Upewnij się, że kolejka niedostarczonych komunikatów jest monitorowana i że wszystkie przychodzące do niej komunikaty są przetwarzane. Program obsługi kolejki niedostarczonych komunikatów jest uruchamiany jako program narzędziowy zadania wsadowego i może być używany do wykonywania różnych działań na wybranych komunikatach w kolejce niedostarczonych komunikatów. Więcej informacji na ten temat zawiera sekcja “Przetwarzanie kolejki niedostarczonych komunikatów” na stronie 1073.

Jeśli konwersja danych jest konieczna, menedżer kolejek przekształca informacje w nagłówku, gdy w wywołaniu MQGET używana jest opcja MQGMO_CONVERT. Jeśli proces umieszczający komunikat jest agentem MCA, po nagłówku znajduje się cały tekst oryginalnego komunikatu.

Komunikaty umieszczone w kolejce niedostarczonych komunikatów mogą zostać obcięte, jeśli są zbyt długie dla tej kolejki. Możliwym wskazaniem tej sytuacji są komunikaty w kolejce niedostarczonych komunikatów, których długość jest taka sama, jak wartość atrybutu **MaxMsgLength** kolejki.

*Przetwarzanie kolejki niedostarczonych komunikatów*
Te informacje zawierają ogólne informacje o interfejsie programistycznym używane podczas przetwarzania kolejki niedostarczonych komunikatów.

Przetwarzanie kolejki niedostarczonych komunikatów zależy od lokalnych wymagań systemowych, ale podczas tworzenia specyfikacji należy wziąć pod uwagę następujące kwestie:

- Komunikat można zidentyfikować jako mający nagłówek kolejki niedostarczonych komunikatów, ponieważ wartością pola formatu w deskryptorze MQMD jest MQFMT_DEAD_LETTER_HEADER.
- W systemie IBM MQ for z/OS korzystającym z produktu CICS, jeśli agent MCA umieści ten komunikat w kolejce niedostarczonych komunikatów, pole *PutApplType* to MQAT_CICS, a pole *PutApplName* to *ApplId* systemu CICS , po którym następuje nazwa transakcji agenta MCA.
- Przyczyna, dla której komunikat ma zostać skierowany do kolejki niedostarczonych komunikatów, znajduje się w polu *Reason* nagłówka kolejki niedostarczonych komunikatów.
- Nagłówek kolejki niedostarczonych komunikatów zawiera szczegóły dotyczące nazwy kolejki docelowej i nazwy menedżera kolejek.
- Nagłówek kolejki niedostarczonych komunikatów zawiera pola, które muszą zostać przywrócone w deskryptorze komunikatu przed umieszczeniem komunikatu w kolejce docelowej. Są to:

  1. *Encoding*
  2. *CodedCharSetId*
  3. *Format*

- Deskryptor komunikatu jest taki sam, jak w oryginalnej aplikacji, z wyjątkiem trzech przedstawionych pól (Encoding, CodedCharSetIdi Format).

Aplikacja kolejki niedostarczonych komunikatów musi wykonać co najmniej jedną z następujących czynności:

- Sprawdź zawartość pola *Reason* . Komunikat mógł zostać umieszczony przez agent MCA z następujących powodów:
  - Komunikat był dłuższy niż maksymalna wielkość komunikatu dla kanału

    Przyczyna: MQRC_MSG_TOO_BIG_FOR_CHANNEL
  - Nie można umieścić komunikatu w jego kolejce docelowej

    Przyczyną jest dowolny kod przyczyny MQRC_ *, który może zostać zwrócony przez operację MQPUT .
  - Program użytkownika obsługi wyjścia zażądał tego działania

    Kod przyczyny to kod dostarczony przez program użytkownika lub domyślny MQRC_SUPPRESSED_BY_EXIT
- Spróbuj przekazać komunikat do zamierzonego miejsca docelowego, jeśli jest to możliwe.
- Zachowaj komunikat przez pewien czas przed odrzuceniem, gdy zostanie określona przyczyna przekierowania, ale nie można jej natychmiast skorygować.
- Przekaż instrukcje administratorom, aby rozwiązywały problemy tam, gdzie zostały określone.
- Odrzuć komunikaty, które są uszkodzone lub nie mogą być przetworzone w inny sposób.

Istnieją dwa sposoby postępowania z komunikatami, które zostały odzyskane z kolejki niedostarczonych komunikatów:

1. Jeśli komunikat dotyczy kolejki lokalnej:
   - Wykonaj wszystkie tłumaczenia kodu wymagane do wyodrębnienia danych aplikacji
   - Przeprowadzanie konwersji kodu dla tych danych, jeśli jest to funkcja lokalna
   - Umieść komunikat wynikowy w kolejce lokalnej ze wszystkimi szczegółami odtworzonego deskryptora komunikatu
2. Jeśli komunikat dotyczy kolejki zdalnej, umieść go w kolejce.

Więcej informacji na temat obsługi niedostarczonych komunikatów w rozproszonym środowisku kolejkowania zawiera sekcja Co się dzieje, gdy nie można dostarczyć komunikatu?.

# Programowanie rozsyłania grupowego

Ten temat zawiera informacje o zadaniach programistycznych rozsyłania grupowego produktu IBM MQ , takich jak nawiązywanie połączenia z menedżerem kolejek i raportowanie wyjątków.

Funkcja rozsyłania grupowego w produkcie IBM MQ została zaprojektowana w taki sposób, aby była jak najbardziej przezroczysta dla użytkownika i nadal była kompatybilna z istniejącymi aplikacjami. Zdefiniowanie obiektu COMMINFO i ustawienie parametrów **MCAST** i **COMMINFO** obiektu TOPIC oznacza, że istniejące aplikacje IBM MQ nie wymagają znacznego przebudowywania w celu użycia rozsyłania grupowego. Mogą jednak wystąpić pewne ograniczenia (więcej informacji na ten temat zawiera sekcja “Rozsyłanie grupowe i MQI” na stronie 1074 ) i pewne problemy z bezpieczeństwem, które należy rozważyć (więcej informacji na ten temat zawiera sekcja Zabezpieczenia rozsyłania grupowego ).

## Rozsyłanie grupowe i MQI

Ten temat zawiera informacje dotyczące głównych pojęć związanych z interfejsem kolejki komunikatów (Message Queue Interface-MQI) i ich związku z rzutowaniem rozsyłania w produkcie IBM MQ .

Subskrypcje rozsyłania grupowego są nietrwałe. Ponieważ nie są używane żadne kolejki fizyczne, nie ma miejsca, w którym można przechowywać komunikaty w trybie bez połączenia, które są tworzone przez trwałe subskrypcje.

Po zasubskrybowaniu przez aplikację tematu rozsyłania grupowego jest ona oddawana z powrotem do uchwytu obiektu, z którego może korzystać lub z wywołania MQGET, tak jakby była uchwytem kolejki. Oznacza to, że obsługiwane są tylko zarządzane subskrypcje rozsyłania grupowego (subskrypcje utworzone za pomocą komendy MQSO_MANAGED). Oznacza to, że nie można utworzyć subskrypcji i wskazać komunikatów w kolejce. Oznacza to, że komunikaty muszą być pobierane z uchwytu obiektu zwróconego w wywołaniu subskrypcji. Na kliencie komunikaty są przechowywane w buforze komunikatów, dopóki nie zostaną wykorzystane przez klienta. Więcej informacji na ten temat zawiera sekcja MessageBuffer w pliku konfiguracyjnym klienta . Jeśli klient nie nadążyć za szybkością publikowania, komunikaty są usuwane zgodnie z wymaganiami, a najstarsze są usuwane jako pierwsze.

Zazwyczaj jest to decyzja administracyjna, czy aplikacja używa rozsyłania grupowego, czy nie, określona przez ustawienie atrybutu MCAST obiektu TOPIC. Jeśli aplikacja publikująca musi upewnić się, że rozsyłanie grupowe nie jest używane, może użyć opcji `MQOO_NO_MULTICAST` . Podobnie aplikacja subskrybująca może zapewnić, że nie jest używane rozsyłanie grupowe, subskrybując opcję `MQSO_NO_MULTICAST` .

Funkcja rozsyłania grupowego produktu IBM MQ obsługuje użycie selektorów komunikatów. Selektor jest używany przez aplikację do rejestrowania zainteresowania tylko tymi komunikatami, które mają właściwości zgodne z zapytaniem SQL92 reprezentowanym przez łańcuch wyboru. Więcej informacji na temat selektorów komunikatów zawiera sekcja “Selektory” na stronie 31.

W poniższej tabeli przedstawiono wszystkie główne koncepcje interfejsu MQI i sposób ich powiązania z rzutowaniem typu Multicast:

*Tabela 154. Pojęcia dotyczące interfejsu MQI i ich związek z rozsyłaniem grupowym*

| **Pojęcie MQI** | **Działanie przy próbie użycia rozsyłania grupowego** | **Kod przyczyny** |
|---|---|---|
| Umieszczanie komunikatu o zerowej długości | Odrzucone | 2005 (07D5) (RC2005): MQRC_BUFFER_LENGTH_ERROR |
| Grupowanie | Odrzucone | 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR |
| Segmentacja | Odrzucone | 2443 (098B) (RC2443): MQRC_SEGMENTATION_NOT_ALLOWED |
| Lista dystrybucyjna | Odrzucone | 2154 (086A) (RC2154): MQRC_RECS_PRESENT_ERROR |
| MQINQ | Odrzucone dla uchwytów tematów: MQINQ i MQSET tematów nie są obsługiwane. | 2038 (07F6) (RC2038): MQRC_NOT_OPEN_FOR_INQUIRE |
| MQINQ | Zaakceptowano dla zarządzanego uchwytu. Można uzyskać tylko zapytanie `Bieżąca głębokość` . | • Jeśli wartością jest Bieżąca głębokość, nie ma odpowiedniego kodu przyczyny.<br>• Jeśli wartość jest inna niż Bieżąca głębokość, kod przyczyny to 2067 (0813) (RC2067): MQRC_SELECTOR_ERROR. |
| MQSET | Odrzucono dla wszystkich uchwytów. | 2040 (07F8) (RC2040): MQRC_NOT_OPEN_FOR_SET |
| Transakcje (XA lub nie) | Odrzucone | 2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE |

*Tabela 154. Pojęcia dotyczące interfejsu MQI i ich związek z rozsyłaniem grupowym (kontynuacja)*

| **Pojęcie MQI** | **Działanie przy próbie użycia rozsyłania grupowego** | **Kod przyczyny** |
|---|---|---|
| Przeglądanie komunikatów | Odrzucone | 2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE |
| Zablokuj komunikaty | Odrzucone | 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR |
| Przeglądaj ze znacznikiem | Odrzucone | 2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE |
| Przekaż kontekst | Odrzucone | 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR |
| MQPUT1 | Odrzucono. Próba wykonania komendy MQPUT1 na temat tylko rozsyłania grupowego jest niepoprawna. | 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY |
| Subskrypcja trwała | Odrzucono, jeśli temat jest oznaczony jako Tylko rozsyłania grupowego, w przeciwnym razie zostanie wykonana subskrypcja inna niż rozsyłania grupowego. | 2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED |
| TopicString > 255 | Odrzucono. Jeśli łańcuch tematu jest dłuższy niż 255 znaków, jest on odrzucany przez klienta. | 2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR |
| Utworzono niezarządzaną subskrypcję | Odrzucono, jeśli temat jest oznaczony jako Tylko rozsyłania grupowego, w przeciwnym razie zostanie wykonana subskrypcja inna niż rozsyłania grupowego. | 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR |
| MQPMO_NOT_OWN_SUBS, | Odrzucone | 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR |

Następujące elementy rozszerzają niektóre pojęcia związane z MQI z poprzedniej tabeli i udostępniają informacje na temat niektórych pojęć związanych z MQI, które nie znajdują się w tabeli:

**Trwałość komunikatu**
W przypadku nietrwałych subskrybentów rozsyłania grupowego trwałe komunikaty z publikatora są dostarczane w sposób nienaprawialny.

**Obcinanie komunikatu**
Obsługiwane jest obcinanie komunikatów, co oznacza, że aplikacja może:

1. Wydaj komendę MQGET.
2. Pobranie MQRC_TRUNCATED_MSG_FAILED nie powiodło się.
3. Przydziel większy bufor.
4. Ponownie uruchom komendę MQGET, aby pobrać komunikat.

**Utrata ważności subskrypcji**
Utrata ważności subskrypcji nie jest obsługiwana. Każda próba ustawienia utraty ważności jest ignorowana.

## Wysoka dostępność dla rozsyłania grupowego

Te informacje umożliwiają zrozumienie działania produktu IBM MQ Multicast w trybie ciągłym komunikacji równorzędnej. Mimo że produkt IBM MQ nawiązuje połączenie z menedżerem kolejek produktu IBM MQ , komunikaty nie przepływają przez ten menedżer kolejek.

Mimo że połączenie z menedżerem kolejek musi zostać nawiązane w celu wywołania MQOPEN lub MQSUB obiektu tematu rozsyłania grupowego, same komunikaty nie przepływają przez menedżer kolejek. Dlatego po zakończeniu operacji MQOPEN lub MQSUB w obiekcie tematu rozsyłania można kontynuować przesyłanie komunikatów rozsyłania, nawet jeśli utracono połączenie z menedżerem kolejek. Istnieją dwa tryby działania:

**Nawiązywane jest zwykłe połączenie z menedżerem kolejek.**
Komunikacja rozsyłania grupowego jest możliwa, gdy istnieje połączenie z menedżerem kolejek. Jeśli połączenie nie powiedzie się, zostaną zastosowane normalne reguły MQI, na przykład operacja MQPUT do uchwytu obiektu rozsyłania zwraca wartość 2009 (07D9) (RC2009): MQRC_CONNECTION_BROKEN.

**Nawiązywane jest ponowne połączenie klienta z menedżerem kolejek.**
Komunikacja grupowa jest możliwa nawet podczas cyklu ponownego połączenia. Oznacza to, że nawet po zerwaniu połączenia z menedżerem kolejek nie ma to wpływu na umieszczanie i odbieranie komunikatów rozsyłania grupowego. Klient próbuje ponownie nawiązać połączenie z menedżerem kolejek, a jeśli ponowne połączenie nie powiedzie się, uchwyt połączenia zostanie zerwany i wszystkie wywołania MQI, w tym wywołania rozsyłania grupowego, zakończą się niepowodzeniem. Więcej informacji na ten temat zawiera sekcja Automatyczne ponowne nawiązywanie połączenia z klientem .

Jeśli dowolna aplikacja jawnie wyda komendę MQDISC, wszystkie subskrypcje rozsyłania grupowego i uchwyty obiektów zostaną zamknięte.

### Rozsyłanie ciągłe operacji typu każdy z każdym

Jedną z zalet komunikacji równorzędnej między klientami jest to, że komunikaty nie muszą przepływać przez menedżer kolejek. Oznacza to, że w przypadku zerwania połączenia z menedżerem kolejek przesyłanie komunikatów jest kontynuowane. W przypadku ciągłych wymagań dotyczących komunikatów w tym trybie obowiązują następujące ograniczenia:

- Połączenie musi zostać nawiązane przy użyciu jednej z opcji MQCNO_RECONNECT_ * dla operacji ciągłej. Ten proces oznacza, że chociaż sesja komunikacyjna może być przerwana, rzeczywisty uchwyt połączenia nie jest zerwany i jest w stanie ponownego połączenia. Jeśli ponowne połączenie nie powiedzie się, uchwyt połączenia zostanie zerwany, co uniemożliwi dalsze wywołania MQI.
- W tym trybie obsługiwane są tylko wartości MQPUT, MQGET, MQINQ i Konsumenty asynchroniczne. Wszystkie komendy MQOPEN, MQCLOSE lub MQDISC wymagają do zakończenia ponownego połączenia z menedżerem kolejek.

- Status przepływa do zatrzymanego menedżera kolejek. W związku z tym każdy stan w menedżerze kolejek może być nieaktualny lub nieaktualny. Oznacza to, że klienty mogą wysyłać i odbierać komunikaty, a status menedżera kolejek nie jest znany. Więcej informacji na ten temat zawiera sekcja Monitorowanie aplikacji rozsyłania grupowego .

## Konwersja danych w MQI na potrzeby przesyłania komunikatów w trybie rozsyłania grupowego

Te informacje umożliwiają zrozumienie sposobu działania konwersji danych na potrzeby przesyłania komunikatów w trybie rozsyłania grupowego produktu IBM MQ .

IBM MQ Multicast jest współużytkowanym, bezpołączeniowym protokołem, dlatego nie jest możliwe, aby każdy klient zgłaszał konkretne żądania konwersji danych. Każdy klient zasubskrybowany w tym samym strumieniu rozsyłania grupowego otrzymuje te same dane binarne, dlatego jeśli wymagana jest konwersja danych IBM MQ , konwersja jest wykonywana lokalnie na każdym kliencie.

Dane są konwertowane na kliencie dla ruchu rozsyłania grupowego IBM MQ . Jeśli podano opcję **MQGMO_CONVERT** , konwersja danych jest wykonywana zgodnie z żądaniem. Formaty zdefiniowane przez użytkownika wymagają zainstalowania wyjścia konwersji danych na kliencie. Informacje na temat bibliotek znajdujących się obecnie w pakietach klienta i serwera zawiera sekcja “Zapisywanie wyjść konwersji danych” na stronie 1013 .

Informacje na temat administrowania konwersją danych zawiera sekcja Włączanie konwersji danych na potrzeby przesyłania komunikatów w trybie rozsyłania grupowego.

Więcej informacji na temat konwersji danych zawiera sekcja Konwersja danych.

Więcej informacji na temat wyjść konwersji danych i `ClientExitPath`zawiera sekcja ClientExitPath w pliku konfiguracyjnym klienta.

## Raportowanie wyjątków rozsyłania grupowego

Use this information to learn about IBM MQ Multicast event handlers and reporting IBM MQ Multicast exceptions.

IBM MQ Rozsyłanie grupowe pomaga określić problem, wywołując procedurę obsługi zdarzeń w celu zgłoszenia zdarzeń rozsyłania grupowego, które są zgłaszane przy użyciu standardowego mechanizmu procedury obsługi zdarzeń systemu IBM MQ .

Pojedyncze zdarzenie rozsyłania grupowego może spowodować wywołanie więcej niż jednego zdarzenia IBM MQ , ponieważ może istnieć wiele uchwytów połączenia `MQHCONN` używających tego samego nadajnika lub odbiornika rozsyłania grupowego. Jednak każdy wyjątek rozsyłania grupowego powoduje wywołanie tylko jednej procedury obsługi zdarzeń dla jednego połączenia IBM MQ .

Stała IBM MQ `MQCBDO_EVENT_CALL` umożliwia aplikacjom rejestrowanie wywołań zwrotnych w celu odbierania tylko zdarzeń IBM MQ , a `MQCBDO_MC_EVENT_CALL` umożliwia aplikacjom rejestrowanie wywołań zwrotnych w celu odbierania tylko zdarzeń rozsyłania. Jeśli używane są obie stałe, odbierane są oba typy zdarzeń.

### Wysyłanie żądań zdarzeń rozsyłania grupowego

Zdarzenia rozsyłania grupowego IBM MQ używają stałej `MQCBDO_MC_EVENT_CALL` w polu `cbd.Options` . W poniższym przykładzie przedstawiono sposób żądania zdarzeń rozsyłania grupowego:

```
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);
```

Jeśli dla pola `cbd.Options` określono opcję `MQCBDO_MC_EVENT_CALL` , procedura obsługi zdarzeń jest wysyłana tylko do zdarzeń rozsyłania grupowego IBM MQ zamiast zdarzeń na poziomie połączenia. Aby zażądać, aby oba typy zdarzeń były wysyłane do procedury obsługi zdarzeń, aplikacja musi określić

stałą MQCBDO_EVENT_CALL w polu cbd.Options oraz stałą MQCBDO_MC_EVENT_CALL , jak pokazano w poniższym przykładzie:

```
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);
```

Jeśli żadna z tych stałych nie jest używana, do procedury obsługi zdarzeń wysyłane są tylko zdarzenia na poziomie połączenia.

Więcej informacji na temat wartości pola Options zawiera sekcja Opcje (MQLONG).

## Format zdarzenia rozsyłania grupowego

IBM MQ Wyjątki rozsyłania grupowego obejmują pewne informacje pomocnicze, które są zwracane w parametrze **Buffer** funkcji zwrotnej. Wskaźnik **Buffer** wskazuje na tablicę wskaźników, a pole MQCBC.DataLength określa wielkość tablicy w bajtach. Pierwszy element tablicy zawsze wskazuje krótki opis zdarzenia. W zależności od typu zdarzenia można podać więcej parametrów. Poniższa tabela zawiera listę wyjątków:

*Tabela 155. Opisy kodów zdarzeń rozsyłania grupowego*

| Kod zdarzenia | Opis | Dane dodatkowe |
|---|---|---|
| MQMCEV_PACKET_LOSS, | Nienaprawialna utrata pakietów | Liczba utraconych pakietów |
| MQMCEV_HEARTBEAT_TIMEOUT, | Długi brak pakietu sterującego pulsem | N/D |
| KONFLIKT WERSJI MQMCEV_VERSION_CONFLICT | Odbiór pakietów z nowszą wersją protokołu | N/D |
| NIEZAWODNOŚĆ MQMCEV_RELIABILITY | Różne tryby niezawodności nadajnika i odbiornika | N/D |
| MQMCEV_CLOSED_TRANS, | Transmisja tematu jest zamykana przez 1 źródło | N/D |
| BŁĄD MQMCEV_STREAM_ERROR | Wykryto błąd w strumieniu | N/D |
| MQMCEV_NEW_SOURCE | Nowe źródło rozpoczyna przesyłanie tematu | Struktura źródłowa |
| MQMCEV_RECEIVE_QUEUE_TRIMMED | Pakiety usunięte z PacketQ z powodu upływu czasu lub miejsca | Liczba obciętych pakietów |
| MQMCEV_PACKET_LOSS_NACK_EXPIRE | Nienaprawialna utrata pakietów z powodu utraty ważności NACK | Liczba utraconych pakietów |
| MQMCEV_ACK_RETRIES_EXCEEDED | Liczba pakietów usuniętych z historii po przekroczeniu **max_ack_retries** | Liczba usuniętych pakietów |
| MQMCEV_STREAM_SUSPEND_NACK | W strumieniu zaakceptowanym przez ten temat zostały zawieszone NACK | Zawieś identyfikator strumienia<br>Czas (w milisekundach), przez który strumień jest zawieszany |

*Tabela 155. Opisy kodów zdarzeń rozsyłania grupowego (kontynuacja)*

| Kod zdarzenia | Opis | Dane dodatkowe |
|---|---|---|
| MQMCEV_STREAM_RESUME_NACK | NACK zostały wznowione po zawieszeniu ich w strumieniu | Identyfikator strumienia |
| MQMCEV_STREAM_WYDALONY | Strumień zaakceptowany przez ten wątek został odrzucony z powodu żądania wydalenia | Identyfikator strumienia |
| PIERWSZY_KOMUNIKAT_MQMCEV_MESSAGE | Pierwszy komunikat ze źródła | Numer komunikatu |
| MQMCEV_LATE_JOIN_FAILURE | Nie powiodło się uruchomienie opóźnionej sesji łączenia | N/D |
| MQMCEV_MESSAGE_LOSS | Nienaprawialna utrata komunikatu | Liczba utraconych komunikatów |
| MQMCEV_SEND_PACKET_FAILURE | Wysłanie pakietu rozsyłania przez nadajnik rozsyłania nie powiodło się | N/D |
| MQMCEV_REPAIR_DELAY | Odbiornik rozsyłania nie otrzymał pakietu naprawczego dla zaległej NAK | N/D |
| MQMCEV_MEMORY_ALERT_ON | Bufory odbiorcze odbiornika zapełniają się | Procent wykorzystania puli buforów |
| MQMCEV_MEMORY_ALERT_OFF | Bufory odbiorcze odbiornika są wyłączone do normalnego | Procent wykorzystania puli buforów |
| MQMCEV_NACK_ALERT_ON | Szybkość żądań pakietów naprawy odbiornika osiągnęła wskaźnik wysokiego poziomu | Bieżąca szybkość żądań naprawy w pakietach na sekundę |
| MQMCEV_NACK_ALERT_OFF | Szybkość żądań pakietów naprawy odbiornika jest do wartości normalnej | Bieżąca szybkość żądań naprawy w pakietach na sekundę |
| MQMCEV_REPAIR_ALERT_ON | Osiągnięto wskaźnik wysokiego poziomu szybkości wysyłania pakietów naprawczych nadajnika | N/D |
| MQMCEV_REPAIR_ALERT_OFF | Szybkość wysyłania pakietów w naprawie nadajnika jest do wartości normalnej | N/D |
| MQMCEV_SHM_DEST_UNUSABLE, | Wykryto, że region pamięci współużytkowanej używany przez miejsce docelowe tematu nadajnika jest nie do użycia | N/D |
| MQMCEV_SHM_PORT_BEZUŻYTECZNY | Wykryto, że port pamięci współużytkowanej używany przez instancję odbiornika jest nie do użycia | N/D |
| Niepowodzenie MQMCEV_CCT_GETTIME_FAILED | Pobranie czasu z koordynowanego czasu klastra nie powiodło się | N/D |

| *Tabela 155. Opisy kodów zdarzeń rozsyłania grupowego (kontynuacja)* | | |
|---|---|---|
| **Kod zdarzenia** | **Opis** | **Dane dodatkowe** |
| MQMCEV_DEST_INTERFACE_FAILURE | Interfejs sieciowy używany przez miejsce docelowe tematu nadawcy uległ awarii, a zapasowy interfejs sieciowy jest niedostępny | |
| MQMCEV_DEST_INTERFACE_FAILOVER, | Działanie interfejsu sieciowego używanego przez miejsce docelowe tematu nadajnika zakończyło się niepowodzeniem i pomyślnie wykonano przełączenie awaryjne na inny interfejs. | |
| MQMCEV_PORT_INTERFACE-NIEPOWODZENIE | Interfejs sieciowy używany przez odbiornik rmmPort uległ awarii, a zapasowy interfejs sieciowy jest niedostępny (lub uległ awarii). | KonfiguracjaRMM |
| MQMCEV_PORT_INTERFACE_FAILOVER | Działanie interfejsu sieciowego używanego przez odbiornik rmmPort zakończyło się niepowodzeniem i pomyślnie wykonano przełączenie awaryjne na inny interfejs. | KonfiguracjaRMM |

# Kodowanie w języku C

Należy zwrócić uwagę na informacje w poniższych sekcjach dotyczące kodowania programów IBM MQ w języku C.


- "Parametry wywołań MQI" na stronie 1081
- "Parametry z niezdefiniowanym typem danych" na stronie 1082
- "Typy danych" na stronie 1082
- "Manipulowanie łańcuchami binarnymi" na stronie 1082
- "Manipulowanie łańcuchami znaków" na stronie 1082
- "Wartości początkowe dla struktur" na stronie 1083
- "Wartości początkowe dla struktur dynamicznych" na stronie 1083
- "Użyj z C++" na stronie 1084


## Parametry wywołań MQI

Parametry *tylko wejściowe* i typu MQHCONN, MQHOBJ, MQHMSG lub MQLONG są przekazywane przez wartość. W przypadku wszystkich innych parametrów *adres* parametru jest przekazywany przez wartość.

Nie wszystkie parametry przekazywane przez adres muszą być określane przy każdym wywołaniu funkcji. Jeśli konkretny parametr nie jest wymagany, wskaźnik pusty może być określony jako parametr w wywołaniu funkcji zamiast adresu danych parametru. Parametry, dla których jest to możliwe, są określone w opisach wywołań.

Żaden parametr nie jest zwracany jako wartość funkcji; w terminologii C oznacza to, że wszystkie funkcje zwracają wartość void.

Atrybuty funkcji są definiowane przez zmienną makra MQENTRY. Wartość tej zmiennej makra zależy od środowiska.

## Parametry z niezdefiniowanym typem danych

Funkcje MQGET, MQPUT i MQPUT1 mają parametr **`Buffer`** , który ma niezdefiniowany typ danych. Ten parametr służy do wysyłania i odbierania danych komunikatu aplikacji.

Parametry tego sortowania są wyświetlane w przykładach w języku C jako tablice MQBYTE. W ten sposób można zadeklarować parametry, ale zwykle wygodniej jest zadeklarować je jako strukturę opisującą układ danych w komunikacie. Parametr funkcji jest zadeklarowany jako wskaźnik do wartości typu void i dlatego adres wszystkich danych może być określony jako parametr w wywołaniu funkcji.

## Typy danych

Wszystkie typy danych są definiowane za pomocą instrukcji `typedef` .

Dla każdego typu danych zdefiniowany jest również odpowiedni typ danych wskaźnika. Nazwa typu danych wskaźnika jest nazwą typu danych elementarnych lub strukturalnych z przedrostkiem P oznaczającym wskaźnik. Atrybuty wskaźnika są definiowane przez zmienną makra MQPOINTER. Wartość tej zmiennej makra zależy od środowiska. Poniższy kod ilustruje sposób deklarowania typów danych wskaźników:

```
#define MQPOINTER                   /* depends on environment */
...
typedef MQLONG  MQPOINTER PMQLONG;  /* pointer to MQLONG */
typedef MQMD   MQPOINTER PMQMD;     /* pointer to MQMD */
```

## Manipulowanie łańcuchami binarnymi

Łańcuchy danych binarnych są deklarowane jako jeden z typów danych MQBYTEn.

Podczas kopiowania, porównywania lub ustawiania pól tego typu należy używać funkcji języka C `memcpy`, `memcmp`lub `memset`:

```
#include <string.h>
#include "cmqc.h"

MQMD MyMsgDesc;

memcpy(MyMsgDesc.MsgId,           /* set "MsgId" field to nulls    */
       MQMI_NONE,                 /* ...using named constant       */
       sizeof(MyMsgDesc.MsgId));

memset(MyMsgDesc.CorrelId,        /* set "CorrelId" field to nulls */
       0x00,                      /* ...using a different method   */
       sizeof(MQBYTE24));
```

Nie należy używać funkcji łańcuchowych strcpy, strcmp, strncpy lub strncmp, ponieważ nie działają one poprawnie z danymi zadeklarowanymi jako MQBYTE24.

## Manipulowanie łańcuchami znaków

Gdy menedżer kolejek zwraca dane znakowe do aplikacji, dane znakowe są zawsze dopełniane odstępami do zdefiniowanej długości pola. Menedżer kolejek nie zwraca łańcuchów zakończonych znakiem o kodzie zero, ale można ich użyć w danych wejściowych. Dlatego podczas kopiowania, porównywania lub konkatenowania takich łańcuchów należy użyć funkcji łańcuchowych strncpy, strncmp lub strncat.

Nie należy używać funkcji łańcuchowych, które wymagają, aby łańcuch był zakończony wartością null (strcpy, strcmp i strcat). Ponadto nie należy używać funkcji strlen do określenia długości łańcucha; zamiast niej należy użyć funkcji sizeof do określenia długości pola.

## Wartości początkowe dla struktur

Plik włączany <cmqc.h> definiuje różne zmienne makra, których można użyć w celu udostępnienia wartości początkowych dla struktur podczas deklarowania instancji tych struktur. Te zmienne makra mają nazwy w postaci MQxxx_DEFAULT, gdzie MQxxx reprezentuje nazwę struktury. Użyj ich w następujący sposób:

```
MQMD   MyMsgDesc = {MQMD_DEFAULT};
MQPMO  MyPutOpts = {MQPMO_DEFAULT};
```

W przypadku niektórych pól znakowych interfejs MQI definiuje konkretne wartości, które są poprawne (na przykład dla pól *StrucId* lub dla pola *Format* w strukturze MQMD). Dla każdej z poprawnych wartości podano dwie zmienne makra:

- Jedna zmienna makra definiuje wartość jako łańcuch o długości, z wyłączeniem domniemanej wartości NULL, która dokładnie odpowiada zdefiniowanej długości pola. W poniższych przykładach symbol ¬ reprezentuje pojedynczy znak odstępu:

  ```
  #define MQMD_STRUC_ID "MD¬¬"
  #define MQFMT_STRING "MQSTR¬¬¬"
  ```

  Użyj tego formularza z funkcjami memcpy i memcmp.
- Inna zmienna makra definiuje wartość jako tablicę znaków; nazwa tej zmiennej makra jest nazwą łańcucha z przyrostkiem _ARRAY. Na przykład:

  ```
  #define MQMD_STRUC_ID_ARRAY 'M','D','¬','¬'
  #define MQFMT_STRING_ARRAY 'M','Q','S','T','R','¬','¬','¬'
  ```

  Ten formularz służy do inicjowania pola, gdy instancja struktury jest zadeklarowana z wartościami innymi niż wartości udostępnione przez zmienną makra MQMD_DEFAULT.

## Wartości początkowe dla struktur dynamicznych

Jeśli wymagana jest zmienna liczba instancji struktury, instancje są zwykle tworzone w pamięci głównej uzyskanej dynamicznie przy użyciu funkcji calloc lub malloc.

Aby zainicjować pola w takich strukturach, zaleca się zastosowanie następującej techniki:

1. Zadeklaruj instancję struktury za pomocą odpowiedniej zmiennej makra MQxxx_DEFAULT w celu zainicjowania struktury. Ta instancja staje się *modelem* dla innych instancji:

   ```
   MQMD ModelMsgDesc = {MQMD_DEFAULT};
                                     /* declare model instance */
   ```

   Zakoduj statyczne lub automatyczne słowa kluczowe w deklaracji, aby nadać instancji modelu statyczny lub dynamiczny czas życia (zgodnie z wymaganiami).
2. Użyj funkcji calloc lub malloc, aby uzyskać pamięć dla dynamicznej instancji struktury:

   ```
   PMQMD InstancePtr;
   InstancePtr = malloc(sizeof(MQMD));
                                     /* get storage for dynamic instance */
   ```
3. Użyj funkcji memcpy, aby skopiować instancję modelu do instancji dynamicznej:

   ```
   memcpy(InstancePtr,&ModelMsgDesc,sizeof(MQMD));
                                     /* initialize dynamic instance */
   ```

### Użyj z C++

W przypadku języka programowania C++ pliki nagłówkowe zawierają następujące dodatkowe instrukcje, które są dołączane tylko wtedy, gdy używany jest kompilator C + +:

```
#ifdef __cplusplus
  extern "C" {
#endif

/* rest of header file */

#ifdef __cplusplus
  }
#endif
```

## Windows Kodowanie w języku Visual Basic

Informacje, które należy wziąć pod uwagę podczas kodowania programów IBM MQ w języku Microsoft Visual Basic. Parametr Visual Basic jest obsługiwany tylko w systemie Windows.

**Uwaga:**

Stabilized Od wersji IBM WebSphere MQ 7.0, poza środowiskiem .NET , obsługa produktu Visual Basic (VB) została ustabilizowana na poziomie IBM WebSphere MQ 6.0 . Większość nowych funkcji dodanych do produktu IBM WebSphere MQ 7.0 lub nowszego nie jest dostępna dla aplikacji VB. W przypadku programowania w języku VB.NETnależy użyć klas IBM MQ dla .NET. Więcej informacji na ten temat zawiera artykuł Tworzenie aplikacji .NET.

Deprecated Począwszy od wersji IBM MQ 9.0, obsługa języka Microsoft Visual Basic 6.0 jest nieaktualna. Zalecaną technologią zastępczą są klasy IBM MQ dla .NET.

Aby uniknąć niezamierzonego tłumaczenia danych binarnych przekazywanych między systemami Visual Basic i IBM MQ, należy użyć definicji MQBYTE zamiast MQSTRING. CMQB.BAS definiuje kilka nowych typów MQBYTE, które są równoważne definicji bajtu C i używają ich w strukturach IBM MQ . Na przykład w przypadku struktury MQMD (deskryptor komunikatu) wartość MsgId (identyfikator komunikatu) jest definiowana jako MQBYTE24.

Visual Basic nie ma typu danych wskaźnikowego, dlatego odwołania do innych struktur danych IBM MQ mają przesunięcie, a nie wskaźnik. Zadeklaruj strukturę złożoną składającą się z dwóch struktur składowych i określ strukturę złożoną w wywołaniu. IBM MQ Obsługa języka Visual Basic udostępnia wywołanie MQCONNXAny w celu umożliwienia i umożliwienia aplikacjom klienckim określania właściwości kanału w połączeniu klienckim. Akceptuje on strukturę beztypową (MQCNOCD) zamiast typowej struktury MQCNO.

Struktura MQCNOCD jest strukturą złożoną składającą się z obiektu MQCNO, po którym następuje obiekt MQCD. Ta struktura jest zadeklarowana w pliku nagłówkowym wyjść CMQXB. Użyj procedury MQCNOCD_DEFAULTS do zainicjowania struktury MQCNOCD. Udostępniono przykład tworzenia wywołań MQCONNX (amqscnxb.vbp).

Parametr MQCONNXAny ma takie same parametry jak parametr MQCONNX, z tą różnicą, że parametr **ConnectOpts** jest zadeklarowany jako dowolny typ danych, a nie jako typ danych MQCNO. Umożliwia to funkcji akceptowanie struktury MQCNO lub MQCNOCD. Ta funkcja jest zadeklarowana w głównym pliku nagłówkowym CMQB.

**Pojęcia pokrewne**

"Przygotowywanie programów Visual Basic w systemie Windows" na stronie 1050
Informacje, które należy wziąć pod uwagę podczas używania programów Microsoft Visual Basic w systemie Windows.

**Odsyłacze pokrewne**

"Łączenie aplikacji Visual Basic z kodem IBM MQ MQI client" na stronie 949
Aplikacje Microsoft Visual Basic można połączyć za pomocą kodu IBM MQ MQI client w systemie Windows.

# Kodowanie w języku COBOL

Podczas kodowania programów IBM MQ w języku COBOL należy zapoznać się z informacjami zawartymi w poniższej sekcji.

## Stałe nazwane

W nazwie wyświetlane są nazwy stałych zawierające znak podkreślenia (_). W języku COBOL zamiast znaku podkreślenia należy użyć znaku łącznika (-). Stałe, które mają wartości łańcucha znaków, używają znaku pojedynczego cudzysłowu (') jako separatora łańcucha. Aby kompilator akceptował ten znak, należy użyć opcji APOST kompilatora.

Plik kopii CMQV zawiera deklaracje stałych nazwanych jako elementy level-10 . Aby użyć stałych, zadeklaruj element level-01 jawnie, a następnie użyj instrukcji COPY do skopiowania deklaracji stałych:

```
WORKING-STORAGE SECTION.
01 MQM-CONSTANTS.
COPY CMQV.
```

Jednak ta metoda powoduje, że stałe zajmują pamięć w programie, nawet jeśli nie są przywoływane. Jeśli stałe znajdują się w wielu oddzielnych programach w tej samej jednostce uruchamiania, istnieje wiele kopii stałych; może to spowodować użycie znacznej ilości pamięci głównej. Można tego uniknąć, dodając klauzulę GLOBAL do deklaracji level-01 :

```
* Declare a global structure to hold the constants
01 MQM-CONSTANTS GLOBAL.
COPY CMQV.
```

Spowoduje to przydzielenie pamięci tylko dla *jednego* zestawu stałych w jednostce uruchamiania. Stałe mogą być jednak przywołane przez *dowolny* program w jednostce uruchamiania, a nie tylko przez program, który zawiera deklarację level-01 .

## Zapewnianie wyrównania struktury

Należy zadbać o to, aby struktury IBM MQ przekazywane w celu rozpoczęcia wywołania produktu MQ były wyrównane względem granic słów. Granica słowa wynosi 4 bajty dla procesów 32-bitowych, 8 bajtów dla procesów 64-bitowych i 16 bajtów dla procesów 128-bitowych ( IBM i ).

Tam, gdzie to możliwe, wszystkie struktury IBM MQ należy umieścić razem, tak aby wszystkie były wyrównane do granicy.

# Kodowanie w języku asemblera systemu System/390 (interfejs kolejki komunikatów)

Podczas kodowania programów IBM MQ for z/OS w języku asembler należy zapoznać się z informacjami zawartymi w poniższych sekcjach.

## Nazwy

Nazwy parametrów w opisach wywołań i nazwy pól w opisach struktur są wyświetlane z mieszaną wielkością liter. W makrach języka asemblera dostarczanych z produktem IBM MQwszystkie nazwy są pisane wielkimi literami.

## Korzystanie z wywołań MQI

Interfejs MQI jest interfejsem wywołania, dlatego programy w języku asemblera muszą przestrzegać konwencji łączenia systemu operacyjnego.

W szczególności, przed wywołaniem MQI, programy w języku asembler muszą zarejestrować R13 w obszarze zapisu co najmniej 18 pełnych słów. Ten obszar składowania udostępnia pamięć dla wywoływanego programu. Zapisuje on rejestry programu wywołującego przed zniszczeniem jego treści i odtwarza zawartość rejestrów programu wywołującego po powrocie.

**Uwaga:** Jest to ważne dla programów w języku asembler systemu CICS , które używają makra DFHEIENT do skonfigurowania swojej dynamicznej pamięci masowej, ale które przesłaniają domyślną wartość DATAREG z R13 do innych rejestrów. Gdy interfejs CICS Resource Manager odbierze sterowanie z kodu pośredniczącego, zapisuje bieżącą treść rejestrów pod adresem, który wskazuje produkt R13 . Niezarezerwowanie obszaru zapisu w tym celu daje nieprzewidywalne wyniki i prawdopodobnie spowoduje nieprawidłowe zakończenie w programie CICS.

## Deklarowanie stałych

Większość stałych jest deklarowanych jako równe w makrze CMQA.

Jednak następujące stałe nie mogą być definiowane jako równoważne i nie są one uwzględniane podczas wywoływania makra przy użyciu opcji domyślnych:

- MQACT_NONE
- MQCI_BRAK
- MQFMT_BRAK
- MQFMT_ADMIN,
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- MQFMT_EVENT, ZDARZENIE
- MQFMT_IMS,
- MQFMT_IMS_VAR_STRING (łańcuch zmiennych)
- MQFMT_PCF,
- MQFMT_STRING
- MQFMT_TRIGGER,
- MQFMT_XMIT_Q_HEADER
- MQMI_BRAK

Aby je uwzględnić, podczas wywoływania makra należy dodać słowo kluczowe EQUONLY=NO.

CMQA jest chroniona przed wieloma deklaracją, więc można ją dołączyć wiele razy. Jednak słowo kluczowe EQUONLY odnosi skutek tylko przy pierwszym dołączonym makrze.

## Określanie nazwy struktury

Aby umożliwić zadeklarowanie więcej niż jednej instancji struktury, makro generujące strukturę poprzedza nazwę każdego pola łańcuchem, który może być podany przez użytkownika, i znakiem podkreślenia (_).

Określ łańcuch podczas wywoływania makra. Jeśli łańcuch nie zostanie określony, makro użyje nazwy struktury do skonstruowania przedrostka:

```
* Declare two object descriptors
CMQODA         Prefix used="MQOD_" (the default)
MY_MQOD CMQODA         Prefix used="MY_MQOD_"
```

Deklaracje struktur w sekcji Opisy wywołań zawierają przedrostek domyślny.

## Określanie formy struktury

Makra mogą generować deklaracje struktury w jednej z dwóch form, sterowanych przez parametr DSECT:

**DSECT = TAK**
Instrukcja DSECT języka asemblera jest używana do uruchamiania nowej sekcji danych; definicja struktury następuje bezpośrednio po instrukcji DSECT. Nie przydzielono pamięci, więc inicjowanie nie jest możliwe. Etykieta w wywołaniu makra jest używana jako nazwa sekcji danych; jeśli nie określono etykiety, używana jest nazwa struktury.

**DSECT = NIE**
Instrukcje DC języka asemblera są używane do definiowania struktury w bieżącej pozycji w procedurze. Pola są inicjowane wartościami, które można określić, kodując odpowiednie parametry w wywołaniu makra. Pola, dla których nie określono wartości w wywołaniu makra, są inicjowane wartościami domyślnymi.

Jeśli nie określono parametru DSECT, przyjmowany jest parametr DSECT = NO.

## Sterowanie listingiem

Za pomocą parametru LIST można sterować wyglądem deklaracji struktury w listingu języka asemblera:

**LIST = TAK**
Deklaracja struktury jest wyświetlana na liście języka asemblera.

**LISTA = NIE**
Deklaracja struktury nie pojawia się na liście języka asemblera. Przyjmuje się, że parametr LIST nie jest określony.

## Określanie wartości początkowych dla zmiennych

Wartość, która ma być używana do inicjowania pola w strukturze, można określić, kodując nazwę tego pola (bez przedrostka) jako parametr w wywołaniu makra, wraz z wymaganą wartością.

Na przykład, aby zadeklarować strukturę deskryptora komunikatu z polem *MsgType* zainicjowanym za pomocą MQMT_REQUEST i polem *ReplyToQ* zainicjowanym za pomocą łańcucha MY_REPLY_TO_QUEUE, należy użyć następującego kodu:

```
MY_MQMD    CMQMDA      MSGTYPE=MQMT_REQUEST,       X
REPLYTOQ=MY_REPLY_TO_QUEUE
```

W przypadku określenia stałej nazwanej (lub równości) jako wartości w wywołaniu makra należy użyć makra CMQA w celu zdefiniowania stałej nazwanej. Nie można ujmować w apostrofy ('') wartości, które są łańcuchami znaków.

## Pisanie programów, które można reenterable

Produkt IBM MQ używa swoich struktur zarówno dla wejścia, jak i wyjścia. Jeśli program ma być ponownie wprowadzalny:

1. Zdefiniuj wersje robocze pamięci masowej struktur jako DSECT lub zdefiniuj struktury wstawiane w już zdefiniowanym DSECT. Następnie skopiuj DSECT do pamięci masowej, która jest uzyskiwana za pomocą:

- W przypadku programów wsadowych i TSO-makra asemblera STORAGE lub GETMAIN z/OS
- W systemie CICS: komenda DSECT pamięci roboczej (DFHEISTG) lub EXEC CICS GETMAIN

Aby poprawnie zainicjować te struktury pamięci roboczej, należy skopiować stałą wersję odpowiedniej struktury do wersji pamięci roboczej.

**Uwaga:** Struktury MQMD i MQXQH mają więcej niż 256 bajtów długości. Aby skopiować te struktury do pamięci masowej, należy użyć instrukcji asemblera MVCL.

2. Zarezerwuj miejsce w pamięci, korzystając z formularza LIST ( MF=L) makra CALL. Jeśli do wywołania MQI używane jest makro CALL, należy użyć formularza EXECUTE ( MF=E) makra, używając wcześniej zarezerwowanej pamięci masowej, jak to pokazano w przykładzie w sekcji "Korzystanie z funkcji CEDF" na stronie 1088. Więcej przykładów na ten temat można znaleźć w przykładowych programach w języku asembler dostarczonych z produktem IBM MQ.

Użyj opcji RENT języka asemblera, aby określić, czy program może być ponownie wprowadzany.

Informacje na temat pisania programów z możliwością ponownego wpisywania zawiera publikacja *z/OS MVS Application Development Guide: Assembler Language Programs*.

### Korzystanie z funkcji CEDF

Aby użyć transakcji dostarczonej przez CICS, CEDF ( CICS Execution Diagnostic Facility) w celu ułatwienia debugowania programu, należy dodać słowo kluczowe ,VL do każdej instrukcji CALL , na przykład:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

Poprzednim przykładem jest kod języka asemblera, w którym PARMAREA jest obszarem w podanej pamięci roboczej.

## Korzystanie z wywołań MQI

Interfejs MQI jest interfejsem wywołania, dlatego programy w języku asemblera muszą przestrzegać konwencji łączenia systemu operacyjnego. W szczególności, przed wywołaniem MQI, programy w języku asembler muszą zarejestrować R13 w obszarze zapisu co najmniej 18 pełnych słów. Ten obszar składowania udostępnia pamięć dla wywoływanego programu. Zapisuje on rejestry programu wywołującego przed zniszczeniem jego treści i odtwarza zawartość rejestrów programu wywołującego po powrocie.

**Uwaga:** Jest to ważne dla programów w języku asembler systemu CICS , które używają makra DFHEIENT do skonfigurowania swojej dynamicznej pamięci masowej, ale które przesłaniają domyślną wartość DATAREG z R13 do innych rejestrów. Gdy interfejs CICS Resource Manager odbierze sterowanie z kodu pośredniczącego, zapisuje bieżącą treść rejestrów pod adresem, który wskazuje produkt R13 . Niezarezerwowanie odpowiedniego obszaru zapisu w tym celu daje nieprzewidywalne rezultaty i prawdopodobnie spowoduje nieprawidłowe zakończenie w programie CICS.

# IBM i Kodowanie programów IBM MQ w języku RPG (tylko w systemieIBM i )

W dokumentacji IBM MQ wszystkie parametry wywołań, nazwy typów danych, pola struktur i nazwy stałych są opisane przy użyciu ich długich nazw. W języku RPG nazwy te są skrócone do sześciu lub mniejszej liczby wielkich liter.

Na przykład w języku RPG pole *MsgType* będzie mieć postać *MDMT* . Więcej informacji na ten temat zawiera podręcznik IBM i Application Programming Reference (ILE/RPG).

# Kodowanie w języku PL/I (tylko w systemiez/OS )

Przydatne informacje podczas kodowania dla IBM MQ w PL/I.

### Struktury

Struktury są deklarowane z atrybutem BASED i dlatego nie zajmują pamięci, chyba że program deklaruje jedną lub więcej instancji struktury.

Instancję struktury można zadeklarować za pomocą atrybutu `like` , na przykład:

```
dcl my_mqmd    like MQMD; /* one instance */
dcl my_other_mqmd like MQMD; /* another one */
```

Pola struktury są deklarowane z atrybutem INITIAL; gdy atrybut `like` jest używany do deklarowania instancji struktury, ta instancja dziedziczy wartości początkowe zdefiniowane dla tej struktury. Należy ustawić tylko te pola, w których wymagana wartość różni się od wartości początkowej.

W języku PL/I nie jest rozróżniana wielkość liter, dlatego nazwy wywołań, pól struktury i stałych mogą być kodowane małymi literami, wielkimi literami lub literami o różnej wielkości.

### Stałe nazwane

Stałe nazwane są deklarowane jako zmienne makra; w rezultacie stałe nazwane, do których program nie odwołuje się, nie zajmują pamięci w skompilowanej procedurze.

Jednak opcja kompilatora, która powoduje, że kod źródłowy jest przetwarzany przez preprocesor makr, musi być określona podczas kompilowania programu.

Wszystkie zmienne makra są zmiennymi znakowymi, nawet tymi, które reprezentują wartości liczbowe. Chociaż może to wydawać się sprzeczne z intuicją, nie powoduje to konfliktu typów danych po podstawieniu zmiennych makra przez procesor makr, na przykład:

```
%dcl MQMD_STRUC_ID char;
%MQMD_STRUC_ID = '''MD ''';

%dcl MQMD_VERSION_1 char;
%MQMD_VERSION_1 = '1';
```

## Korzystanie z przykładowych programów proceduralnych IBM MQ

Te przykładowe programy zostały napisane w językach proceduralnych i przedstawiają typowe zastosowania interfejsu kolejek komunikatów (Message Queue Interface-MQI). Programy IBM MQ na różnych platformach.

### O tym zadaniu

Istnieją dwa zestawy próbek:

- Przykładowe programy dla systemów rozproszonych i systemu IBM i.
- Przykładowe programy dla z/OS.

### Procedura

- Aby uzyskać więcej informacji na temat programów przykładowych, skorzystaj z następujących odsyłaczy:
  - “Korzystanie z przykładowych programów w systemie Multiplatforms” na stronie 1090
  - z/OS “Korzystanie z przykładowych programów w systemie z/OS” na stronie 1194

**Pojęcia pokrewne**

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć języka proceduralnego lub obiektowego. Przed rozpoczęciem projektowania i pisania aplikacji IBM MQ należy zapoznać się z podstawowymi pojęciami związanymi z produktem IBM MQ .

“Tworzenie aplikacji dla składnika IBM MQ” na stronie 5
Użytkownik może tworzyć aplikacje do wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i zasobami pokrewnymi. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i środowiskach.

“Uwagi dotyczące projektowania dla aplikacji IBM MQ” na stronie 51
Po podjęciu decyzji, w jaki sposób aplikacje mogą korzystać z dostępnych platform i środowisk, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

“Pisanie aplikacji proceduralnej dotyczącej kolejkowania” na stronie 744
Ten temat zawiera informacje o pisaniu aplikacji kolejkowania, nawiązywaniu i rozłączaniu połączeń z menedżerem kolejek, publikowaniu i subskrybowaniu oraz otwieraniu i zamykaniu obiektów.

“Pisanie aplikacji proceduralnych klienta” na stronie 941
Informacje potrzebne do pisania aplikacji klienckich w systemie IBM MQ przy użyciu języka proceduralnego.

“Pisanie aplikacji publikowania/subskrybowania” na stronie 834
Rozpocznij pisanie aplikacji IBM MQ publikowania/subskrypcji.

“Budowanie aplikacji proceduralnej” na stronie 1029
Użytkownik może napisać aplikację IBM MQ w jednym z kilku języków proceduralnych i uruchomić ją na kilku różnych platformach.

“Obsługa błędów proceduralnych programu” na stronie 1069
Te informacje wyjaśniają błędy powiązane z wywołaniami MQI aplikacji podczas wykonywania wywołania lub po dostarczeniu komunikatu do miejsca docelowego.

## Multi Korzystanie z przykładowych programów w systemie Multiplatforms

Te przykładowe programy proceduralne są dostarczane z produktem. Przykłady zostały napisane w języku C i COBOL i przedstawiają typowe zastosowania interfejsu kolejek komunikatów (Message Queue Interface-MQI).

### O tym zadaniu

Przykłady nie mają na celu zademonstrowania ogólnych technik programowania, dlatego pewne sprawdzanie błędów, które można włączyć do programu produkcyjnego, jest pomijane.

Kod źródłowy wszystkich przykładów jest dostarczany wraz z produktem. Źródło to zawiera komentarze, które wyjaśniają techniki kolejkowania komunikatów przedstawione w programach.

IBM i Informacje na temat programowania w języku RPG zawiera publikacja IBM i Application Programming Reference (ILE/RPG).

Nazwy przykładów zaczynają się od przedrostka `amq`. Czwarty znak wskazuje język programowania, a w razie potrzeby kompilator:

- s: język C
- 0: język COBOL w kompilatorach IBM i Micro Focus
- i: tylko kompilatory języka COBOL w systemie IBM
- m: Język COBOL tylko dla kompilatorów Micro Focus

Ósmy znak kodu wykonywalnego wskazuje, czy przykład działa w lokalnym trybie powiązania, czy w trybie klienta. Jeśli nie ma ósmego znaku, przykład jest uruchamiany w trybie powiązań lokalnych. Jeśli ósmy znak to 'c', przykład działa w trybie klienta.

Przed uruchomieniem przykładowych aplikacji należy najpierw utworzyć i skonfigurować menedżer kolejek. Informacje na temat konfigurowania menedżera kolejek do akceptowania połączeń klientów zawiera sekcja “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym” na stronie 1101.

## Procedura

- Aby uzyskać więcej informacji na temat programów przykładowych, skorzystaj z następujących odsyłaczy:
  - “Funkcje demonstrowane w przykładowych programach w środowisku wieloplatformowym” na stronie 1092
  - “Przygotowywanie i uruchamianie programów przykładowych” na stronie 1100
  - “Przykładowy program obsługi wyjścia funkcji API” na stronie 1107
  - “Przykładowy program wykorzystania asynchronicznego” na stronie 1108
  - “Przykładowy program asynchronicznego umieszczania” na stronie 1109
  - “Przykładowe programy do przeglądania” na stronie 1110
  - “Przykładowy program przeglądarki” na stronie 1111
  - “Przykład transakcji CICS” na stronie 1112
  - “Przykładowy program Connect” na stronie 1113
  - “Przykładowy program do konwersji danych” na stronie 1114
  - “Przykłady koordynacji bazy danych” na stronie 1114
  - “Przykład procedury obsługi kolejki niedostarczonych komunikatów” na stronie 1121
  - “Przykładowy program listy dystrybucyjnej” na stronie 1121
  - “Przykładowe programy Echo” na stronie 1122
  - “Przykładowe programy Get” na stronie 1123
  - “Przykładowe programy o wysokiej dostępności” na stronie 1125
  - “Przykładowe programy do sprawdzania” na stronie 1129
  - “Właściwości zapytania programu przykładowego uchwytu komunikatu” na stronie 1130
  - “Przykładowe programy publikowania/subskrypcji” na stronie 1130
  - “Przykładowy program wyjścia publikowania” na stronie 1134
  - “Przykładowe programy Put” na stronie 1135
  - “Przykładowe programy komunikatów referencyjnych” na stronie 1137
  - “Przykładowe programy żądania” na stronie 1145
  - “Przykładowe programy Set” na stronie 1151
  - “Przykładowy program TLS” na stronie 1152
  - “Przykładowe programy wyzwalające” na stronie 1155
  - “Korzystanie z przykładów TUXEDO w systemie AIX, Linux, and Windows” na stronie 1157
  - “Korzystanie z wyjścia zabezpieczeń SSPI w systemie Windows” na stronie 1167
  - “Uruchamianie przykładów przy użyciu kolejek zdalnych” na stronie 1168
  - “Przykładowy program monitorowania kolejki klastra (AMQSCLM)” na stronie 1168
  - “Przykładowy program do wyszukiwania punktów końcowych połączenia (Connection Endpoint Lookup-CEPL)” na stronie 1178

**Pojęcia pokrewne**

“Przykładowe programy w języku C++” na stronie 548
Dostarczane są cztery przykładowe programy demonstrujące pobieranie i umieszczanie komunikatów.

**Zadania pokrewne**

“Korzystanie z przykładowych programów w systemie z/OS” na stronie 1194
Przykładowe aplikacje proceduralne dostarczane z produktem IBM MQ for z/OS demonstrują typowe zastosowania interfejsu MQI (Message Queue Interface).

### Multi *Funkcje demonstrowane w przykładowych programach w środowisku wieloplatformowym*

Kolekcja tabel przedstawiających techniki demonstrowane przez programy przykładowe IBM MQ .

Wszystkie przykłady otwierają i zamykają kolejki przy użyciu wywołań MQOPEN i MQCLOSE, dlatego te techniki nie są wymienione oddzielnie w tabelach. Należy zapoznać się z nagłówkiem zawierającym interesującą platformę.

z/OS W przypadku platformy z/OS należy zapoznać się z sekcją "Korzystanie z przykładowych programów w systemie z/OS" na stronie 1194.

Linux AIX *Przykłady dla systemów AIX and Linux*

Techniki demonstrowane przez programy przykładowe dla produktu IBM MQ for AIX or Linux.

Aby dowiedzieć się, gdzie są przechowywane programy przykładowe dla systemu IBM MQ for AIX or Linux , należy zapoznać się z sekcją "Przygotowywanie i uruchamianie programów przykładowych w systemie AIX and Linux" na stronie 1104 .

Tabela 156 na stronie 1092 Tabela zawiera listę udostępnionych plików źródłowych w językach C i COBOL oraz informacje o tym, czy plik wykonywalny serwera lub klienta jest dołączony.

*Tabela 156. Przykładowe programy demonstrujące użycie interfejsu MQI (C i COBOL) w systemie AIX and Linux.*

Tabela z czterema kolumnami. W pierwszych kolumnach wymieniono techniki wykazane przez próbki. Druga kolumna zawiera listę przykładów w języku C, a trzecia-przykłady w języku COBOL, które demonstrują każdą z technik wymienionych w pierwszej kolumnie. Czwarta kolumna pokazuje, czy plik wykonywalny serwera C jest, czy nie jest, oraz piąta kolumna pokazuje, czy plik wykonywalny klienta C jest, czy nie jest.

| **Technika** | **C (źródło) ("1" na stronie 1094)** | **COBOL (źródło) ("2" na stronie 1094)** | **Serwer (kod wykonywalny C)** | **Klient (kod wykonywalny C)** |
|---|---|---|---|---|
| Korzystanie z interfejsu publikowania/ subskrypcji | amqspuba<br>amqssuba<br>amqssbxa | brak próbki | amqspub<br>amqssub<br>amqssbx | brak próbki |
| Umieszczanie komunikatów przy użyciu wywołania MQPUT | amqsput0 | amq0put0 | amqsput | amqsputc |
| Umieszczanie pojedynczego komunikatu przy użyciu wywołania MQPUT1 | amqsinqa<br>amqsecha | amqminqx<br>amqmechx<br>amqiinqx<br>amqiechx | amqsinq<br>amqsech | amqsechc |
| Umieszczanie komunikatów na liście dystrybucyjnej ("3" na stronie 1094) | amqsptl0 | amq0ptl0.cbl | amqsptl, | amqsptlc |
| Odpowiadanie na komunikat żądania | amqsinqa | amqminqx<br>amqiinqx | amqsinq | brak próbki |
| Pobieranie komunikatów za pomocą przeglądania (bez oczekiwania) | amqsgbr0 | amq0gbr0 | amqsgbr | brak próbki |
| Pobieranie komunikatów (oczekiwanie z limitem czasu) | amqsget0 | amq0get0 | amqsget | amqsgetc |
| Pobieranie komunikatów (nieograniczony czas oczekiwania) | amqstrg0 | brak próbki | amqstrg | amqstrgc |
| Pobieranie komunikatów (z konwersją danych) | amqsecha | brak próbki | amqsech | brak próbki |

*Tabela 156. Przykładowe programy demonstrujące użycie interfejsu MQI (C i COBOL) w systemie AIX and Linux.*

Tabela z czterema kolumnami. W pierwszych kolumnach wymieniono techniki wykazane przez próbki. Druga kolumna zawiera listę przykładów w języku C, a trzecia-przykłady w języku COBOL, które demonstrują każdą z technik wymienionych w pierwszej kolumnie. Czwarta kolumna pokazuje, czy plik wykonywalny serwera C jest, czy nie jest, oraz piąta kolumna pokazuje, czy plik wykonywalny klienta C jest, czy nie jest.

*(kontynuacja)*

| **Technika** | **C (źródło) (“1” na stronie 1094)** | **COBOL (źródło) (“2” na stronie 1094)** | **Serwer (kod wykonywalny C)** | **Klient (kod wykonywalny C)** |
|---|---|---|---|---|
| Umieszczanie komunikatów referencyjnych w kolejce (“3” na stronie 1094) | amqsprma | brak próbki | amqsprm, | amqsprmc |
| Pobieranie komunikatów referencyjnych z kolejki (“3” na stronie 1094) | amqsgrma | brak próbki | amqsgrm | amqsgrmc |
| Odwołanie do wyjścia kanału komunikatów (“3” na stronie 1094) | amqsqrma amqsxrma | brak próbki | amqsxrm, | brak próbki |
| Przeglądanie pierwszych 20 znaków wiadomości | amqsgbr0 | amq0gbr0 | amqsgbr | amqsgbrc |
| Przeglądanie kompletnych komunikatów | amqsbcg0 | brak próbki | amqsbcg | amqsbcgc |
| Korzystanie ze współużytkowanej kolejki wejściowej | amqsinqa | amqminqx amqiinqx | amqsinq | amqsinqc |
| Korzystanie z wyłącznej kolejki wejściowej | amqstrg0 | amq0req0 | amqstrg | amqstrgc |
| Korzystanie z wywołania MQINQ | amqsinqa | amqminqx amqiinqx | amqsinq | brak próbki |
| Korzystanie z wywołania MQSET | amqsseta | amqmsetx amqisetx | amqsset, | amqssetc |
| Korzystanie z kolejki odpowiedzi | amqsreq0 | amq0req0 | amqsreq | amqsreqc |
| Zgłaszanie wyjątków komunikatów | amqsreq0 | amq0req0 | amqsreq | brak próbki |
| Akceptowanie obciętego komunikatu | amqsgbr0 | amq0gbr0 | amqsgbr | brak próbki |
| Używanie rozstrzygniętej nazwy kolejki | amqsgbr0 | amq0gbr0 | amqsgbr | brak próbki |
| Wyzwalanie procesu | amqstrg0 | brak próbki | amqstrg | amqstrgc |
| Korzystanie z konwersji danych | ( “4” na stronie 1095 ) | brak próbki | brak próbki | brak próbki |
| IBM MQ (koordynowanie menedżerów baz danych zgodnych z interfejsem XA) uzyskujących dostęp do pojedynczej bazy danych przy użyciu języka SQL | amqsxas0.sqc Db2 amqsxas0.ec Informix | amq0xas0.sqb | brak próbki | brak próbki |
| IBM MQ (koordynowanie menedżerów baz danych zgodnych z interfejsem XA) uzyskujących dostęp do dwóch baz danych przy użyciu języka SQL | amqsxag0.c amqsxab0.sqc amqsxaf0.sqc | amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb | brak próbki | brak próbki |
| Transakcja CICS (“5” na stronie 1095) | amqscic0.ccs | brak próbki | amqscic0 | brak próbki |
| Transakcja encina (“3” na stronie 1094) | amqsxae0 | brak próbki | amqsxae0 | brak próbki |

*Tabela 156. Przykładowe programy demonstrujące użycie interfejsu MQI (C i COBOL) w systemie AIX and Linux.*

Tabela z czterema kolumnami. W pierwszych kolumnach wymieniono techniki wykazane przez próbki. Druga kolumna zawiera listę przykładów w języku C, a trzecia-przykłady w języku COBOL, które demonstrują każdą z technik wymienionych w pierwszej kolumnie. Czwarta kolumna pokazuje, czy plik wykonywalny serwera C jest, czy nie jest, oraz piąta kolumna pokazuje, czy plik wykonywalny klienta C jest, czy nie jest.

*(kontynuacja)*

| **Technika** | **C (źródło) ("1" na stronie 1094)** | **COBOL (źródło) ("2" na stronie 1094)** | **Serwer (kod wykonywalny C)** | **Klient (kod wykonywalny C)** |
|---|---|---|---|---|
| Transakcja TUXEDO do umieszczania komunikatów "6" na stronie 1095) | amqstxpx, | brak próbki | brak próbki | brak próbki |
| Transakcja TUXEDO pobierania komunikatów ( "6" na stronie 1095) | amqstxgx | brak próbki | brak próbki | brak próbki |
| Serwer dla TUXEDO ("6" na stronie 1095) | amqstxsx | brak próbki | brak próbki | brak próbki |
| procedura obsługi kolejki niedostarczonych komunikatów | Katalog `./tools/c/Samples/dlq` ("7" na stronie 1095) | brak próbki | amqsdlq, | brak próbki |
| Z klienta MQI, umieszczanie komunikatu | brak próbki | brak próbki | brak próbki | amqsputc |
| Pobieranie komunikatu z klienta MQI | brak próbki | brak próbki | brak próbki | amqsgetc |
| Nawiązywanie połączenia z menedżerem kolejek przy użyciu programu MQCONNX | amqscnxc | brak próbki | brak próbki | amqscnxc |
| Korzystanie z wyjść funkcji API | amqsaxe0 | brak próbki | amqsaxe | brak próbki |
| Wyjście równoważenia obciążenia klastra | amqswlm0 | brak próbki | amqswlm | brak próbki |
| Asynchroniczne umieszczanie komunikatów i pobieranie statusu przy użyciu wywołania MQSTAT | amqsapt0 | brak próbki | amqsapt | amqsaptc |
| Klienty z możliwością ponownego nawiązania połączenia | amqsphac<br>amqsghac<br>amqsmhac | brak próbki | nie dotyczy | amqsphac<br>amqsghac<br>amqsmhac |
| Używanie konsumentów komunikatów do asynchronicznego odbierania komunikatów z wielu kolejek | amqscbf0 | brak próbki | amqscbf, | amqscbfc |
| Określanie informacji o połączeniu TLS w MQCONNX | amqsssl c | brak próbki | nie dotyczy | amqsssl c |

**Uwagi:**

1. Wersja wykonywalna przykładów IBM MQ MQI client współużytkuje to samo źródło, co przykłady uruchamiane w środowisku serwera.
2. Kompiluj programy rozpoczynające się od łańcucha amqm przy użyciu kompilatora Micro Focus COBOL, programy rozpoczynające się od łańcucha amqi przy użyciu kompilatora IBM COBOL oraz programy rozpoczynające się od łańcuchaamq0przy użyciu jednego z tych języków.
3. AIX Obsługiwane tylko w systemach IBM MQ for AIX .

4. AIX W systemach IBM MQ for AIX ten program nosi nazwę `amqsvfc0.c`
5. AIX Produkt CICS jest obsługiwany tylko przez produkty IBM MQ for AIX .
6. Linux Funkcja TUXEDO nie jest obsługiwana przez produkt IBM MQ for Linux w systemie System p.
7. Źródło programu do obsługi niewysłanych wiadomości składa się z kilku plików i znajduje się w oddzielnym katalogu.

Szczegółowe informacje na temat wsparcia dla systemów AIX and Linux zawiera sekcja Wymagania systemowe produktu IBM MQ.

Windows *Przykłady dla IBM MQ for Windows*

Techniki demonstrowane przez programy przykładowe dla produktu IBM MQ for Windows.

Tabela 157 na stronie 1095 zawiera listę plików źródłowych w językach C i COBOL oraz informacje o tym, czy plik wykonywalny serwera lub klienta został dołączony.

*Tabela 157. Przykładowe programy IBM MQ for Windows demonstrujące użycie interfejsu MQI (C i COBOL)*

| **Technika** | **C (źródło)** | **COBOL (źródło)** | **Serwer (kod wykonywalny C)** | **Klient (kod wykonywalny C)** |
|---|---|---|---|---|
| Korzystanie z interfejsu publikowania/ subskrypcji | amqspuba<br>amqssuba<br>amqssbxa | brak próbki | amqspub<br>amqssub<br>amqssbx | brak próbki |
| Umieszczanie komunikatów przy użyciu wywołania MQPUT | amqsput0 | amq0put0 | amqsput | amqsputc |
| Umieszczanie pojedynczego komunikatu przy użyciu wywołania MQPUT1 | amqsinqa<br>amqsecha | amqminq2<br>amqmech2<br>amqiinq2<br>amqiech2 | amqsinq<br>amqsech | amqsinqc<br>amqsechc |
| Umieszczanie komunikatów na liście dystrybucyjnej | amqsptl0 | amq0ptl0.cbl | amqsptl, | amqsptlc |
| Odpowiadanie na komunikat żądania | amqsinqa | amqminq2<br>amqiinq2 | amqsinq | amqsinqc |
| Pobieranie komunikatów (bez oczekiwania) | amqsgbr0 | amq0gbr0 | amqsgbr | amqsgbrc |
| Pobieranie komunikatów (oczekiwanie z limitem czasu) | amqsget0 | amq0get0 | amqsget | amqsgetc |
| Pobieranie komunikatów (nieograniczony czas oczekiwania) | amqstrg0 | brak próbki | amqstrg | amqstrgc |
| Pobieranie komunikatów (z konwersją danych) | amqsecha | brak próbki | amqsech | amqsechc |
| Umieszczanie komunikatów referencyjnych w kolejce | amqsprma | brak próbki | amqsprm, | amqsprmc |
| Pobieranie komunikatów referencyjnych z kolejki | amqsgrma | brak próbki | amqsgrm | amqsgrmc |
| Wyjście kanału komunikatów odniesienia | amqsqrma<br>amqsxrma | brak próbki | amqsxrm, | brak próbki |
| Przeglądanie pierwszych 20 znaków wiadomości | amqsgbr0 | amq0gbr0 | amqsgbr | amqsgbrc |

*Tabela 157. Przykładowe programy IBM MQ for Windows demonstrujące użycie interfejsu MQI (C i COBOL) (kontynuacja)*

| **Technika** | **C (źródło)** | **COBOL (źródło)** | **Serwer (kod wykonywalny C)** | **Klient (kod wykonywalny C)** |
|---|---|---|---|---|
| Przeglądanie kompletnych komunikatów | amqsbcg0 | brak próbki | amqsbcg | amqsbcgc |
| Korzystanie ze współużytkowanej kolejki wejściowej | amqsinqa | amqminq2<br>amqiinq2 | amqsinq | amqsinqc |
| Korzystanie z wyłącznej kolejki wejściowej | amqstrg0 | amq0req0 | amqstrg | amqstrgc |
| Korzystanie z wywołania MQINQ | amqsinqa | amqminq2<br>amqiinq2 | amqsinq | amqsinqc |
| Korzystanie z wywołania MQSET | amqsseta | amqmset2<br>amqiset2 | amqsset, | amqssetc |
| Korzystanie z wywołania MQINQMP | amqsiqma | brak próbki | brak próbki | brak próbki |
| Korzystanie z kolejki odpowiedzi | amqsreq0 | amq0req0 | amqsreq | amqsreqc |
| Zgłaszanie wyjątków komunikatów | amqsreq0 | amq0req0 | amqsreq | amqsreqc |
| Akceptowanie obciętego komunikatu | amqsgbr0 | amq0gbr0 | amqsgbr | amqsgbrc |
| Używanie rozstrzygniętej nazwy kolejki | amqsgbr0 | amq0gbr0 | amqsgbr | amqsgbrc |
| Wyzwalanie procesu | amqstrg0 | brak próbki | amqstrg | amqstrgc |
| Korzystanie z konwersji danych | amqsvfc0 | brak próbki | brak próbki | brak próbki |
| IBM MQ (koordynowanie menedżerów baz danych zgodnych z interfejsem XA) uzyskujących dostęp do pojedynczej bazy danych przy użyciu języka SQL | amqsxas0.sqc Db2<br>amqsxas0.ec Informix | amq0xas0.sqb | brak próbki | brak próbki |
| IBM MQ (koordynowanie menedżerów baz danych zgodnych z interfejsem XA) uzyskujących dostęp do dwóch baz danych przy użyciu języka SQL | amqsxag0.c<br>amqsxab0.sqc<br>Db2<br>amqsxaf0.sqc Db2 | amq0xag0.cbl<br>amq0xab0.sqb<br>amq0xaf0.sqb | brak próbki | brak próbki |
| Transakcja TUXEDO do umieszczania komunikatów | amqstxpx, | brak próbki | brak próbki | brak próbki |
| Transakcja TUXEDO do pobierania komunikatów | amqstxgx | brak próbki | brak próbki | brak próbki |
| Serwer dla TUXEDO | amqstxsx | brak próbki | brak próbki | brak próbki |
| procedura obsługi kolejki niedostarczonych komunikatów | Katalog `./tools/c/Samples/dlq` (“1” na stronie 1097 ) | brak próbki | amqsdlq, | brak próbki |
| Z IBM MQ MQI client-umieszczanie komunikatu | brak próbki | brak próbki | brak próbki | amqsputc |

*Tabela 157. Przykładowe programy IBM MQ for Windows demonstrujące użycie interfejsu MQI (C i COBOL) (kontynuacja)*

| Technika | C (źródło) | COBOL (źródło) | Serwer (kod wykonywalny C) | Klient (kod wykonywalny C) |
|---|---|---|---|---|
| Pobieranie komunikatu z serwisu IBM MQ MQI client | brak próbki | brak próbki | brak próbki | amqsgetc |
| Nawiązywanie połączenia z menedżerem kolejek przy użyciu programu MQCONNX | amqscnxc | brak próbki | brak próbki | amqscnxc |
| Korzystanie z wyjść funkcji API | amqsaxe0 | brak próbki | amqsaxe | brak próbki |
| Równoważenie obciążenia klastra | amqswlm0 | brak próbki | amqswlm | brak próbki |
| Procedury bezpieczeństwa SSPI | amqsspin | brak próbki | amqrspin.dll | amqrspin.dll |
| Asynchroniczne umieszczanie komunikatów i pobieranie statusu przy użyciu wywołania MQSTAT | amqsapt0 | brak próbki | amqsapt | amqsaptc |
| Klienty z możliwością ponownego nawiązania połączenia | amqsphac<br>amqsghac<br>amqsmhac | brak próbki | Nie dotyczy | amqsphac<br>amqsghac<br>amqsmhac |
| Używanie konsumentów komunikatów do asynchronicznego odbierania komunikatów z wielu kolejek | amqscbf0 | brak próbki | amqscbf, | amqscbfc |
| Określanie informacji o połączeniu TLS w MQCONNX | amqssslc | brak próbki | nie dotyczy | amqssslc |

**Uwagi:**

1. Źródło programu do obsługi niewysłanych wiadomości składa się z kilku plików i znajduje się w oddzielnym katalogu.

Windows *Przykłady Visual Basic dla systemu IBM MQ for Windows*

Techniki demonstrowane przez programy przykładowe dla systemu IBM MQ w systemach Windows .

Tabela 158 na stronie 1097 przedstawia techniki demonstrowane przez programy przykładowe IBM MQ for Windows .

Projekt może zawierać kilka plików. Po otwarciu projektu w języku Visual Basic pozostałe pliki są ładowane automatycznie. Nie udostępniono żadnych programów wykonywalnych.

Wszystkie przykładowe projekty, z wyjątkiem mqtrivc.vbp, są skonfigurowane do pracy z serwerem IBM MQ . Aby dowiedzieć się, w jaki sposób zmienić przykładowe projekty do pracy z klientami programu IBM MQ , należy zapoznać się z sekcją “Przygotowywanie programów Visual Basic w systemie Windows” na stronie 1050.

*Tabela 158. Przykładowe programy IBM MQ for Windows demonstrujące użycie interfejsu MQI (Visual Basic)*

| Technika | Nazwa pliku projektu |
|---|---|
| Umieszczanie komunikatów przy użyciu wywołania MQPUT | amqsputb.vbp |
| Pobieranie komunikatów przy użyciu wywołania MQGET | amqsgetb.vbp |
| Przeglądanie kolejki za pomocą wywołania MQGET | amqsbcgb.vbp |
| Prosty przykład MQGET i MQPUT (klient) | mqtrivc.vbp |

| *Tabela 158. Przykładowe programy IBM MQ for Windows demonstrujące użycie interfejsu MQI (Visual Basic) (kontynuacja)* | |
|---|---|
| **Technika** | **Nazwa pliku projektu** |
| Prosty przykład MQGET i MQPUT (serwer) | mqtrivs.vbp |
| Umieszczanie i pobieranie łańcuchów i struktur zdefiniowanych przez użytkownika za pomocą komend MQPUT i MQGET | strings.vbp |
| Używanie struktur PCF do uruchamiania i zatrzymywania kanału | pcfsamp.vbp |
| Tworzenie kolejki przy użyciu interfejsu MQAI | amqsaicq.vbp |
| Wyświetlanie kolejek menedżera kolejek za pomocą interfejsu MQAI | amqsailq.vbp |
| Monitorowanie zdarzeń przy użyciu interfejsu MQAI | amqsaiem.vbp |

IBM i *Przykłady dla IBM i*

Techniki demonstrowane przez programy przykładowe dla systemu IBM MQ w systemach IBM i .

Tabela 159 na stronie 1098 przedstawia techniki demonstrowane przez programy przykładowe IBM MQ for IBM i . Niektóre techniki występują w więcej niż jednym programie przykładowym, ale tylko jeden program jest wymieniony w tabeli.

| *Tabela 159. Przykładowe programy demonstrujące użycie interfejsu MQI (C i COBOL) w systemie IBM i* | | | | |
|---|---|---|---|---|
| **Technika** | **C (źródło) ( "1" na stronie 1100 )** | **COBOL (źródło) ( "2" na stronie 1100 )** | **RPG (źródło) ( "3" na stronie 1100 )** | **Klient (kod wykonywalny C) (4)** |
| Umieszczanie komunikatów przy użyciu wywołania MQPUT | AMQSPUT0 | AMQ0PUT4 | AMQ3PUT4 | amqsputc |
| Umieszczanie komunikatów ze zbioru danych przy użyciu wywołania MQPUT | AMQSPUT4 | brak próbki | brak próbki | brak próbki |
| Umieszczanie pojedynczego komunikatu przy użyciu wywołania MQPUT1 | AMQSINQ4, AMQSECH4 | AMQ0INQ4, AMQ0ECH4 | AMQ3INQ4, AMQ3ECH4 | AMQSINQC, AMQSECHC |
| Umieszczanie komunikatów na liście dystrybucyjnej | AMQSPTL4 | brak próbki | brak próbki | Komenda AMQSPTLC |
| Odpowiadanie na komunikat żądania | AMQSINQ4 | AMQ0INQ4 | AMQ3INQ4 | Komenda AMQSINQC |
| Pobieranie komunikatów (bez oczekiwania) | AMQSGBR4 | AMQ0GBR4 | AMQ3GBR4 | Komenda AMQSGBRC |
| Pobieranie komunikatów (oczekiwanie z limitem czasu) | AMQSGET4 | AMQ0GET4 | AMQ3GET4 | Komenda AMQSGETC |
| Pobieranie komunikatów (nieograniczony czas oczekiwania) | AMQSTRG4 | brak próbki | AMQ3TRG4 | Komenda AMQSTRGC |
| Pobieranie komunikatów (z konwersją danych) | AMQSECH4 | AMQ0ECH4 | AMQ3ECH4 | Komenda AMQSECHC |
| Umieszczanie komunikatów referencyjnych w kolejce | AMQSPRM4 | brak próbki | brak próbki | Komenda AMQSPRMC |
| Pobieranie komunikatów referencyjnych z kolejki | AMQSGRM4 | brak próbki | brak próbki | Komenda AMQSGRMC |

*Tabela 159. Przykładowe programy demonstrujące użycie interfejsu MQI (C i COBOL) w systemie IBM i (kontynuacja)*

| **Technika** | **C (źródło) ( "1" na stronie 1100 )** | **COBOL (źródło) ( "2" na stronie 1100 )** | **RPG (źródło) ( "3" na stronie 1100 )** | **Klient (kod wykonywalny C) (4)** |
|---|---|---|---|---|
| Wyjście kanału komunikatów odniesienia | AMQSQRM4, AMQSXRM4 | brak próbki | brak próbki | Brak próbki |
| Wyjście komunikatu | AMQSCMX4 | brak próbki | brak próbki | Brak próbki |
| Przeglądanie pierwszych 49 znaków komunikatu | AMQSGBR4 | AMQ0GBR4 | AMQ3GBR4 | Komenda AMQSGBRC |
| Przeglądanie kompletnych komunikatów | AMQSBCG4 | brak próbki | brak próbki | Komenda AMQSBCGC |
| Korzystanie ze współużytkowanej kolejki wejściowej | AMQSINQ4 | AMQ0INQ4 | AMQ3INQ4 | Komenda AMQSINQC |
| Korzystanie z wyłącznej kolejki wejściowej | AMQSREQ4 | AMQ0REQ4 | AMQ3REQ4 | AMQSREQC |
| Korzystanie z wywołania MQINQ | AMQSINQ4 | AMQ0INQ4 | AMQ3INQ4 | Komenda AMQSINQC |
| Korzystanie z wywołania MQSET | AMQSSET4 | AMQ0SET4 | AMQ3SET4 | Protokół AMQSSETC |
| Korzystanie z kolejki odpowiedzi | AMQSREQ4 | AMQ0REQ4 | AMQ3REQ4 | AMQSREQC |
| Zgłaszanie wyjątków komunikatów | AMQSREQ4 | AMQ0REQ4 | AMQ3REQ4 | AMQSREQC |
| Akceptowanie obciętego komunikatu | AMQSGBR4 | AMQ0GBR4 | AMQ3GBR4 | Komenda AMQSGBRC |
| Używanie rozstrzygniętej nazwy kolejki | AMQSGBR4 | AMQ0GBR4 | AMQ3GBR4 | Komenda AMQSGBRC |
| Wyzwalanie procesu | AMQSTRG4 | brak próbki | AMQ3TRG4 | Komenda AMQSTRGC |
| Serwer wyzwalaczy | AMQSERV4 | brak próbki | AMQ3SRV4 | brak próbki |
| Korzystanie z serwera wyzwalaczy (w tym z transakcji CICS ) | AMQSERV4 | brak próbki | AMQ3SRV4 | brak próbki |
| Korzystanie z konwersji danych | AMQSVFC4 | brak próbki | brak próbki | brak próbki |
| Korzystanie z wyjść funkcji API | AMQSAXE0 | brak próbki | brak próbki | brak próbki |
| Równoważenie obciążenia klastra | AMQSWLM0 | brak próbki | brak próbki | brak próbki |
| Asynchroniczne umieszczanie komunikatów i pobieranie statusu przy użyciu wywołania MQSTAT | AMQSAPT0 | brak próbki | brak próbki | Komenda AMQSAPTC |
| Korzystanie z interfejsu publikowania/ subskrypcji | AMQSPUBA, AMQSSUBA, AMQSSBXA | brak próbki | brak próbki | AMQSPUBC, AMQSSUBC, AMQSSBXC |
| Klienty z możliwością ponownego połączenia (5) | AMQSPHAC, AMQSGHAC, AMQSMHAC | brak próbki | brak próbki | brak próbki |

*Tabela 159. Przykładowe programy demonstrujące użycie interfejsu MQI (C i COBOL) w systemie IBM i (kontynuacja)*

| Technika | C (źródło) ( “1” na stronie 1100 ) | COBOL (źródło) ( “2” na stronie 1100 ) | RPG (źródło) ( “3” na stronie 1100 ) | Klient (kod wykonywalny C) (4) |
|---|---|---|---|---|
| Używanie konsumentów komunikatów do asynchronicznego odbierania komunikatów z wielu kolejek (5) | Usługa AMQSCBFO | brak próbki | brak próbki | brak próbki |
| Określanie informacji o połączeniu TLS w MQCONNX | Protokół AMQSSSLC | brak próbki | brak próbki | Protokół AMQSSSLC |
| Nawiązywanie połączenia z menedżerem kolejek przy użyciu programu MQCONNX | Komenda AMQSCNXC | brak próbki | brak próbki | Komenda AMQSCNXC |
| Sprawdzanie właściwości uchwytu komunikatu przy użyciu komendy MQINQMP z kolejki komunikatów | AMQISQMA, | brak próbki | brak próbki | AMQISQMC |
| Ustawianie właściwości uchwytu komunikatu za pomocą komendy MQSETMP i umieszczanie go w kolejce komunikatów | Komunikat AMQSSQMA | brak próbki | brak próbki | Komenda AMQSSQMC |

**Uwagi:**

1. Źródło przykładów C znajduje się w zbiorze QMQMSAMP/QCSRC. Zbiory włączane istnieją jako podzbiory w zbiorze QMQM/H.
2. Kod źródłowy przykładów COBOL znajduje się w zbiorach QMQMSAMP/QCBLLESRC. Elementy mają nazwę AMQ0 xxx 4, gdzie xxx wskazuje przykładową funkcję.
3. Kod źródłowy przykładów RPG znajduje się w QMQMSAMP/QRPGLESRC. Podzbiory mają nazwę AMQ3 xxx 4, gdzie xxx wskazuje przykładową funkcję. W QMQM/QRPGLESRC istnieją podzbiory kopii. Każda nazwa elementu ma przyrostek G.
4. Wersja wykonywalna przykładów IBM MQ MQI client współużytkuje to samo źródło, co przykłady uruchamiane w środowisku serwera. Źródło przykładów w środowisku klienta jest takie samo jak serwer. Przykłady IBM MQ MQI client są powiązane z biblioteką klienta LIBMQIC, a przykłady serwera IBM MQ z biblioteką serwera LIBMQM.
5. Jeśli należy uruchomić kod wykonywalny klienta dla przykładowej aplikacji klienta możliwego do ponownego połączenia i aplikacji konsumenta asynchronicznego, należy go skompilować i powiązać z biblioteką wielowątkową LIBMQIC_R. Dlatego musi być uruchamiany w środowisku wielowątkowym. Ustaw zmienną środowiskową QIBM_MULTI_THREADED na 'Y' i uruchom aplikację z powłoki qsh.

   Więcej informacji na ten temat zawiera sekcja Konfigurowanie produktu IBM MQ z użyciem usług Java i JMS .

Więcej informacji zawiera sekcja “Przygotowywanie i uruchamianie programów przykładowych w systemie IBM i” na stronie 1103.

Oprócz tych opcji, przykładowa opcja IBM MQ for IBM i zawiera przykładowy plik danych, który jest używany jako dane wejściowe dla programów przykładowych, AMQSDATA i przykładowe programy CL demonstrujące zadania administracyjne. Przykłady CL są opisane w sekcji Administrowanie produktem IBM i . Za pomocą przykładowego programu CL amqsamp4 można tworzyć kolejki do użycia z przykładowymi programami opisanymi w tym temacie.

Multi

## *Przygotowywanie i uruchamianie programów przykładowych*

Po zakończeniu wstępnego przygotowania można uruchomić programy przykładowe.

## O tym zadaniu

Przed uruchomieniem programów przykładowych należy najpierw utworzyć menedżer kolejek, a także potrzebne kolejki. Konieczne może być również wykonanie dodatkowych czynności przygotowawczych, na przykład w celu uruchomienia przykładów w języku COBOL. Po zakończeniu niezbędnych przygotowań można uruchomić programy przykładowe.

## Procedura

Informacje na temat przygotowywania i uruchamiania programów przykładowych znajdują się w następujących tematach:

- “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym” na stronie 1101
- “Przygotowywanie i uruchamianie programów przykładowych w systemie IBM i” na stronie 1103
- “Przygotowywanie i uruchamianie programów przykładowych w systemie AIX and Linux” na stronie 1104
- “Przygotowywanie i uruchamianie programów przykładowych w systemie Windows” na stronie 1105

Multi *Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym*

Przed uruchomieniem przykładowych aplikacji należy najpierw utworzyć menedżer kolejek. Następnie można skonfigurować menedżer kolejek w taki sposób, aby bezpiecznie akceptował przychodzące żądania połączeń z aplikacji działających w trybie klienta.

## Zanim rozpoczniesz

Upewnij się, że menedżer kolejek już istnieje i został uruchomiony. Określ, czy rekordy uwierzytelniania kanału są już włączone, wprowadzając komendę MQSC:

```
DISPLAY QMGR CHLAUTH
```

**Ważne:** To zadanie oczekuje, że rekordy uwierzytelniania kanału są włączone. Jeśli jest to menedżer kolejek używany przez innych użytkowników i aplikacje, zmiana tego ustawienia będzie miała wpływ na wszystkich innych użytkowników i aplikacje. Jeśli menedżer kolejek nie korzysta z rekordów uwierzytelniania kanału, krok 4 można zastąpić alternatywną metodą uwierzytelniania (na przykład wyjściem zabezpieczeń), która ustawia wartość MCAUSER na *identyfikator nieuprzywilejowanego użytkownika* , który zostanie uzyskany w kroku “1” na stronie 1101.

Aby aplikacja mogła korzystać z kanału, należy znać nazwę kanału, który ma być używany przez aplikację. Należy również wiedzieć, które obiekty, na przykład kolejki lub tematy, mają być używane przez aplikację, aby można było z nich korzystać.

## O tym zadaniu

To zadanie tworzy identyfikator użytkownika nieuprzywilejowanego, który ma być używany dla aplikacji klienckiej łączącej się z menedżerem kolejek. Dostęp jest przyznawany aplikacji klienckiej tylko po to, aby mogła używać kanału, którego potrzebuje, i kolejki, której potrzebuje, przy użyciu tego identyfikatora użytkownika.

## Procedura

1. Uzyskaj identyfikator użytkownika w systemie, w którym działa menedżer kolejek. W przypadku tego zadania ten identyfikator użytkownika nie może być uprzywilejowanym użytkownikiem administracyjnym. Ten ID użytkownika będzie uprawnieniem, które będzie używane do uruchamiania połączenia klienta w menedżerze kolejek.
2. Uruchom program nasłuchujący za pomocą następujących komend, gdzie:

   *nazwa_menedżera_kolejek* to nazwa menedżera kolejek.

*nnnn* jest wybranym numerem portu

a) ALW

W systemach AIX, Linux, and Windows :

```
runmqlsr -t tcp -m qmgr-name -p nnnn
```

b) IBM i

IBM i:

```
STRMQMLSR MQMNAME(qmgr-name) PORT(nnnn)
```

3. Jeśli aplikacja używa systemu SYSTEM.DEF.SVRCONN oznacza, że ten kanał jest już zdefiniowany. Jeśli aplikacja używa innego kanału, utwórz go, wprowadzając następującą komendę MQSC:

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

gdzie *nazwa-kanału* jest nazwą kanału.

4. Utwórz regułę uwierzytelniania kanału umożliwiającą użycie kanału tylko adresowi IP systemu klienta, wydając następującą komendę MQSC:

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

where

*nazwa-kanału* jest nazwą kanału.

*client-machine-IP-address* to adres IP systemu klienta. Jeśli przykładowa aplikacja kliencka działa na tym samym komputerze, co menedżer kolejek, należy użyć adresu IP127.0.0.1, jeśli aplikacja ma nawiązywać połączenie przy użyciu hosta lokalnego. Jeśli kilka różnych komputerów klienckich ma się łączyć, można użyć wzorca lub zakresu zamiast pojedynczego adresu IP. Szczegółowe informacje na ten temat zawiera sekcja Ogólne adresy IP .

*identyfikator nieuprzywilejowanego użytkownika* to identyfikator użytkownika uzyskany w kroku “1” na stronie 1101

5. Jeśli aplikacja używa systemu SYSTEM.DEFAULT.LOCAL.QUEUE to ta kolejka jest już zdefiniowana. Jeśli aplikacja używa innej kolejki, utwórz ją, wydając następującą komendę MQSC:

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

gdzie *nazwa_kolejki* jest nazwą kolejki.

6. Nadaj dostęp do połączenia z menedżerem kolejek i utwórz do niego zapytanie, wprowadzając następującą komendę MQSC:

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

gdzie *non-privileged-user-id* jest identyfikatorem użytkownika uzyskanym w kroku “1” na stronie 1101 .

7. Jeśli aplikacja jest aplikacją typu punkt z punktem, to znaczy korzysta z kolejek, należy nadać dostęp umożliwiający uzyskiwanie zapytań oraz umieszczanie i pobieranie komunikatów przy użyciu kolejki przez ID użytkownika, który ma być używany, za pomocą następujących komend MQSC:

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

where

*nazwa_kolejki* to nazwa kolejki.
*identyfikator nieuprzywilejowanego użytkownika* to identyfikator użytkownika uzyskany w kroku “1” na stronie 1101

8. Jeśli aplikacja jest aplikacją publikującą/subskrybującą, czyli korzysta z tematów, należy przyznać dostęp umożliwiający publikowanie i subskrybowanie przy użyciu tematu przy użyciu identyfikatora użytkownika, który ma być używany, za pomocą komend MQSC:

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

where

*identyfikator nieuprzywilejowanego użytkownika* to identyfikator użytkownika uzyskany w kroku “1” na stronie 1101
Spowoduje to nadanie *nieuprawnionemu ID użytkownika* dostępu do dowolnego tematu w drzewie tematów. Można również zdefiniować obiekt tematu za pomocą komendy **DEFINE TOPIC** i nadać dostęp tylko do części drzewa tematów, do której odwołuje się ten obiekt tematu. Szczegółowe informacje na ten temat zawiera sekcja Kontrolowanie dostępu użytkowników do tematów .

## Co dalej

Aplikacja kliencka może teraz nawiązać połączenie z menedżerem kolejek i umieszczać komunikaty przy użyciu kolejki.

**Pojęcia pokrewne**

ALW Nadawanie dostępu do obiektu IBM MQ w systemie AIX, Linux, and Windows

**Odsyłacze pokrewne**

USTAW CHLAURA
Zdefiniowanie kanału
ZDEFINIUJ QLOCAL
SET AUTHREC
IBM i Uprawnienia IBM MQ w systemie IBM i

IBM i *Przygotowywanie i uruchamianie programów przykładowych w systemie IBM i*

Przed uruchomieniem programów przykładowych w systemie IBM inależy najpierw utworzyć menedżer kolejek, a także potrzebne kolejki. Aby uruchomić przykłady w języku COBOL, może być konieczne wykonanie dodatkowych czynności przygotowawczych.

## O tym zadaniu

Kod źródłowy programów przykładowych IBM MQ for IBM i jest dostępny w bibliotece QMQMSAMP jako podzbiory QCSRC, QCLSRC, QCBLLESRC i QRPGLESRC.

Podczas uruchamiania przykładów można używać własnych kolejek lub można uruchomić przykładowy program AMQSAMP4 w celu utworzenia przykładowych kolejek. Źródło programu AMQSAMP4 znajduje się w zbiorze QCLSRC w bibliotece QMQMSAMP. Można ją skompilować przy użyciu komendy CRTCLPGM.

Aby uruchomić przykłady, należy użyć wersji kodu wykonywalnego w języku C, które są dostarczane w bibliotece QMQM, lub skompilować je w sposób podobny do innych aplikacji IBM MQ .

## Procedura

1. Utwórz menedżer kolejek i skonfiguruj definicje domyślne.

   Należy to zrobić przed uruchomieniem dowolnego z programów przykładowych. Więcej informacji na temat tworzenia menedżera kolejek zawiera sekcja Administrowanie produktem IBM MQ. Informacje na temat konfigurowania menedżera kolejek w celu bezpiecznego akceptowania żądań połączeń

przychodzących z aplikacji działających w trybie klienta zawiera sekcja “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym” na stronie 1101.

2. Aby wywołać jeden z przykładowych programów przy użyciu danych z podzbioru PUT w zbiorze AMQSDATA biblioteki QMQMSAMP, należy użyć następującej komendy:

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

   **Uwaga:** Aby skompilowany moduł używał systemu plików IFS, należy podać opcję SYSIFCOPT (*IFSIO) w komendzie CRTCMOD, a następnie należy podać nazwę pliku przekazaną jako parametr w następującym formacie:

```
home/me/myfile
```

3. Jeśli mają być używane wersje COBOL przykładów Inquire, Set i Echo, należy zmienić definicje procesów przed uruchomieniem tych przykładów.

   W przykładach Inquire, Set i Echo przykładowe definicje wyzwalają wersje C tych przykładów. Aby uzyskać wersje w języku COBOL, należy zmienić definicje procesów:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   W systemie IBM imożna użyć komendy **CHGMQMPRC** (szczegółowe informacje na ten temat zawiera sekcja Zmiana procesu MQ (CHGMQMPRC) ) lub zmodyfikować i uruchomić komendę **AMQSAMP4** z alternatywną definicją.

4. Uruchom programy przykładowe.

   Więcej informacji na temat parametrów oczekiwanych przez każdą z próbek można znaleźć w opisach poszczególnych prób.

   **Uwaga:** W przypadku przykładowych programów w języku COBOL, gdy nazwy kolejek są przekazywane jako parametry, należy podać 48 znaków, dopełnianie odstępami, jeśli jest to konieczne. Wszystkie znaki inne niż 48 znaków powodują, że działanie programu kończy się niepowodzeniem z kodem przyczyny 2085.

**Odsyłacze pokrewne**

“Przykłady dla IBM i” na stronie 1098
Techniki demonstrowane przez programy przykładowe dla systemu IBM MQ w systemach IBM i .

Linux AIX *Przygotowywanie i uruchamianie programów przykładowych w systemie AIX and Linux*

Przed uruchomieniem programów przykładowych w systemie AIX and Linuxnależy najpierw utworzyć menedżer kolejek, a także potrzebne kolejki. Aby uruchomić przykłady w języku COBOL, może być konieczne wykonanie dodatkowych czynności przygotowawczych.

## O tym zadaniu

Przykładowe pliki produktu IBM MQ w systemach AIX and Linux znajdują się w katalogach wymienionych w sekcji Tabela 160 na stronie 1104 , jeśli w czasie instalacji użyto wartości domyślnych.

*Tabela 160. Gdzie można znaleźć przykłady dla produktu IBM MQ w systemach AIX and Linux*

| **Zawartość** | **Katalog** |
|---|---|
| Pliki źródłowe | *MQ_INSTALLATION_PATH*/samp |
| pliki źródłowe programu obsługi kolejki niedostarczonych komunikatów | *MQ_INSTALLATION_PATH*/samp/dlq |
| Pliki wykonywalne | *MQ_INSTALLATION_PATH*/samp/bin |

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Do pracy z przykładami potrzebny jest zestaw kolejek. Można użyć własnych kolejek lub uruchomić przykładowy plik MQSC amqscos0.tst w celu utworzenia zestawu. Aby uruchomić przykłady, należy użyć dostarczonych wersji kodu wykonywalnego lub skompilować wersje kodu źródłowego w taki sam sposób, jak w przypadku innych aplikacji, przy użyciu kompilatora ANSI.

## Procedura

1. Utwórz menedżer kolejek i skonfiguruj definicje domyślne.

   Należy to zrobić przed uruchomieniem dowolnego z programów przykładowych. Więcej informacji na temat tworzenia menedżera kolejek zawiera sekcja Administrowanie produktem IBM MQ. Informacje na temat konfigurowania menedżera kolejek w celu bezpiecznego akceptowania żądań połączeń przychodzących z aplikacji działających w trybie klienta zawiera sekcja “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym” na stronie 1101.
2. Jeśli nie używasz własnych kolejek, uruchom przykładowy plik MQSC amqscos0.tst , aby utworzyć zestaw kolejek.

   W tym celu w systemach AIX and Linux wpisz:

   ```
   runmqsc QManagerName <amqscos0.tst > /tmp/sampobj.out
   ```

   Sprawdź plik `sampobj.out` , aby upewnić się, że nie wystąpiły żadne błędy.
3. Jeśli mają być używane wersje COBOL przykładów Inquire, Set i Echo, należy zmienić definicje procesów przed uruchomieniem tych przykładów.

   W przykładach Inquire, Set i Echo przykładowe definicje wyzwalają wersje C tych przykładów. Aby uzyskać wersje w języku COBOL, należy zmienić definicje procesów:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   W systemie Windowsnależy to zrobić, edytując plik `amqscos0.tst` i zmieniając nazwy plików wykonywalnych w języku C na nazwy plików wykonywalnych w języku COBOL przed użyciem komendy **runmqsc** do uruchomienia tych przykładów.
4. Uruchom programy przykładowe.

   Aby uruchomić przykład, wprowadź jego nazwę, po której następują dowolne parametry, na przykład:

   ```
   amqsput myqueue qmanagername
   ```

   gdzie *myqueue* jest nazwą kolejki, w której zostaną umieszczone komunikaty, a *qmanagername* jest menedżerem kolejek, który jest właścicielem produktu *myqueue*.

   Więcej informacji na temat parametrów oczekiwanych przez każdą z próbek można znaleźć w opisach poszczególnych prób.

   **Uwaga:** W przypadku przykładowych programów w języku COBOL, gdy nazwy kolejek są przekazywane jako parametry, należy podać 48 znaków, dopełnianie odstępami, jeśli jest to konieczne. Wszystkie znaki inne niż 48 znaków powodują, że działanie programu kończy się niepowodzeniem z kodem przyczyny 2085.

**Odsyłacze pokrewne**

“Przykłady dla systemów AIX and Linux” na stronie 1092
Techniki demonstrowane przez programy przykładowe dla produktu IBM MQ for AIX or Linux.

Windows *Przygotowywanie i uruchamianie programów przykładowych w systemie Windows*

Przed uruchomieniem programów przykładowych w systemie Windowsnależy najpierw utworzyć menedżer kolejek, a także potrzebne kolejki. Aby uruchomić przykłady w języku COBOL, może być konieczne wykonanie dodatkowych czynności przygotowawczych.

## O tym zadaniu

Przykładowe pliki IBM MQ for Windows znajdują się w katalogach wymienionych w sekcji Tabela 161 na stronie 1106, jeśli w czasie instalacji użyto wartości domyślnych. Wartością domyślną napędu instalacyjnego jest < c: >.

*Tabela 161. Gdzie znaleźć przykłady dla produktu IBM MQ for Windows*

| **Zawartość** | **Katalog** |
|---|---|
| Kod źródłowy w języku C | *MQ_INSTALLATION_PATH*\Tools\C\Przykłady |
| Kod źródłowy przykładu programu obsługi niedostarczonych komunikatów | *MQ_INSTALLATION_PATH*\Tools\C\Samples\DLQ |
| Kod źródłowy COBOL | *MQ_INSTALLATION_PATH*\Tools\Cobol \ Przykłady |
| Pliki wykonywalne C [1] | *MQ_INSTALLATION_PATH*\ Tools\C\Samples \ Bin (wersje 32-bitowe)<br>*MQ_INSTALLATION_PATH*\ Tools\C\Samples\Bin64 (wersje 64-bitowe) |
| Przykładowe pliki MQSC | *MQ_INSTALLATION_PATH*\Tools\MQSC\Przykłady |
| Kod źródłowy Visual Basic | *MQ_INSTALLATION_PATH*\Tools\VB\SampVB6 |
| .NET – przykłady | *MQ_INSTALLATION_PATH*\Tools\dotnet \ Przykłady |

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

**Uwaga:** Dostępne są 64-bitowe wersje niektórych przykładowych plików wykonywalnych w języku C.

Do pracy z przykładami potrzebny jest zestaw kolejek. Można użyć własnych kolejek lub uruchomić przykładowy plik MQSC amqscos0.tst w celu utworzenia zestawu kolejek. Aby uruchomić przykłady, należy użyć dostarczonych wersji wykonywalnych lub skompilować wersje źródłowe w taki sam sposób, jak w przypadku innych aplikacji IBM MQ for Windows .

## Procedura

1. Utwórz menedżer kolejek i skonfiguruj definicje domyślne.

   Należy to zrobić przed uruchomieniem dowolnego z programów przykładowych. Więcej informacji na temat tworzenia menedżera kolejek zawiera sekcja Administrowanie produktem IBM MQ. Informacje na temat konfigurowania menedżera kolejek w celu bezpiecznego akceptowania żądań połączeń przychodzących z aplikacji działających w trybie klienta zawiera sekcja “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym” na stronie 1101.
2. Jeśli nie używasz własnych kolejek, uruchom przykładowy plik MQSC amqscos0.tst , aby utworzyć zestaw kolejek.

   W tym celu w systemach Windows wpisz:

   ```
   runmqsc QManagerName < amqscos0.tst > sampobj.out
   ```

   Sprawdź plik `sampobj.out` , aby upewnić się, że nie wystąpiły żadne błędy. Ten plik znajduje się w bieżącym katalogu.
3. Jeśli mają być używane wersje COBOL przykładów Inquire, Set i Echo, należy zmienić definicje procesów przed uruchomieniem tych przykładów.

   W przykładach Inquire, Set i Echo przykładowe definicje wyzwalają wersje C tych przykładów. Aby uzyskać wersje w języku COBOL, należy zmienić definicje procesów:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS

- SYSTEM.SAMPLE.ECHOPROCESS

W systemie Windowsnależy to zrobić, edytując plik `amqscos0.tst` i zmieniając nazwy plików wykonywalnych w języku C na nazwy plików wykonywalnych w języku COBOL przed użyciem komendy **runmqsc** do uruchomienia tych przykładów.

4. Uruchom programy przykładowe.

   Aby uruchomić przykład, wprowadź jego nazwę, po której następują dowolne parametry, na przykład:

   ```
   amqsput myqueue qmanagername
   ```

   gdzie *myqueue* jest nazwą kolejki, w której zostaną umieszczone komunikaty, a *qmanagername* jest menedżerem kolejek, który jest właścicielem produktu *myqueue*.

   Więcej informacji na temat parametrów oczekiwanych przez każdą z próbek można znaleźć w opisach poszczególnych prób.

   **Uwaga:** W przypadku przykładowych programów w języku COBOL, gdy nazwy kolejek są przekazywane jako parametry, należy podać 48 znaków, dopełnianie odstępami, jeśli jest to konieczne. Wszystkie znaki inne niż 48 znaków powodują, że działanie programu kończy się niepowodzeniem z kodem przyczyny 2085.

**Odsyłacze pokrewne**

“Przykłady dla IBM MQ for Windows” na stronie 1095
Techniki demonstrowane przez programy przykładowe dla produktu IBM MQ for Windows.

“Przykłady Visual Basic dla systemu IBM MQ for Windows” na stronie 1097
Techniki demonstrowane przez programy przykładowe dla systemu IBM MQ w systemach Windows .

### *Przykładowy program obsługi wyjścia funkcji API*

Przykładowe wyjście funkcji API generuje śledzenie MQI do pliku określonego przez użytkownika z przedrostkiem zdefiniowanym w zmiennej środowiskowej **MQAPI_TRACE_LOGFILE** .

Więcej informacji na temat wyjść funkcji API zawiera sekcja “Pisanie i kompilowanie wyjść funkcji API w wersji wieloplatformowej” na stronie 982.

**Źródło**
amqsaxe0.c

**Binarny**
amqsaxe

## Konfigurowanie dla przykładowego wyjścia

1. Dodaj następujące informacje do sekcji ApiExitLocal w pliku `qm.ini` .

   **Platformy inne niż Windows**

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH/samp/bin/amqsaxe
   Name=SampleApiExit
   ```

   gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog, w którym zainstalowano produkt IBM MQ .

   **Windows**

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH\Tools\c\Samples\bin\amqsaxe
   Name=SampleApiExit
   ```

   gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog, w którym zainstalowano produkt IBM MQ .

2. Ustawianie zmiennej środowiskowej **MQAPI_TRACE_LOGFILE**

```
MQAPI_TRACE_LOGFILE=/tmp/MqiTrace
```

3. Uruchom aplikację.

   Pliki wyjściowe są tworzone w katalogu `/tmp` o nazwach podobnych do następujących: `MqiTrace.pid.tid.log`.

### *Przykładowy program wykorzystania asynchronicznego*

Przykładowy program amqscbf demonstruje użycie obiektu MQCB i obiektu MQCTL do asynchronicznego odbierania komunikatów z wielu kolejek.

Program amqscbf jest udostępniany jako kod źródłowy w języku C oraz jako binarny plik wykonywalny klienta i serwera na platformach AIX, Linux, and Windows .

Program jest uruchamiany z wiersza komend i przyjmuje następujące parametry opcjonalne:

```
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
```

Podaj więcej niż jedną nazwę kolejki, aby odczytywać komunikaty z wielu kolejek (przykład obsługuje maksymalnie dziesięć kolejek).

**Uwaga:** Parametr **Reconnect type** jest poprawny tylko dla programów klienckich.

#### Przykład

W przykładzie przedstawiono program amqscbf uruchamiany jako program serwera odczytujący jeden komunikat z QL1 , a następnie zatrzymywany.

Użyj programu Eksplorator IBM MQ , aby umieścić komunikat testowy w systemie QL1. Zatrzymaj program, naciskając klawisz Enter.

```
C:\>amqscbf QL1
Sample AMQSCBF0 start

Press enter to end
Message Call (9 Bytes) :
Message 1

Sample AMQSCBF0 end
```

## Co przedstawia amqscbf

Przykład przedstawia sposób odczytywania komunikatów z wielu kolejek w kolejności ich nadejścia. Wymaga to dużo więcej kodu przy użyciu synchronicznego wywołania MQGET. W przypadku wykorzystania asynchronicznego nie jest wymagane odpytywanie, a program IBM MQzarządza wątkami i pamięcią masową. Przykład "rzeczywistego świata" musiałby zajmować się błędami; w przykładowej próbie błędy są wypisywane na konsoli.

Kod przykładowy składa się z następujących kroków:

1. Zdefiniuj funkcję zwrotną wykorzystania pojedynczego komunikatu.

```
void MessageConsumer(MQHCONN      hConn,
                     MQMD    * pMsgDesc,
                     MQGMO   * pGetMsgOpts,
                     MQBYTE  * Buffer,
```

```
                          MQCBC   * pContext)
{ ... }
```

2. Nawiąż połączenie z menedżerem kolejek,

```
MQCONNX(QMName,&cno,&Hcon,&CompCode,&CReason);
```

3. Otwórz kolejki wejściowe i powiąż każdą z nich z funkcją zwrotną `MessageConsumer` .

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   Parametr `cbd.CallbackFunction` nie musi być ustawiany dla każdej kolejki; jest to pole tylko wejściowe. Można jednak powiązać inną funkcję zwrotną z każdą kolejką.

4. Rozpoczęcie korzystania z komunikatów,

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Poczekaj, aż użytkownik nacisnie klawisz Enter, a następnie zatrzymaj przetwarzanie komunikatów.

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. Na koniec rozłącz się z menedżerem kolejek,

```
MQDISC(&Hcon,&CompCode,&Reason);
```

### *Przykładowy program asynchronicznego umieszczania*

Sekcja zawiera informacje na temat uruchamiania przykładu amqsapt i projektowania przykładowego programu Asynchronous Put.

Przykładowy program do asynchronicznego umieszczania umieszcza komunikaty w kolejce przy użyciu asynchronicznego wywołania MQPUT, a następnie pobiera informacje o statusie przy użyciu wywołania MQSTAT. Nazwa tego programu na różnych platformach znajduje się w sekcji “Funkcje demonstrowane w przykładowych programach w środowisku wieloplatformowym” na stronie 1092 .

## Uruchamianie przykładu amqsapt

Ten program przyjmuje do 6 parametrów:

1. Nazwa kolejki docelowej (wymagane)
2. Nazwa menedżera kolejek (opcjonalna)
3. Opcje otwierania (opcjonalne)
4. Opcje zamykania (opcjonalne)
5. Nazwa docelowego menedżera kolejek (opcjonalna)
6. Nazwa kolejki dynamicznej (opcjonalna)

Jeśli menedżer kolejek nie zostanie określony, program amqsapt nawiąże połączenie z domyślnym menedżerem kolejek.

## Projekt przykładowego programu asynchronicznego umieszczania

Program korzysta z wywołania MQOPEN z podanymi opcjami wyjściowymi lub z opcjami MQOO_OUTPUT i MQOO_FAIL_IF_QUIESCING w celu otwarcia kolejki docelowej na potrzeby umieszczania komunikatów.

Jeśli nie można otworzyć kolejki, program wyświetla komunikat o błędzie zawierający kod przyczyny zwrócony przez wywołanie MQOPEN. Aby zachować prostotę programu, w tym przypadku i w kolejnych wywołaniach MQI program używa wartości domyślnych dla wielu opcji.

Dla każdego wiersza danych wejściowych program odczytuje tekst do buforu i używa wywołania MQPUT z opcją MQPMO_ASYNC_RESPONSE w celu utworzenia komunikatu datagramu zawierającego tekst tego wiersza i asynchronicznego umieszczenia go w kolejce docelowej. Działanie programu jest kontynuowane do momentu, gdy zostanie osiągnięty koniec wejścia lub wywołanie MQPUT nie powiedzie się. Jeśli program osiągnie koniec danych wejściowych, zamyka kolejkę za pomocą wywołania MQCLOSE.

Następnie program wysyła wywołanie MQSTAT, zwracając strukturę MQSTS i wyświetla komunikaty zawierające liczbę pomyślnie umieszczonych komunikatów, liczbę umieszczonych komunikatów z ostrzeżeniem oraz liczbę niepowodzeń.

### *Przykładowe programy do przeglądania*

Przykładowe programy przeglądania przeglądają komunikaty w kolejce za pomocą wywołania MQGET.

Nazwy tych programów można znaleźć w sekcji "Funkcje demonstrowane w przykładowych programach w środowisku wieloplatformowym" na stronie 1092 .

## Projekt przykładowego programu przeglądania

Program otwiera kolejkę docelową za pomocą wywołania MQOPEN z opcją MQOO_BROWSE. Jeśli nie można otworzyć kolejki, program wyświetla komunikat o błędzie zawierający kod przyczyny zwrócony przez wywołanie MQOPEN.

Dla każdego komunikatu w kolejce program używa wywołania MQGET do skopiowania komunikatu z kolejki, a następnie wyświetla dane zawarte w komunikacie. Wywołanie MQGET używa następujących opcji:

**MQGMO_BROWSE_NEXT**
Po wywołaniu MQOPEN kursor przeglądania jest umieszczany logicznie przed pierwszym komunikatem w kolejce, dlatego ta opcja powoduje zwrócenie ***pierwszego*** komunikatu podczas pierwszego wywołania.

**MQGMO_NO_WAIT**
Program nie czeka, jeśli w kolejce nie ma żadnych komunikatów.

**MQGMO_ACCEPT_TRUNCATED_MSG**
Wywołanie MQGET określa bufor o stałej wielkości. Jeśli komunikat jest dłuższy niż ten bufor, program wyświetla obcięty komunikat wraz z ostrzeżeniem, że komunikat został obcięty.

Program demonstruje sposób, w jaki należy usunąć zawartość pól *MsgId* i *CorrelId* struktury MQMD po każdym wywołaniu MQGET, ponieważ wywołanie ustawia te pola na wartości zawarte w pobieranym komunikacie. Usunięcie zaznaczenia tych pól oznacza, że kolejne wywołania MQGET pobierają komunikaty w kolejności, w jakiej są przechowywane w kolejce.

Program kontynuuje działanie do końca kolejki; wywołanie MQGET zwraca kod przyczyny MQRC_NO_MSG_AVAILABLE i program wyświetla komunikat ostrzegawczy. Jeśli wywołanie MQGET nie powiedzie się, program wyświetli komunikat o błędzie zawierający kod przyczyny.

Następnie program zamyka kolejkę za pomocą wywołania MQCLOSE.

*Przykładowe programy do przeglądania dla systemu AIX, Linux, and Windows*
Należy rozważyć użycie tego tematu podczas nauki o przeglądaniu przykładowych programów w systemie AIX, Linux, and Windows.

Wersja C programu przyjmuje 2 parametry

1. Nazwa kolejki źródłowej (wymagana)
2. Nazwa menedżera kolejek (opcjonalna)

Jeśli menedżer kolejek nie jest określony, nawiązywane jest połączenie z menedżerem domyślnym. Na przykład wprowadź jedną z następujących wartości:

- `amqsgbr myqueue qmanagername`
- `amqsgbrc myqueue qmanagername`
- `amq0gbr0 myqueue`

gdzie `myqueue` jest nazwą kolejki, z której będą wyświetlane komunikaty, a `qmanagername` jest menedżerem kolejek, który jest właścicielem produktu `myqueue`.

Jeśli parametr `qmanagername`zostanie pominięty podczas uruchamiania przykładu w języku C, przyjmuje się, że domyślny menedżer kolejek jest właścicielem kolejki.

Wersja języka COBOL nie ma żadnych parametrów. Zostanie nawiązane połączenie z domyślnym menedżerem kolejek, a po jego uruchomieniu zostanie wyświetlone zapytanie:

```
Please enter the name of the target queue
```

W takim przypadku wyświetlanych jest tylko 50 pierwszych znaków każdego komunikatu, po których następuje łańcuch `- - - truncated` .

*Przeglądanie przykładowych programów w systemie IBM i*
Każdy program pobiera kopie wszystkich komunikatów w kolejce, która została określona podczas wywoływania programu; komunikaty pozostają w kolejce.

Można użyć dostarczonej kolejki SYSTEM.SAMPLE.LOCAL-uruchom przykładowy program Put jako pierwszy, aby umieścić niektóre komunikaty w kolejce. Można użyć kolejki SYSTEM.SAMPLE.ALIAS, która jest aliasem tej samej kolejki lokalnej. Działanie programu jest kontynuowane do momentu osiągnięcia końca kolejki lub niepowodzenia wywołania MQI.

Przykłady w języku C umożliwiają określenie nazwy menedżera kolejek, zwykle jako drugiego parametru, w sposób podobny do przykładowych systemów Windows . Na przykład:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER' 'QM01')
```

Jeśli menedżer kolejek nie jest określony, nawiązywane jest połączenie z menedżerem domyślnym. Dotyczy to również przykładów RPG. Jednak w przypadku przykładów RPG należy podać nazwę menedżera kolejek, zamiast zezwalać na jej ustawienie domyślne.

### ALW *Przykładowy program przeglądarki*

Przykładowy program przeglądarki odczytuje i zapisuje zarówno deskryptor komunikatu, jak i pola treści wszystkich komunikatów w kolejce.

Przykładowy program został napisany jako narzędzie, nie tylko w celu zademonstrowania techniki. Nazwy tych programów można znaleźć w sekcji “Funkcje demonstrowane w przykładowych programach w środowisku wieloplatformowym” na stronie 1092 .

Program ten przyjmuje następujące parametry pozycyjne:

1. Nazwa kolejki źródłowej (wymagane)
2. Nazwa menedżera kolejek (wymagane)
3. Parametr opcjonalny dla właściwości (opcjonalny)

Użyj następujących zmiennych środowiskowych, aby podać referencje używane do uwierzytelniania w menedżerze kolejek:

**ID_UŻYTKOWNIKA MQSAMP_ID**
Ustaw identyfikator użytkownika, który ma być używany do uwierzytelniania połączenia, jeśli do uwierzytelniania w menedżerze kolejek ma być używany identyfikator użytkownika i hasło. Program poprosi o podanie hasła, które ma być dołączone do identyfikatora użytkownika.

V 9.3.4 Linux AIX **MQSAMP_TOKEN**

Ustaw niepustą wartość, aby podać znacznik uwierzytelniania w celu uwierzytelnienia w menedżerze kolejek. Program pyta o znacznik uwierzytelniania. Znaczniki uwierzytelniania mogą być używane tylko przez przykład **amqsbcgc** , który używa powiązań klienta.

Aby uruchomić te programy, wprowadź jedną z następujących komend:

- `amqsbcg` *`myqueue qmanagername`*
- `amqsbcgc` *`myqueue qmanagername`*

gdzie *myqueue* jest nazwą kolejki, w której będą przeglądane komunikaty, a *qmanagername* jest menedżerem kolejek, który jest właścicielem produktu *myqueue*.

Odczytuje każdy komunikat z kolejki i zapisuje następujące informacje na wyjściu standardowym:

- Sformatowane pola deskryptora komunikatu
- Dane komunikatu (zrzucone w formacie szesnastkowym i, jeśli to możliwe, znakowym)

*Tabela 162. Dopuszczalne wartości parametru właściwości*

| **Wartość** | **zachowanie;** |
|---|---|
| 0 | Zachowanie domyślne. Właściwości dostarczane do aplikacji zależą od atrybutu kolejki **`PropertyControl`** , z którego jest pobierany komunikat. |
| 1 | Uchwyt komunikatu jest tworzony i używany z usługą MQGET. Właściwości komunikatu, z wyjątkiem tych, które są zawarte w deskryptorze komunikatu (lub rozszerzeniu), są wyświetlane w sposób podobny do deskryptora komunikatu. Na przykład:<br>`****Message properties****`<br>`  property name: property value`<br>Lub jeśli nie są dostępne żadne właściwości:<br>`****Message properties****`<br>`  None`<br>Wartości liczbowe są wyświetlane przy użyciu `printf`, wartości łańcuchowe są ujęte w pojedynczy cudzysłów, a łańcuchy bajtów są ujęte w znaki X i pojedynczy cudzysłów, tak jak w przypadku deskryptora komunikatu. |
| 2 | Określono parametr MQGMO_NO_PROPERTIES, więc zostaną zwrócone tylko właściwości deskryptora komunikatu. |
| 3 | Określono właściwość MQGMO_PROPERTIES_FORCE_MQRFH2 , więc wszystkie właściwości są zwracane w danych komunikatu. |
| 4 | Określono parametr MQGMO_PROPERTIES_COMPATIBILITY, dzięki czemu wszystkie właściwości mogą być zwracane w zależności od tego, czy właściwość IBM MQ jest dołączona, w przeciwnym razie właściwości są usuwane. |

Program jest ograniczony do drukowania pierwszych 65535 znaków komunikatu i kończy się niepowodzeniem z powodu `truncated msg` , jeśli odczytany jest dłuższy komunikat.

Przykład danych wyjściowych tego programu narzędziowego zawiera sekcja Przeglądanie kolejek.

### *Przykład transakcji CICS*

Udostępniono przykładowy program transakcyjny CICS o nazwie amqscic0.ccs dla kodu źródłowego i amqscic0 dla wersji pliku wykonywalnego. Transakcje można budować za pomocą standardowych narzędzi systemu CICS .

Szczegółowe informacje na temat komend wymaganych dla danej platformy zawiera sekcja “Budowanie aplikacji proceduralnej” na stronie 1029 .

Transakcja odczytuje komunikaty z kolejki transmisji SYSTEM.SAMPLE.CICS.WORKQUEUE w domyślnym menedżerze kolejek i umieszcza je w kolejce lokalnej, której nazwa jest zawarta w nagłówku transmisji komunikatu. Wszystkie niepowodzenia są wysyłane do kolejki SYSTEM.SAMPLE.CICS.DLQ.

**Uwaga:** Do utworzenia tych kolejek i przykładowych kolejek wejściowych można użyć przykładowego skryptu MQSC amqscic0.tst .

### *Przykładowy program Connect*

Program przykładowy Connect umożliwia zapoznanie się z wywołaniem MQCONNX i jego opcjami z poziomu klienta. Przykład nawiązuje połączenie z menedżerem kolejek przy użyciu wywołania MQCONNX, zapytuje o nazwę menedżera kolejek przy użyciu wywołania MQINQ i wyświetla ją. Należy również zapoznać się z informacjami na temat uruchamiania przykładu amqscnxc.

**Uwaga:** Przykładowy program Connect jest przykładem klienta. Można go skompilować i uruchomić na serwerze, ale funkcja ma znaczenie tylko na kliencie i dostarczane są tylko pliki wykonywalne klienta.

## Uruchamianie przykładu amqscnxc

Składnia wiersza komend programu przykładowego Connect jest następująca:

```
amqscnxc [-x ConnName [-c SvrconnChannelName]] [-u User] [QMgrName]
```

Parametry są opcjonalne, a ich kolejność nie jest istotna z wyjątkiem parametru QMgrName, który, jeśli został podany, musi być ostatni. Parametry są następujące:

**ConnName**
Nazwa połączenia TCP/IP menedżera kolejek serwera

Jeśli nazwa połączenia TCP/IP nie zostanie określona, zostanie wydana komenda MQCONNX z parametrem *ClientConnPtr* ustawionym na wartość NULL.

**SvrconnChannelNazwa**
Nazwa kanału połączenia z serwerem

Jeśli zostanie podana nazwa połączenia TCP/IP, ale nie kanał połączenia z serwerem (odwrotność nie jest dozwolona), w przykładzie zostanie użyta nazwa SYSTEM.DEF.SVRCONN.

**Użytkownik**
Nazwa użytkownika, która ma być używana do uwierzytelniania połączenia

Jeśli ta opcja zostanie podana, program poprosi o podanie hasła, które ma być dołączone do identyfikatora użytkownika.

**QMgrName**
Nazwa docelowego menedżera kolejek

Jeśli docelowy menedżer kolejek nie zostanie określony, przykład nawiąże połączenie z dowolnym menedżerem kolejek, który nasłuchuje przy użyciu podanej nazwy połączenia TCP/IP.

**Uwaga:** Jeśli jako jedyny parametr zostanie wprowadzony znak zapytania lub jeśli zostaną podane niepoprawne parametry, zostanie wyświetlony komunikat wyjaśniający sposób użycia programu.

Jeśli przykład zostanie uruchomiony bez opcji wiersza komend, do określenia informacji o połączeniu zostanie użyta zawartość zmiennej środowiskowej MQSERVER. (W tym przykładzie dla MQSERVER ustawiono wartość SYSTEM.DEF.SVRCONN/TCP/machine.site.company.com). Zostaną wyświetlone następujące dane wyjściowe:

```
Sample AMQSCNXC start
Connecting to the default queue manager
with no client connection information specified.
Connection established to queue manager machine

Sample AMQSCNXC end
```

W przypadku uruchomienia przykładu i podania nazwy połączenia TCP/IP i nazwy kanału połączenia serwera, ale bez nazwy docelowego menedżera kolejek, wykonaj następujące czynności:

```
amqscnxc -x machine.site.company.com -c SYSTEM.ADMIN.SVRCONN
```

używana jest domyślna nazwa menedżera kolejek, a dane wyjściowe są wyświetlane w następujący sposób:

```
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
```

Jeśli zostanie uruchomiony przykład i zostanie podana nazwa połączenia TCP/IP oraz nazwa docelowego menedżera kolejek, na przykład:

```
amqscnxc -x machine.site.company.com MACHINE
```

Zostaną wyświetlone dane wyjściowe podobne do następujących:

```
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
```

### *Przykładowy program do konwersji danych*

Przykładowy program do konwersji danych jest szkieletem procedury wyjścia konwersji danych. Sekcja zawiera informacje na temat projektowania przykładu konwersji danych.

Nazwy tych programów można znaleźć w sekcji “Funkcje demonstrowane w przykładowych programach w środowisku wieloplatformowym” na stronie 1092 .

## Projekt próbki konwersji danych

Każda procedura wyjścia konwersji danych przekształca pojedynczy nazwany format komunikatu. Ten szkielet jest przeznaczony jako opakowanie fragmentów kodu wygenerowanych przez program narzędziowy do generowania wyjścia konwersji danych.

Program narzędziowy tworzy po jednym fragmencie kodu dla każdej struktury danych; kilka takich struktur tworzy format, więc do tego szkieletu dodawanych jest kilka fragmentów kodu w celu utworzenia procedury konwersji danych całego formatu.

Następnie program sprawdza, czy konwersja zakończyła się powodzeniem, czy niepowodzeniem, i zwraca wartości wymagane do programu wywołującego.

### *Przykłady koordynacji bazy danych*

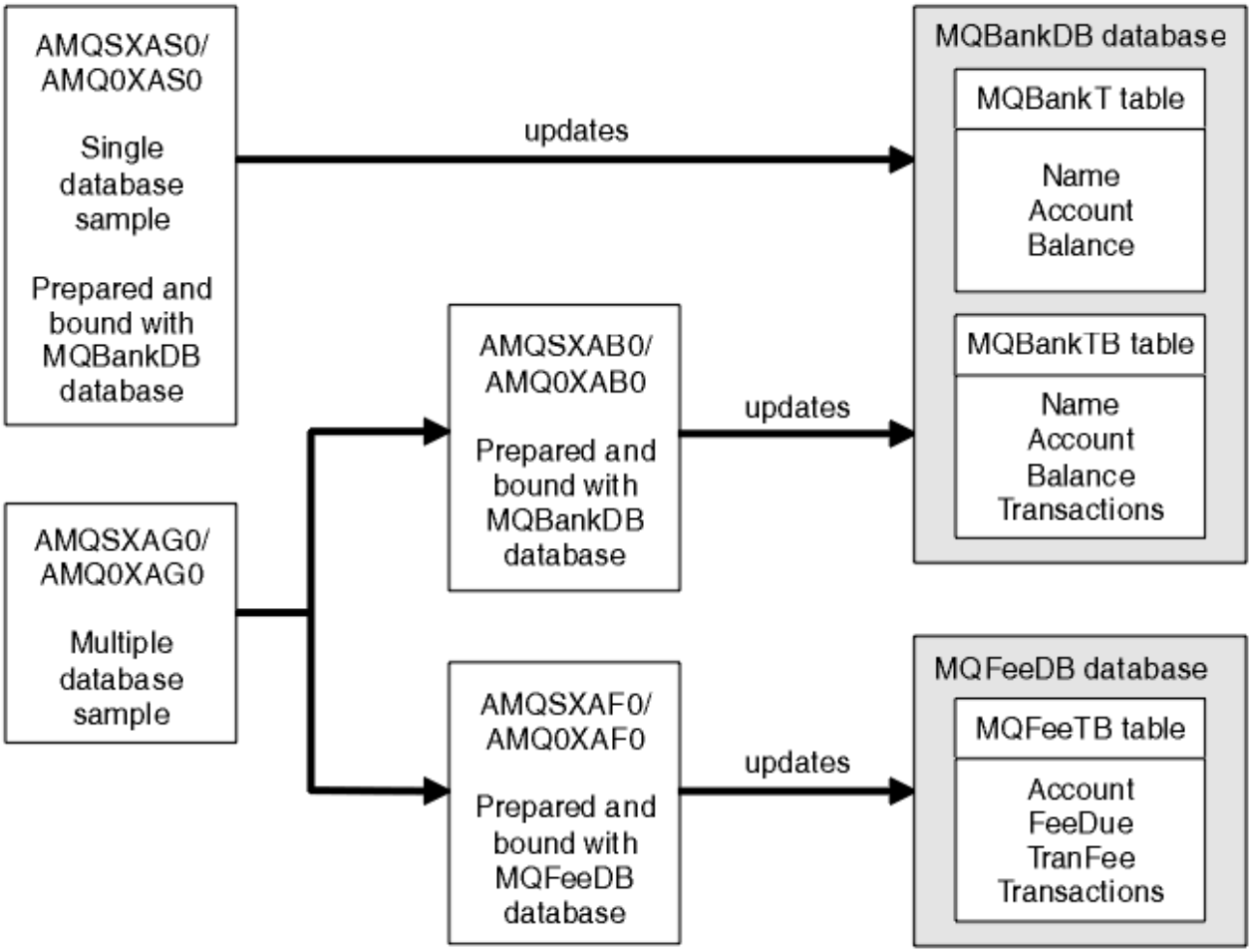
Dostępne są dwa przykłady demonstrujące, w jaki sposób produkt IBM MQ może koordynować zarówno aktualizacje IBM MQ , jak i aktualizacje bazy danych w ramach tej samej jednostki pracy.

Przykłady te są następujące:

1. AMQSXAS0 (w języku C) lub AMQ0XAS0 (w języku COBOL), który aktualizuje pojedynczą bazę danych w obrębie jednostki pracy produktu IBM MQ .
2. AMQSXAG0 (w języku C) lub AMQ0XAG0 (w języku COBOL), AMQSXAB0 (w języku C) lub AMQ0XAB0 (w języku COBOL) i AMQSXAF0 (w języku C) lub AMQ0XAF0 (w języku COBOL), które razem aktualizują dwie bazy danych w jednostce pracy produktu IBM MQ i pokazują, w jaki sposób można uzyskać

dostęp do wielu baz danych. Te przykłady są udostępniane w celu pokazania użycia wywołania MQBEGIN, mieszanych wywołań SQL i IBM MQ oraz miejsca i czasu połączenia z bazą danych.

Rysunek 128 na stronie 1115 pokazuje, w jaki sposób udostępnione przykłady są używane do aktualizowania baz danych:




*Rysunek 128. Przykłady koordynacji bazy danych*

Programy odczytują komunikat z kolejki (w punkcie synchronizacji), a następnie, korzystając z informacji zawartych w komunikacie, uzyskują odpowiednie informacje z bazy danych i aktualizują je. Następnie drukowany jest nowy status bazy danych.

Logika programu jest następująca:

1. Użyj nazwy kolejki wejściowej z argumentu programu
2. Nawiąż połączenie z domyślnym menedżerem kolejek (lub opcjonalnie z nazwą podaną w języku C) przy użyciu wywołania MQCONN
3. Otwórz kolejkę (przy użyciu MQOPEN) dla danych wejściowych, gdy nie ma żadnych niepowodzeń
4. Uruchamianie jednostki pracy za pomocą komendy MQBEGIN
5. Pobierz następny komunikat (za pomocą komendy MQGET) z kolejki w punkcie synchronizacji
6. Pobierz informacje z baz danych
7. Aktualizuj informacje z baz danych
8. Zatwierdź zmiany za pomocą MQCMIT
9. Wydrukuj zaktualizowane informacje (żaden komunikat nie jest dostępny jako błąd i pętla się kończy)
10. Zamknij kolejkę za pomocą komendy MQCLOSE
11. Rozłącz z kolejką za pomocą komendy MQDISC

W próbkach używane są kursory SQL, dzięki czemu odczyty z baz danych (czyli wielu instancji) są blokowane podczas przetwarzania komunikatu, co umożliwia jednoczesne uruchomienie wielu instancji tych programów. Kursory są jawnie otwierane, ale niejawnie zamykane przez wywołanie MQCMIT.

Przykładowa pojedyncza baza danych (AMQSXAS0 lub AMQ0XAS0) nie zawiera instrukcji SQL CONNECT, a połączenie z bazą danych jest nawiązywane niejawnie przez produkt IBM MQ za pomocą wywołania MQBEGIN. Wiele przykładowych baz danych (AMQSXAG0 lub AMQ0XAG0, AMQSXAB0 lub AMQ0XAB0i AMQSXAF0 lub AMQ0XAF0) zawiera instrukcje SQL CONNECT, ponieważ niektóre produkty bazy danych zezwalają na tylko jedno aktywne połączenie. Jeśli tak nie jest w przypadku produktu bazodanowego lub jeśli dostęp do pojedynczej bazy danych jest uzyskiwany w wielu produktach bazodanowych, można usunąć instrukcje SQL CONNECT.

Przykłady są przygotowywane z produktem bazodanowym IBM Db2 , dlatego może być konieczne ich zmodyfikowanie w celu umożliwienia współpracy z innymi produktami bazodanowymi.

Podczas sprawdzania błędów SQL używane są procedury w języku UTIL.C i CHECKERR.CBL dostarczony przez Db2. Należy je skompilować lub zastąpić przed kompilacją i dowiązaniem.

**Uwaga:** Jeśli używasz źródła Micro Focus COBOL CHECKERR.MFC dla sprawdzania błędów SQL, należy zmienić identyfikator programu na wielkie litery, czyli CHECKERR, aby komenda AMQ0XAS0 łąła się poprawnie.

*Tworzenie baz danych i tabel*
Przed skompilowaniem przykładów należy utworzyć bazy danych i tabele.

Aby utworzyć bazy danych, należy użyć zwykłej metody dla produktu bazodanowego, na przykład:

```
DB2 CREATE DB MQBankDB
DB2 CREATE DB MQFeeDB
```

Utwórz tabele za pomocą instrukcji SQL w następujący sposób:

W języku C:

```
EXEC SQL CREATE TABLE MQBankT(Name         VARCHAR(40) NOT NULL,
                              Account      INTEGER     NOT NULL,
                              Balance      INTEGER     NOT NULL,
                              PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQBankTB(Name         VARCHAR(40) NOT NULL,
                               Account      INTEGER     NOT NULL,
                               Balance      INTEGER     NOT NULL,
                               Transactions INTEGER,
                               PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQFeeTB(Account      INTEGER     NOT NULL,
                              FeeDue       INTEGER     NOT NULL,
                              TranFee      INTEGER     NOT NULL,
                              Transactions INTEGER,
                              PRIMARY KEY (Account));
```

W języku COBOL:

```
EXEC SQL CREATE TABLE
  MQBankT(Name         VARCHAR(40) NOT NULL,
          Account      INTEGER     NOT NULL,
          Balance      INTEGER     NOT NULL,
          PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQBankTB(Name         VARCHAR(40) NOT NULL,
           Account      INTEGER     NOT NULL,
           Balance      INTEGER     NOT NULL,
           Transactions INTEGER,
           PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
```

```
  MQFeeTB(Account      INTEGER   NOT NULL,
          FeeDue       INTEGER   NOT NULL,
          TranFee      INTEGER   NOT NULL,
          Transactions INTEGER,
          PRIMARY KEY (Account))
  END-EXEC.
```

Wprowadź dane do tabel za pomocą instrukcji SQL w następujący sposób:

```
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);
⋮
```

**Uwaga:** W przypadku języka COBOL należy użyć tych samych instrukcji SQL, ale na końcu każdego wiersza należy dodać łańcuch END_EXEC .

*Prekompilowanie, kompilowanie i łączenie przykładów*
Dowiedz się, jak prekompilować, kompilować i łączyć przykłady w językach C i COBOL.

Prekompiluj pliki `.SQC` (w języku C) i `.SQB` (w języku COBOL) i powiąż je z odpowiednią bazą danych, aby utworzyć pliki `.C` lub `.CBL` . W tym celu należy użyć typowej metody dla danego produktu bazodanowego.

## Prekompilacja w języku C

```
db2 connect to MQBankDB
db2 prep AMQSXAS0.SQC
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQSXAB0.SQC
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQSXAF0.SQC
db2 connect reset
```

## Prekompilacja w języku COBOL

```
db2 connect to MQBankDB
db2 prep AMQ0XAS0.SQB bindfile target ibmcob
db2 bind AMQ0XAS0.BND
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQ0XAB0.SQB bindfile target ibmcob
db2 bind AMQ0XAB0.BND
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQ0XAF0.SQB bindfile target ibmcob
db2 bind AMQ0XAF0.BND
db2 connect reset
```

## Kompilowanie i łączenie

W poniższych przykładowych komendach używane są symbole *DB2TOP* i *MQ_INSTALLATION_PATH*. *DB2TOP* reprezentuje katalog instalacyjny produktu Db2 . *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

- AIX W systemie AIXścieżka do katalogu jest następująca:

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

- Windows W systemach Windows ścieżka do katalogu zależy od ścieżki wybranej podczas instalowania produktu. Jeśli wybrano ustawienia domyślne, ścieżka jest następująca:

```
c:\sqllib
```

**Uwaga:** Przed wydaniem komendy dowiązania w systemach Windows należy upewnić się, że zmienna środowiskowa LIB zawiera ścieżki do bibliotek Db2 i IBM MQ .

Skopiuj następujące pliki do katalogu tymczasowego:

- Plik `amqsxag0.c` z instalacji produktu IBM MQ

  **Uwaga:** Plik ten znajduje się w następujących katalogach:

  – Linux AIX W systemach AIX and Linux:

  ```
  MQ_INSTALLATION_PATH/samp/xatm
  ```

  – Windows W systemach Windows:

  ```
  MQ_INSTALLATION_PATH\tools\c\samples\xatm
  ```

- Pliki `.c` uzyskane przez prekompilację plików źródłowych `.sqc` , `amqsxas0.sqc`, `amqsxaf0.sqc`i `amqsxab0.sqc`.
- Pliki `util.c` i `util.h` z instalacji produktu Db2 .

  **Uwaga:** Te pliki można znaleźć w katalogu:

  ```
  DB2TOP/samples/c
  ```

Zbuduj pliki obiektów dla każdego pliku `.c` , używając następującej komendy kompilatora dla używanej platformy:

- AIX AIX

```
xlc_r -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o
FILENAME.o FILENAME.c
```

- Windows Systemy Windows

```
cl /c /I MQ_INSTALLATION_PATH\tools\c\include /I DB2TOP\include
FILENAME.c
```

Zbuduj plik wykonywalny `amqsxag0` za pomocą następującej komendy link dla używanej platformy:

- AIX 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 Systemy Windows

```
link util.obj amqsxaf0.obj amqsxab0.obj amqsxag0.obj mqm.lib db2api.lib
/out:amqsxag0.exe
```

Zbuduj plik wykonywalny `amqsxas0` , używając następujących komend kompilacji i konsolidacji dla używanej platformy:

- AIX AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2
-L MQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
```

- Windows Systemy Windows

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

**Dodatkowe informacje**

AIX Jeśli pracujesz w systemie AIX i chcesz uzyskać dostęp do bazy danych Oracle, użyj kompilatora xlc_r i odsyłacza do pliku libmqm_r.a.

*Uruchamianie przykładów*
Te informacje umożliwiają poznanie sposobu konfigurowania menedżera kolejek przed uruchomieniem przykładów koordynacji bazy danych w językach C i COBOL.

Przed uruchomieniem przykładów należy skonfigurować menedżer kolejek z używaną bazą danych. Więcej informacji na ten temat zawiera sekcja Scenariusz 1: Menedżer kolejek wykonuje koordynację.

Poniższe tytuły zawierają informacje na temat uruchamiania przykładów w językach C i COBOL:


- “Przykłady C” na stronie 1119
- “Przykłady w języku COBOL” na stronie 1120


## Przykłady C

Komunikaty muszą mieć następujący format, aby można je było odczytać z kolejki:

```
UPDATE Balance change=nnn WHERE Account=nnn
```

Komenda AMQSPUT może zostać użyta do umieszczenia komunikatów w kolejce.

Przykłady koordynacji bazy danych mają dwa parametry:

1. Nazwa kolejki (wymagane)
2. Nazwa menedżera kolejek (opcjonalnie)

Zakładając, że został utworzony i skonfigurowany menedżer kolejek dla pojedynczej przykładowej bazy danych o nazwie singDBQMz kolejką o nazwie singDBQ, należy zwiększyć konto Roberta Bloggsa o 50 w następujący sposób:

```
AMQSPUT singDBQ singDBQM
```

Następnie należy nacisnąć klawisz w następującym komunikacie:

```
UPDATE Balance change=50 WHERE Account=1
```

W kolejce można umieścić wiele komunikatów.

```
AMQSXAS0 singDBQ singDBQM
```

Następnie drukowany jest zaktualizowany status konta pana Freda Bloggsa.

Zakładając, że został utworzony i skonfigurowany menedżer kolejek dla przykładu z wieloma bazami danych o nazwie multDBQMz kolejką o nazwie multDBQ, należy zmniejszyć konto pani Mary Brown o 75 w następujący sposób:

```
AMQSPUT multDBQ multDBQM
```

Następnie należy nacisnąć klawisz w następującym komunikacie:

```
UPDATE Balance change=-75 WHERE Account=3
```

W kolejce można umieścić wiele komunikatów.

```
AMQSXAG0 multDBQ multDBQM
```

Następnie drukowany jest zaktualizowany status konta Pani Mary Brown.

## Przykłady w języku COBOL

Komunikaty muszą mieć następujący format, aby można je było odczytać z kolejki:

```
UPDATE Balance change=snnnnnnnn WHERE Account=nnnnnnnn
```

Dla uproszczenia `Balance change` musi być ośmioznakową liczbą ze znakiem, a `Account` musi być ośmioznakową liczbą.

Do umieszczenia komunikatów w kolejce można użyć przykładowego programu AMQSPUT.

Przykłady nie pobierają żadnych parametrów i używają domyślnego menedżera kolejek. Można go skonfigurować w taki sposób, aby w danym momencie był uruchamiany tylko jeden z przykładów. Zakładając, że domyślny menedżer kolejek został skonfigurowany dla przykładu pojedynczej bazy danych z kolejką o nazwie singDBQ, należy zwiększyć konto pana Freda Bloggsa o 50 w następujący sposób:

```
AMQSPUT singDBQ
```

Następnie należy nacisnąć klawisz w następującym komunikacie:

```
UPDATE Balance change=+00000050 WHERE Account=00000001
```

W kolejce można umieścić wiele komunikatów:

```
AMQ0XAS0
```

Wpisz nazwę kolejki:

```
singDBQ
```

Następnie drukowany jest zaktualizowany status konta pana Freda Bloggsa.

Zakładając, że domyślny menedżer kolejek został skonfigurowany dla przykładu z wieloma bazami danych z kolejką o nazwie multDBQ, należy zmniejszyć konto pani Mary Brown o 75 w następujący sposób:

```
AMQSPUT multDBQ
```

Następnie należy nacisnąć klawisz w następującym komunikacie:

```
UPDATE Balance change=-00000075 WHERE Account=00000003
```

W kolejce można umieścić wiele komunikatów:

```
AMQ0XAG0
```

Wpisz nazwę kolejki:

```
multDBQ
```

Następnie drukowany jest zaktualizowany status konta Pani Mary Brown.

### *Przykład procedury obsługi kolejki niedostarczonych komunikatów*

Udostępniono przykładową procedurę obsługi kolejki niedostarczonych komunikatów. Nazwa wersji pliku wykonywalnego to `amqsdlq`. Jeśli program do obsługi niedostarczonych komunikatów ma być inny niż program **RUNMQDLQ**, dostępne jest źródło przykładu, którego można użyć jako podstawy.

Przykład jest podobny do programu obsługi niedostarczonych komunikatów udostępnionego w produkcie, ale informacje o śledzeniu i błędach są inne. Dostępne są dwie zmienne środowiskowe:

**ŚLEDZENIE ODQ_TRACE**
Ustaw wartość YES lub `yes` , aby włączyć śledzenie.

**ODQ_MSG**
Ustaw nazwę pliku zawierającego komunikaty o błędach i komunikaty informacyjne. Udostępniony plik ma nazwę `amqsdlq.msg`.

Zmienne te należy udostępnić w środowisku za pomocą komend **export** lub **set** , w zależności od platformy. Śledzenie jest wyłączany za pomocą komendy **unset** .

Plik komunikatów o błędach, `amqsdlq.msg`, można zmodyfikować, aby dostosować go do własnych wymagań. Przykład umieszcza komunikaty na wyjściu standardowym, **a nie** w pliku dziennika błędów IBM MQ .

Więcej informacji na temat działania programu obsługi niedostarczonych komunikatów oraz sposobu jego uruchamiania zawiera sekcja Przetwarzanie komunikatów w IBM MQ kolejce niedostarczonych komunikatów lub podręcznik *System Management Guide* dla używanej platformy.

### *Przykładowy program listy dystrybucyjnej*

Przykład listy dystrybucyjnej amqsptl0 przedstawia przykład umieszczania komunikatu w kilku kolejkach komunikatów. Jest on oparty na przykładzie MQPUT amqsput0.

## Uruchamianie przykładowej listy dystrybucyjnej amqsptl0

Przykład listy dystrybucyjnej działa w podobny sposób, jak próby typu put.

Przyjmuje on następujące parametry:

- Nazwy kolejek
- Nazwy menedżerów kolejek

Te wartości są wprowadzane jako pary. Na przykład:

```
amqsptl0 queue1 qmanagername1 queue2 qmanagername2
```

Kolejki są otwierane za pomocą komendy MQOPEN, a komunikaty są umieszczane w kolejkach za pomocą komendy MQPUT. Kody przyczyny są zwracane, jeśli którakolwiek z nazw kolejek lub menedżerów kolejek nie została rozpoznana.

Należy pamiętać o zdefiniowaniu kanałów między menedżerami kolejek, aby komunikaty mogły przepływać między nimi. Program przykładowy nie robi tego dla użytkownika.

## Projekt próbki listy dystrybucyjnej

Rekordy komunikatów umieszczania (MQPMR) określają atrybuty komunikatów dla każdego miejsca docelowego. Przykład udostępnia wartości dla parametrów *MsgId* i *CorrelId*, które przesłaniają wartości określone w strukturze MQMD.

Pole *PutMsgRecFields* w strukturze MQPMO wskazuje, które pola są obecne w MQPMR:

```
MQLONG PutMsgRecFields=MQPMRF_MSG_ID + MQPMRF_CORREL_ID;
```

Następnie próba przydziela rekordy odpowiedzi i rekordy obiektów. Rekordy obiektów (MQOR) wymagają co najmniej jednej pary nazw i parzystej liczby nazw, czyli *ObjectName* i *ObjectQMgrName*.

Następny etap obejmuje nawiązanie połączenia z menedżerami kolejek przy użyciu usługi MQCONN. W tym przykładzie podejmowana jest próba nawiązania połączenia z menedżerem kolejek powiązanym z pierwszą kolejką w obiekcie MQOR. Jeśli próba ta nie powiedzie się, rekordy obiektów są kolejno przeglądane. Użytkownik jest informowany, czy nie jest możliwe nawiązanie połączenia z żadnym menedżerem kolejek i zakończenie działania programu.

Kolejki docelowe są otwierane za pomocą komendy MQOPEN, a komunikat jest umieszczany w tych kolejkach za pomocą komendy MQPUT. Wszystkie problemy i niepowodzenia są zgłaszane w rekordach odpowiedzi (MQRR).

Na koniec kolejki docelowe są zamykane za pomocą komendy MQCLOSE, a program rozłącza się z menedżerem kolejek za pomocą komendy MQDISC. Te same rekordy odpowiedzi są używane dla każdego wywołania o wartości *CompCode* i *Reason*.

### *Przykładowe programy Echo*

Przykładowe Programy Echo echo komunikatu przesyłanego z kolejki komunikatów do kolejki odpowiedzi.

Nazwy tych programów można znaleźć w sekcji “Funkcje demonstrowane w przykładowych programach w środowisku wieloplatformowym” na stronie 1092 .

Programy mają być uruchamiane jako programy wyzwalane.

W systemach IBM ii AIX, Linux, and Windows ich jedynym wejściem jest struktura MQTMC2 (komunikat wyzwalacza), która zawiera nazwę kolejki docelowej i menedżera kolejek. Wersja języka COBOL używa domyślnego menedżera kolejek.

IBM i W systemie IBM i, aby proces wyzwalania działał, upewnij się, że przykładowy program Echo, który ma być używany, jest wyzwalany przez komunikaty przychodzące do kolejki SYSTEM.SAMPLE.ECHO. W tym celu należy określić nazwę przykładowego programu Echo, który ma być używany, w polu *ApplId* definicji procesu SYSTEM.SAMPLE.ECHOPROCESS. (W tym celu można użyć komendy CHGMQMPRC; szczegółowe informacje można znaleźć w sekcji Zmiana procesu MQ (CHGMQMPRC)). Przykładowa kolejka ma wyzwalacz typu FIRST, więc jeśli przed uruchomieniem próbki żądania w kolejce znajdują się już komunikaty, próbka Echo nie jest wyzwalana przez wysyłane komunikaty.

Po poprawnym ustawieniu definicji należy najpierw uruchomić zadanie AMQSERV4 w jednym zadaniu, a następnie uruchomić zadanie AMQSREQ4 w innym. Zamiast wartości AMQSERV4można użyć wartości AMQSTRG4 , ale potencjalne opóźnienia wprowadzania zadań mogą ułatwić śledzenie tego, co się dzieje.

Przykładowe programy żądania służą do wysyłania komunikatów do kolejki SYSTEM.SAMPLE.ECHO. Przykładowe programy echo wysyłają komunikat odpowiedzi zawierający dane z komunikatu żądania do kolejki odpowiedzi określonej w komunikacie żądania.

## Projekt programów przykładowych Echo

Program otwiera kolejkę o nazwie określonej w strukturze komunikatu wyzwalacza, która została przekazana podczas uruchamiania. (Dla jasności, jest to nazywane kolejką żądań). Program używa wywołania MQOPEN do otwarcia tej kolejki dla wejścia współużytkowanego.

Program korzysta z wywołania MQGET w celu usunięcia komunikatów z tej kolejki. To wywołanie używa opcji MQGMO_ACCEPT_TRUNCATED_MSG, MQGMO_CONVERT i MQGMO_WAIT z przedziałem czasu oczekiwania wynoszącym 5 sekund. Program testuje deskryptor każdego komunikatu, aby sprawdzić, czy jest to komunikat żądania; jeśli nie, program usuwa komunikat i wyświetla komunikat ostrzegawczy.

Dla każdego wiersza danych wejściowych program odczytuje tekst do buforu i używa wywołania MQPUT1 w celu umieszczenia komunikatu żądania zawierającego tekst tego wiersza w kolejce odpowiedzi.

Jeśli wywołanie MQGET nie powiedzie się, program umieszcza komunikat raportu w kolejce odpowiedzi, ustawiając w polu *Feedback* deskryptora komunikatu kod przyczyny zwrócony przez komendę MQGET.

Jeśli w kolejce żądań nie ma już żadnych komunikatów, program zamyka tę kolejkę i rozłącza się z menedżerem kolejek.

IBM i W systemie IBM iprogram może również odpowiadać na komunikaty wysyłane do kolejki z platform innych niż IBM MQ for IBM i, chociaż w takiej sytuacji nie jest dostarczana żadna próbka. Aby program ECHO działał:

- Napisz program, poprawnie określając parametry **Format**, **Encoding**i **CCSID** , w celu wysyłania komunikatów żądań tekstowych.

  Program ECHO żąda od menedżera kolejek przeprowadzenia konwersji danych komunikatu, jeśli jest to konieczne.
- Podaj wartość CONVERT (*YES) w kanale nadawczym IBM MQ for IBM i , jeśli napisany program nie zapewnia podobnej konwersji dla odpowiedzi.

### *Przykładowe programy Get*

Przykładowe programy pobierające komunikaty z kolejki za pomocą wywołania MQGET.

Nazwy tych programów można znaleźć w sekcji “Funkcje demonstrowane w przykładowych programach w środowisku wieloplatformowym” na stronie 1092 .

## Projekt przykładowego programu Get

Program otwiera kolejkę docelową za pomocą wywołania MQOPEN z opcją MQOO_INPUT_AS_Q_DEF. Jeśli nie można otworzyć kolejki, program wyświetla komunikat o błędzie zawierający kod przyczyny zwrócony przez wywołanie MQOPEN.

Dla każdego komunikatu w kolejce program używa wywołania MQGET do usunięcia komunikatu z kolejki, a następnie wyświetla dane zawarte w komunikacie. Wywołanie MQGET używa opcji MQGMO_WAIT z parametrem *WaitInterval* o wartości 15 sekund, aby program oczekiwał przez ten okres, jeśli w kolejce nie ma komunikatu. Jeśli przed upływem tego okresu nie nadejdzie żaden komunikat, wywołanie nie powiedzie się i zostanie zwrócony kod przyczyny MQRC_NO_MSG_AVAILABLE.

Program demonstruje, w jaki sposób należy wyczyścić pola *MsgId* i *CorrelId* struktury MQMD po każdym wywołaniu MQGET, ponieważ wywołanie to ustawia te pola na wartości zawarte w pobieranym komunikacie. Usunięcie zaznaczenia tych pól oznacza, że kolejne wywołania MQGET pobierają komunikaty w kolejności, w jakiej są przechowywane w kolejce.

Wywołanie MQGET określa bufor o stałej wielkości. Jeśli komunikat jest dłuższy niż ten bufor, wywołanie nie powiedzie się i program zostanie zatrzymany.

Działanie programu jest kontynuowane do momentu, gdy wywołanie MQGET zwróci kod przyczyny MQRC_NO_MSG_AVAILABLE lub wywołanie MQGET nie powiedzie się. Jeśli wywołanie nie powiedzie się, program wyświetli komunikat o błędzie zawierający kod przyczyny.

Następnie program zamyka kolejkę za pomocą wywołania MQCLOSE.

*Uruchamianie przykładów amqsget i amqsgetc*

Każdy z tych programów ma następujące parametry pozycyjne:

1. Nazwa kolejki źródłowej (wymagane)
2. Nazwa menedżera kolejek (opcjonalna)

   Jeśli menedżer kolejek nie jest określony, program **amqsget** nawiązuje połączenie z domyślnym menedżerem kolejek, a program **amqsgetc** nawiązuje połączenie z menedżerem kolejek identyfikowanym przez zmienną środowiskową *MQSERVER* lub plik definicji kanału klienta.
3. Opcje otwierania (opcjonalnie)

   Jeśli opcje otwarcia nie są określone, w przykładzie używana jest wartość 8193, która jest kombinacją tych dwóch opcji:

   - MQOO_INPUT_AS_Q_DEF
   - MQOO_FAIL_IF_QUIESCING,
4. Opcje zamykania (opcjonalnie)

   Jeśli opcje zamykania nie są określone, przykład używa wartości 0, która jest równa MQCO_NONE.

Użyj następujących zmiennych środowiskowych, aby podać referencje używane do uwierzytelniania w menedżerze kolejek:

**ID_UŻYTKOWNIKA MQSAMP_ID**
Ustaw identyfikator użytkownika, który ma być używany do uwierzytelniania połączenia, jeśli do uwierzytelniania w menedżerze kolejek ma być używany identyfikator użytkownika i hasło. Program poprosi o podanie hasła, które ma być dołączone do identyfikatora użytkownika.

**MQSAMP_TOKEN**
Ustaw niepustą wartość, aby podać znacznik uwierzytelniania w celu uwierzytelnienia w menedżerze kolejek. Program pyta o znacznik uwierzytelniania. Znaczniki uwierzytelniania mogą być używane tylko przez przykład **amqsgetc** , który używa powiązań klienta.

Aby uruchomić te programy, wprowadź jedną z następujących komend:

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

gdzie *moja_kolejka* jest nazwą kolejki, z której program będzie pobierał komunikaty, a *nazwa_menedżera_kolejek* jest menedżerem kolejek, do którego należy *moja_kolejka*.

## Używanie amqsget i amqsgetc

Należy zauważyć, że program **amqsget** nawiązuje połączenie lokalne z menedżerem kolejek przy użyciu pamięci współużytkowanej w celu przyłączenia do menedżera kolejek i jako taki może być uruchamiany tylko w systemie, w którym rezyduje menedżer kolejek, podczas gdy program **amqsgetc** nawiązuje połączenie w stylu klienta (nawet w przypadku nawiązywania połączenia z menedżerem kolejek w tym samym systemie).

W przypadku korzystania z produktu **amqsgetc** należy podać szczegóły aplikacji dotyczące rzeczywistego dostępu do menedżera kolejek w odniesieniu do hosta lub adresu IP menedżera kolejek oraz portu nasłuchiwania menedżera kolejek.

Zwykle jest to wykonywane przy użyciu zmiennej środowiskowej **MQSERVER** lub przez zdefiniowanie szczegółów połączenia przy użyciu tabeli definicji kanału klienta, która może być również udostępniana produktowi **amqsgetc** przy użyciu zmiennych środowiskowych. Na przykład patrz sekcja MQCCDTURL.

Przykład użycia programu **MQSERVER**łączącego się lokalnie z menedżerem kolejek, w którym działa program nasłuchujący na porcie 1414 i który używa domyślnego kanału połączenia z serwerem:

```
export MQSERVER="SYSTEM.DEF.SVRCONN/TCP/ localhost(1414)"
```

### *Przykładowe programy o wysokiej dostępności*

Programy przykładowe o wysokiej dostępności w systemach **amqsghac**, **amqsphac**i **amqsmhac** używają zautomatyzowanego ponownego połączenia klienta w celu zademonstrowania odtwarzania po awarii menedżera kolejek. Program **amqsfhac** sprawdza, czy menedżer kolejek korzystający z sieciowej pamięci masowej zachowuje integralność danych po awarii.

Programy **amqsghac**, **amqsphac**i **amqsmhac** są uruchamiane z wiersza komend i mogą być używane w kombinacji w celu zademonstrowania ponownego połączenia po awarii jednej instancji menedżera kolejek z wieloma instancjami.

Alternatywnie można użyć przykładów **amqsghac**, **amqsphac**i **amqsmhac** , aby zademonstrować ponowne połączenie klienta z menedżerami kolejek z pojedynczą instancją, zwykle skonfigurowanymi w grupie menedżerów kolejek.

Aby zachować prosty przykład, który można łatwo skonfigurować, wyświetlane są przykładowe programy łączące się ponownie z menedżerem kolejek z jedną instancją, który jest uruchamiany, zatrzymywany, a następnie restartowany ponownie (patrz sekcja "Konfigurowanie menedżera kolejek i sterowanie nim" na stronie 1127).

Aby sprawdzić integralność systemu plików, należy użyć opcji **amqsfhac** równolegle z opcją **amqmfsck** . Więcej informacji na ten temat zawiera sekcja **amqmfsck** (sprawdzanie systemu plików) i sekcja Weryfikowanie zachowania współużytkowanego systemu plików .

**amqsphac *queueName* [*qMgrNazwa*]**

- **amqsphac** jest aplikacją IBM MQ MQI client . Umieszcza sekwencję komunikatów w kolejce z dwusekundowym opóźnieniem między każdym komunikatem i wyświetla zdarzenia wysłane do jego procedury obsługi zdarzeń.
- Żaden punkt synchronizacji nie jest używany do umieszczania komunikatów w kolejce.
- Można ponownie nawiązać połączenie z dowolnym menedżerem kolejek w tej samej grupie menedżerów kolejek.

**amqsghac *queueName* [*qMgrNazwa*]**

- **amqsghac** jest aplikacją IBM MQ MQI client . Pobiera komunikaty z kolejki i wyświetla zdarzenia wysłane do procedury obsługi zdarzeń.
- Żaden punkt synchronizacji nie jest używany do pobierania komunikatów z kolejki.
- Można ponownie nawiązać połączenie z dowolnym menedżerem kolejek w tej samej grupie menedżerów kolejek.

**amqsmhac -s *sourceQueueNazwa* -t *targetQueueNazwa* [ -m *qMgrNazwa* ] [ -w *waitInterval* ]**

- **amqsmhac** jest aplikacją IBM MQ MQI client . Kopiuje on komunikaty z jednej kolejki do drugiej z domyślnym odstępem czasu oczekiwania wynoszącym 15 minut po ostatnim odebranym komunikacie, zanim program zakończy działanie.
- Komunikaty są kopiowane w ramach punktu synchronizacji.
- Ponowne połączenie można nawiązać tylko z tym samym menedżerem kolejek.

**amqsfhac *QueueManagerNazwa QueueName SideQueueNazwa InTransactionLicznik RepeatCount* (0 | 1 | 2)**

- **amqsfhac** jest aplikacją IBM MQ MQI client . Sprawdza on, czy menedżer kolejek z wieloma instancjami programu IBM MQ korzystający z sieciowej pamięci masowej, takiej jak NAS lub klastrowy system plików, utrzymuje integralność danych. Wykonaj kroki opisane w sekcji Weryfikowanie zachowania współużytkowanego systemu plików, aby uruchomić komendę **amqsfhac** .
- Używa opcji `MQCNO_RECONNECT_Q_MGR` podczas nawiązywania połączenia z menedżerem kolejek produktu *QueueManagerNazwa*. Po przełączeniu menedżera kolejek następuje automatyczne ponowne nawiązanie połączenia.
- Umieszcza trwałe komunikaty *InTransactionCount*RepeatCount* w kolejce *QueueName* , w którym to czasie menedżer kolejek jest przełączany awaryjnie dowolną liczbę razy. Program **amqsfhac**

ponownie nawiązuje połączenie z menedżerem kolejek za każdym razem i kontynuuje działanie. Test ma na celu upewnienie się, że żadne komunikaty nie zostaną utracone.

- Komunikaty *InTransactionCount* są umieszczane w każdej transakcji. Transakcja jest powtarzana *RepeatCount* razy. Jeśli w ramach transakcji wystąpi niepowodzenie, program **amqsfhac** wycofa zmiany i wprowadzi ponownie transakcję, gdy program **amqsfhac** ponownie nawiąże połączenie z menedżerem kolejek.
- Umieszcza również komunikaty w kolejce bocznikowania *SideQueueName*. Używa ona parametru *SideQueueName* do sprawdzenia, czy wszystkie komunikaty zostały pomyślnie zatwierdzone lub wycofane z kolejki *QueueName* . Jeśli wykryje niespójność, zapisuje komunikat o błędzie.
- Zmiana poziomu śledzenia danych wyjściowych z **amqsfhac** przez ustawienie ostatniego parametru na `(0|1|2)`.

  **0**
  Najmniejszy wynik.

  **1**
  Wyjście pośredniczące.

  **2**
  Większość danych wyjściowych.

## Konfigurowanie połączenia klienckiego

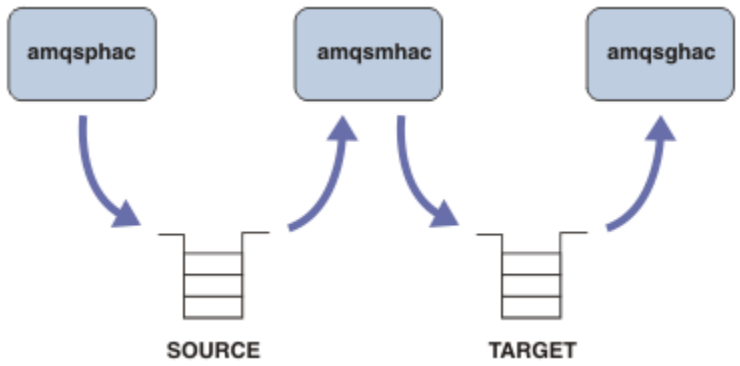
Aby uruchomić przykłady, należy skonfigurować kanał połączenia klienta i serwera. Procedura weryfikacji klienta wyjaśnia sposób konfigurowania środowiska testowego klienta.

Alternatywnie można użyć konfiguracji podanej w poniższym przykładzie.

### Przykład użycia amqsghac, amqsphaci amqsmhac

W tym przykładzie przedstawiono klienty z możliwością ponownego nawiązania połączenia przy użyciu menedżera kolejek z pojedynczą instancją.

Komunikaty są umieszczane w kolejce SOURCE przez **amqsphac**, przesyłane do TARGET przez **amqsmhac**i pobierane z TARGET przez **amqsghac** ; zawiera sekcja Rysunek 129 na stronie 1126.




*Rysunek 129. Przykłady klienta z możliwością ponownego połączenia*

Wykonaj poniższe kroki, aby uruchomić przykłady.

1. Utwórz plik `hasamples.tst` zawierający komendy:

```
DEFINE QLOCAL(SOURCE) REPLACE
DEFINE QLOCAL(TARGET) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)  TRPTYPE(TCP) +
       MCAUSER(MUSR_MQADMIN) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
       CONNAME('LOCALHOST(2345)') QMNAME(QM1) REPLACE
ALTER  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) +
       PORT(2345)
```

```
START  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
START  CHANNEL(CHANNEL1)
```

2. Wpisz następujące komendy w wierszu komend:
   a. `crtmqm QM1`
   b. `strmqm QM1`
   c. `runmqsc QM1 < hasamples.tst`
3. Ustaw zmienną środowiskową **MQCHLLIB** na ścieżkę do pliku definicji kanału klienta `AMQCLCHL.TAB` , na przykład `SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc`.
4. Otwórz trzy nowe okna z ustawionym **MQCHLLIB** ; na przykład w systemie Windowswpisz **start** trzy razy w poprzednim wierszu komend, uruchamiając każdy program w jednym z okien. Patrz krok “5” na stronie 1128 w sekcji “Konfigurowanie menedżera kolejek i sterowanie nim” na stronie 1127).
5. Wpisz komendę `endmqm -r -p QM1` , aby zatrzymać menedżer kolejek, a następnie zezwolić klientom na ponowne nawiązanie połączenia.
6. Wpisz komendę `strmqm QM1` , aby zrestartować menedżer kolejek.

Wyniki uruchomienia przykładów **amqsghac**, **amqsphac**i **amqsmhac** w systemie Windows zostały przedstawione w poniższych przykładach.

### Konfigurowanie menedżera kolejek i sterowanie nim

1. Utwórz menedżer kolejek.

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

   Zapamiętaj katalog danych, aby później ustawić zmienną **MQCHLLIB** .
2. Uruchom menedżer kolejek.

```
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.
```

3. Utwórz kolejki i kanały, zmodyfikuj port nasłuchiwania oraz uruchom program nasłuchujący i kanał.

```
C:\> runmqsc QM1 < hasamples.tst

5724-H72 (C) Copyright IBM Corp. 1994, 2024.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.

     1 : DEFINE QLOCAL(SOURCE) REPLACE
AMQ8006: 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. Udostępnij klientom tabelę kanałów klienta.

   Użyj katalogu danych zwróconego przez komendę **crtmqm** w kroku “1” na stronie 1127i dodaj do niego katalog `@ipcc` , aby ustawić zmienną **MQCHLLIB** .

```
C:\> SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc
```

5. Uruchamianie przykładowych programów w innych oknach

```
C:\> start amqsphac SOURCE QM1
C:\> start amqsmhac -s SOURCE -t TARGET -m QM1
C:\> start amqsghac TARGET QM1
```

6. Zakończ działanie menedżera kolejek i zrestartuj go ponownie.

```
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.
```

**amqsphac**

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

**amqsmhac**

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

**amqsghac**

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

```
message Message 3
message Message 4
message Message 5
```

**Zadania pokrewne**

Weryfikowanie zachowania współużytkowanego systemu plików

**Odsyłacze pokrewne**

**amqmfsck** (sprawdzanie systemu plików)

## *Przykładowe programy do sprawdzania*

Przykładowe programy zapytania sprawdzają niektóre atrybuty kolejki przy użyciu wywołania MQINQ.

Nazwy tych programów można znaleźć w sekcji “Funkcje demonstrowane w przykładowych programach w środowisku wieloplatformowym” na stronie 1092 .

Programy te są przeznaczone do uruchamiania jako programy wyzwalane, dlatego ich jedynym wejściem jest struktura MQTMC2 (komunikat wyzwalacza) dla systemu IBM MQ for Multiplatforms. Ta struktura zawiera nazwę kolejki docelowej z atrybutami, których dotyczy zapytanie. W wersji C używana jest również nazwa menedżera kolejek. Wersja języka COBOL używa domyślnego menedżera kolejek.

Aby proces wyzwalania działał, upewnij się, że program przykładowy Inquire, który ma być używany, jest wyzwalany przez komunikaty przychodzące do kolejki SYSTEM.SAMPLE.INQ. W tym celu w polu *ApplicId* definicji procesu SYSTEM.SAMPLE.INQPROCESS. IBM i W systemie IBM imożna w tym celu użyć komendy CHGMQMPRC. Szczegółowe informacje na ten temat zawiera sekcja Zmiana procesu MQ (CHGMQMPRC). Przykładowa kolejka ma wyzwalacz typu FIRST. Jeśli przed uruchomieniem próbki żądania w kolejce znajdują się już komunikaty, próbka zapytania nie jest wyzwalana przez wysyłane komunikaty.

Po poprawnym ustawieniu definicji:

- ALW W systemie AIX, Linux, and Windowsuruchom program **runmqtrm** w jednej sesji, a następnie uruchom program `amqsreq` w innej.
- IBM i W systemie IBM iuruchom program AMQSERV4 w jednej sesji, a następnie program AMQSREQ4 w innej sesji. Zamiast wartości AMQSERV4można użyć wartości AMQSTRG4 , ale potencjalne opóźnienia wprowadzania zadań mogą ułatwić śledzenie tego, co się dzieje.

Przykładowe programy żądania służą do wysyłania komunikatów żądań, z których każdy zawiera tylko nazwę kolejki, do kolejki SYSTEM.SAMPLE.INQ. Dla każdego komunikatu żądania programy przykładowe Inquire wysyłają komunikat odpowiedzi zawierający informacje o kolejce określonej w komunikacie żądania. Odpowiedzi są wysyłane do kolejki odpowiedzi określonej w komunikacie żądania.

IBM i W systemie IBM i, jeśli przykładowy podzbiór zbioru wejściowego to QMQMSAMP.AMQSDATA(INQ) jest używana, ostatnia kolejka o podanej nazwie nie istnieje, dlatego przykład zwraca komunikat raportu z kodem przyczyny niepowodzenia.

## Projekt programu przykładowego Inquire

Program otwiera kolejkę o nazwie określonej w strukturze komunikatu wyzwalacza, która została przekazana podczas uruchamiania. (Dla jasności będzie to *kolejka żądań*). Program używa wywołania MQOPEN do otwarcia tej kolejki dla wejścia współużytkowanego.

Program korzysta z wywołania MQGET w celu usunięcia komunikatów z tej kolejki. To wywołanie używa opcji MQGMO_ACCEPT_TRUNCATED_MSG i MQGMO_WAIT z odstępem czasu oczekiwania wynoszącym 5 sekund. Program testuje deskryptor każdego komunikatu, aby sprawdzić, czy jest to komunikat żądania; jeśli nie, program usuwa komunikat i wyświetla komunikat ostrzegawczy.

Dla każdego komunikatu żądania usuniętego z kolejki żądań program odczytuje nazwę kolejki (która będzie zwana *kolejką docelową* ). i otwiera tę kolejkę za pomocą wywołania MQOPEN z opcją MQOO_INQ. Następnie program używa wywołania MQINQ do uzyskania informacji o wartościach atrybutów *InhibitGet*, **CurrentQDepth**i **OpenInputCount** kolejki docelowej.

Jeśli wywołanie MQINQ powiedzie się, program użyje wywołania MQPUT1 do umieszczenia komunikatu odpowiedzi w kolejce odpowiedzi. Ten komunikat zawiera wartości trzech atrybutów.

Jeśli wywołanie MQOPEN lub MQINQ nie powiedzie się, program użyje wywołania MQPUT1 do umieszczenia komunikatu raportu w kolejce odpowiedzi. W polu *Feedback* deskryptora komunikatu tego raportu znajduje się kod przyczyny zwrócony przez wywołanie MQOPEN lub MQINQ, w zależności od tego, które wywołanie nie powiodło się.

Po wywołaniu MQINQ program zamyka kolejkę docelową za pomocą wywołania MQCLOSE.

Jeśli w kolejce żądań nie ma już żadnych komunikatów, program zamyka tę kolejkę i rozłącza się z menedżerem kolejek.

### *Właściwości zapytania programu przykładowego uchwytu komunikatu*

AMQSIQMA jest przykładowym programem w języku C służącym do sprawdzania właściwości uchwytu komunikatu z kolejki komunikatów i jest przykładem użycia wywołania API MQINQMP.

Ten przykład tworzy uchwyt komunikatu i umieszcza go w polu MsgHandle w strukturze MQGMO. Następnie w przykładzie pobierany jest jeden komunikat, a następnie wykonywane jest zapytanie i drukowane są wszystkie właściwości, które zostały zapełnione uchwytem komunikatu.

```
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
```

### *Przykładowe programy publikowania/subskrypcji*

Przykładowe programy publikowania/subskrypcji demonstrują użycie funkcji publikowania i subskrypcji w produkcie IBM MQ.

Istnieją trzy przykładowe programy w języku C ilustrujące sposób programowania w interfejsie publikowania/subskrypcji produktu IBM MQ . Istnieje kilka przykładów w języku C, które korzystają ze starszych interfejsów, oraz przykłady w języku Java . Przykłady produktu Java używają interfejsu publikowania/subskrypcji produktu IBM MQ w pliku com.ibm.mq.jar oraz interfejsu publikowania/subskrypcji produktu JMS w pliku com.ibm.mqjms. Przykłady JMS nie zostały omówione w tym temacie.

## C

Znajdź przykładowy publikator `amqspub` w folderze przykładów C . Należy go uruchomić z dowolną nazwą tematu, która ma być używana jako pierwszy parametr, po której następuje opcjonalna nazwa menedżera kolejek. Na przykład `amqspub mytopic QM3` . Istnieje również wersja klienta o nazwie `amqspubc`. Jeśli zostanie wybrana opcja uruchomienia wersji klienta, najpierw należy zapoznać się z sekcją “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym” na stronie 1101 , aby uzyskać szczegółowe informacje.

Publikator nawiązuje połączenie z domyślnym menedżerem kolejek i odpowiada danymi wyjściowymi `target topic is mytopic` . Każdy wiersz wprowadzony w tym oknie, począwszy od teraz, jest publikowany w pliku `mytopic` .

Otwórz inne okno komend w tym samym katalogu i uruchom program subskrybenta `amqssub`, podając tę samą nazwę tematu i opcjonalną nazwę menedżera kolejek. Na przykład `amqssub mytopic QM3` .

Subskrybent odpowiada danymi wyjściowymi `Calling MQGET : 30 seconds wait time` . Od tej chwili wiersze wpisane w publikatorze są wyświetlane w danych wyjściowych subskrybenta.

Uruchom innego subskrybenta w innym oknie komend i obserwuj, jak obaj subskrybenci otrzymują publikacje.

Pełną dokumentację parametrów, w tym opcje ustawiania, można znaleźć w przykładowym kodzie źródłowym. Wartości pola opcji subskrybenta zostały opisane w następującym temacie: Opcje (MQLONG).

Istnieje inny przykładowy subskrybent `amqssbx`, który oferuje dodatkowe opcje subskrypcji jako przełączniki wiersza komend.

Wpisz komendę `amqssbx -d mysub -t mytopic -k`, aby wywołać subskrybent przy użyciu trwałych subskrypcji, które są zachowywane po zakończeniu działania subskrybenta.

Przetestuj subskrypcję, publikując inny element przy użyciu publikatora. Poczekaj 30 sekund na zakończenie działania subskrybenta. Opublikuj więcej elementów w tym samym temacie. Zrestartuj subskrybent. Ostatni element opublikowany w czasie, gdy subskrybent nie był uruchomiony, jest wyświetlany przez subskrybent natychmiast po jego zrestartowaniu.

## Starsza wersja języka C

Istnieje dodatkowy zestaw przykładów w języku C, który demonstruje kolejkowane komendy. Niektóre z tych przykładów zostały pierwotnie dostarczone jako część pakietu MQ0C Supportpac. Możliwości przedstawione w przykładach są w pełni obsługiwane ze względu na kompatybilność.

Nie zaleca się używania umieszczonego w kolejce interfejsu komend. Jest ona znacznie bardziej złożona niż interfejs API publikowania/subskrypcji i nie ma żadnego ważnego powodu funkcjonalnego, aby programować złożone komendy w kolejce. Jednak może się okazać, że podejście kolejkowane jest bardziej odpowiednie, na przykład dlatego, że interfejs jest już używany, lub dlatego, że środowisko programistyczne ułatwia budowanie złożonego komunikatu i wywoływanie ogólnego wywołania MQPUT zamiast tworzenia różnych wywołań MQSUB.

Dodatkowe przykłady znajdują się w podkatalogu `pubsub` w folderze `samples`.

Istnieje sześć typów przykładów wymienionych w sekcji Tabela 163 na stronie 1131.

*Tabela 163. Kategorie wcześniejszych przykładowych programów w języku C publikowania/subskrypcji*

| Kategoria | Programy | Komentarze |
|---|---|---|
| RFH1 | `amqssr1a.c`<br>`amqspr1a.c` | Prosty przykład publikowania/subskrypcji zbudowany przy użyciu komunikatów w formacie RFH1. |
| RFH2 | `amqssr2a.c`<br>`amqspr2a.c` | Prosty przykład publikowania/subskrypcji zbudowany przy użyciu komunikatów w formacie RFH2. |
| Przykłady MQAI | `amqsppca.c`<br>`amqsspca.c` | Prosty przykład publikowania/subskrypcji zbudowany przy użyciu komend PCF i interfejsu komend MQAI. |
| MA0C Usługa wyników przy użyciu RFH1 | `amqsgama.c`<br>`amqsresa.c` | Usługa wyników zbudowana przy użyciu nagłówków RFH1<br>1. Wymaga kolejek zdefiniowanych w `amqsgama.tst` i `amqsresa.tst`<br>2. `amqsresa` należy uruchomić przed `amqsgama` |
| MA0C Usługa wyników przy użyciu RFH2 | `amqsgr2a.c`<br>`amqsrr2a.c` | Usługa wyników zbudowana przy użyciu nagłówków RFH2<br>1. Wymaga kolejek zdefiniowanych w `amqsgama.tst` i `amqsresa.tst`<br>2. `amqsresa` należy uruchomić przed `amqsgama` |
| Przykład wyjścia routingu z publikowania/subskrybowania | `amqspsra.c` | Demonstruje sposób zmiany miejsca docelowego kolejki lub menedżera kolejek dla komunikatu publikowania/subskrypcji w wyjściu routingu. |

## Przykładowy program dla systemu Java

Java Przykład `MQPubSubApiSample.java` łączy publikator i subskrybentów w jeden program. Jego źródłowe i skompilowane pliki klas znajdują się w folderze przykładów `wmqjava` .

Jeśli zostanie wybrana opcja uruchomienia w trybie klienta, najpierw należy zapoznać się ze szczegółami w sekcji "Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym" na stronie 1101 .

Uruchom przykład z wiersza komend za pomocą komendy Java , jeśli skonfigurowano środowisko Java . Przykład można również uruchomić z obszaru roboczego Eclipse programu IBM MQ Explorer, w którym jest już skonfigurowane programistyczne środowisko robocze Java .

Konieczne może być zmodyfikowanie niektórych właściwości programu przykładowego w celu jego uruchomienia. W tym celu należy podać parametry dla maszyny JVM lub zmodyfikować źródło.

Instrukcje w sekcji "Uruchamianie przykładu MQPubSubApiSample Java" na stronie 1132 przedstawiają sposób uruchamiania przykładu z obszaru roboczego Eclipse .

*Uruchamianie przykładu MQPubSubApiSample Java*
Sposób uruchamiania komendy MQPubSubApiSample za pomocą narzędzi programistycznych Java z platformy Eclipse .

## Zanim rozpoczniesz

Otwórz środowisko robocze Eclipse . Utwórz nowy katalog obszaru roboczego i wybierz go. Zamknij okno powitalne.

Przed uruchomieniem jako klient wykonaj kroki opisane w sekcji "Konfigurowanie menedżera kolejek w celu akceptowania połączeń klienckich w systemie wieloplatformowym" na stronie 1101 .

## O tym zadaniu

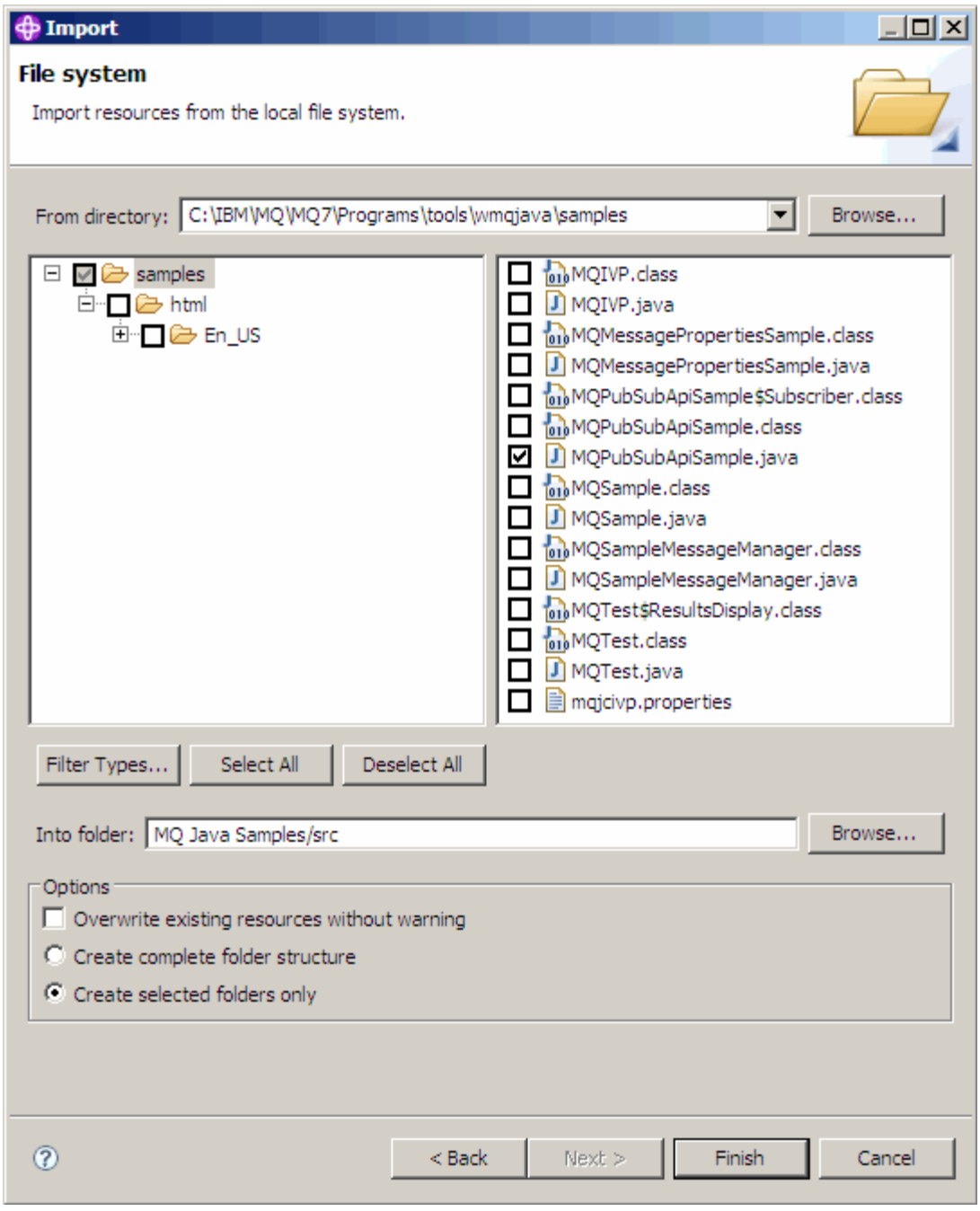
Przykładowy program Java publish/subscribe to program IBM MQ MQI client Java . Przykład jest uruchamiany bez modyfikacji przy użyciu domyślnego menedżera kolejek, który nasłuchuje na porcie 1414. Zadanie opisuje ten prosty przypadek i wskazuje ogólnie, w jaki sposób należy podać parametry i zmodyfikować przykład, aby dostosować go do różnych konfiguracji produktu IBM MQ . Przykład został zilustrowany w systemie Windows. Ścieżki do plików będą się różnić na innych platformach.

## Procedura

1. Importowanie przykładowych programów Java
   a) W środowisku roboczym kliknij opcję **Okna** > **Otwórz perspektywę** > **Inne** > **Java** , a następnie kliknij przycisk **OK**.
   b) Przejdź do widoku **Eksplorator pakietów** .
   c) Kliknij prawym przyciskiem myszy biały obszar w widoku **Eksplorator pakietów** . Kliknij opcję **Nowy** > **Java projekt**.
   d) W polu **`Project name`** wpisz `MQ Java Samples`. Kliknij przycisk **Dalej**.
   e) Na panelu **`Java Settings`** przejdź do karty **Biblioteki** .
   f) Kliknij opcję **Dodaj zewnętrzne pliki JAR**.
   g) Przejdź do katalogu *MQ_INSTALLATION_PATH* `\java\lib` , gdzie *MQ_INSTALLATION_PATH* jest folderem instalacyjnym IBM MQ , a następnie wybierz opcje `com.ibm.mq.jar` i `com.ibm.mq.jmqi.jar`
   h) Kliknij przycisk **Otwórz** > **Zakończ.**
   i) Kliknij prawym przyciskiem myszy plik `src` w widoku **Eksplorator pakietów** .
   j) Wybierz **Importuj ...** > **Ogólne** > **System plików** > **Następny** > **Przeglądaj**... i przejdź do ścieżki *MQ_INSTALLATION_PATH* `\tools\wmqjava\samples` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym IBM MQ .

k) Na panelu **Import** Rysunek 130 na stronie 1133kliknij ikonę `samples` (nie zaznaczaj pola wyboru).

l) Wybierz opcję `MQPubSubApiSample.java`. Pole **`Into folder`** powinno zawierać łańcuch `MQ Java Samples/src`. Kliknij przycisk **Zakończ**.




*Rysunek 130. Import systemu plików*

2. Uruchom przykładowy program publikowania/subskrypcji.

   Istnieją dwa sposoby uruchamiania programu, w zależności od tego, czy konieczna jest zmiana parametrów domyślnych.

   - Pierwszy wybór powoduje uruchomienie programu bez wprowadzania żadnych zmian:
     - W menu głównym obszaru roboczego rozwiń folder `src` . Kliknij prawym przyciskiem myszy plik **MQPubSubApiSample.java Uruchom jako** > **1. Java Aplikacja**

- Drugi wybór powoduje uruchomienie programu z parametrami lub ze zmodyfikowanym kodem źródłowym dla danego środowiska:
  - Otwórz plik `MQPubSubApiSample.java` i przestudiować konstruktor `MQPubSubApiSample` .
  - Zmodyfikuj atrybuty programu.

    Atrybuty te można modyfikować za pomocą przełącznika `-D` wirtualnej maszyny języka Java lub przez podanie wartości domyślnej dla właściwości System p, edytując kod źródłowy.

    - topicObject
    - QueueManagerName
    - subscriberCount

    Te atrybuty można zmieniać tylko przez edycję kodu źródłowego w konstruktorze.

    - nazwa hosta
    - Port
    - kanał

    Aby ustawić właściwości System p, należy zakodować wartość domyślną w akcesorium, na przykład:

    ```
    queueManagerName = System.getProperty("com.ibm.mq.pubSubSample.queueManagerName",
    "QM3");
    ```

    Można również podać ten parametr dla maszyny JVM za pomocą opcji `-D` , jak pokazano w następujących krokach:

  a. Skopiuj pełną nazwę pliku System.Property , który ma zostać ustawiony, na przykład: `com.ibm.mq.pubSubSample.queueManagerName`.
  b. W obszarze roboczym kliknij prawym przyciskiem myszy opcję **Uruchom** > **Otwórz okno dialogowe uruchamiania**. Kliknij dwukrotnie opcję Java Aplikacja na karcie **Utwórz, zarządzaj i uruchamiaj aplikacje** , a następnie kliknij kartę **(x) = Argumenty** .
  c. W panelu **VM arguments:** (Argumenty maszyny VM) wpisz `-D` i wklej nazwę System.property , `com.ibm.mq.pubSubSample.queueManagerName`, po której następuje `=QM3`. Kliknij przycisk **Zastosuj** > **Uruchom**.
  d. Dodaj kolejne argumenty w postaci listy rozdzielanej przecinkami lub jako dodatkowe wiersze w panelu, bez przecinków.

     Na przykład: `-Dcom.ibm.mq.pubSubSample.queueManagerName=QM3`, `-Dcom.ibm.mq.pubSubSample.subscriberCount=6`.

### *Przykładowy program wyjścia publikowania*

AMQSPSE0 to przykładowy program w języku C służący do przechwytywania publikacji przed jej dostarczeniem do subskrybenta. Wyjście może na przykład zmienić nagłówki komunikatów, ładunek lub miejsce docelowe albo uniemożliwić opublikowanie komunikatu w subskrybencie.

Aby uruchomić przykład, wykonaj następujące czynności:

1. Skonfiguruj menedżer kolejek:
   - Linux AIX W systemach AIX and Linux do pliku `qm.ini` należy dodać sekcję podobną do następującej:

     ```
     PublishSubscribe:
     PublishExitPath=Module
     PublishExitFunction=EntryPoint
     ```

     gdzie moduł to *MQ_INSTALLATION_PATH*`/samp/bin/amqspse`. *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

- Windows W systemie Windows należy ustawić równoważne atrybuty w rejestrze.

2. Upewnij się, że moduł jest dostępny dla IBM MQ.
3. Zrestartuj menedżer kolejek, aby pobrać konfigurację.
4. W procesie aplikacji, która ma być śledzona, opisz miejsce, w którym powinny być zapisywane pliki śledzenia. Na przykład:
   - Linux AIX W systemach AIX and Linux upewnij się, że katalog `/var/mqm/trace` istnieje i wyeksportuj zmienną środowiskową **MQPSE_TRACE_LOGFILE** :

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

   - Windows W systemie Windowsupewnij się, że katalog `C:\temp` istnieje, i ustaw zmienną środowiskową **MQPSE_TRACE_LOGFILE** :

     ```
     set MQPSE_TRACE_LOGFILE=C:\temp\PubTrace
     ```

### *Przykładowe programy Put*

Przykładowe programy wstawiające komunikaty do kolejki za pomocą wywołania MQPUT.

Nazwy tych programów można znaleźć w sekcji “Funkcje demonstrowane w przykładowych programach w środowisku wieloplatformowym” na stronie 1092 .

## Projekt programu przykładowego Put

Program używa wywołania MQOPEN z opcją MQOO_OUTPUT do otwarcia kolejki docelowej na potrzeby umieszczania komunikatów.

Jeśli nie można otworzyć kolejki, program wyświetla komunikat o błędzie zawierający kod przyczyny zwrócony przez wywołanie MQOPEN. Aby zachować prostotę programu, w tym przypadku i w kolejnych wywołaniach MQI program używa wartości domyślnych dla wielu opcji.

Dla każdego wiersza danych wejściowych program odczytuje tekst do buforu i używa wywołania MQPUT do utworzenia komunikatu datagramu zawierającego tekst tego wiersza. Działanie programu jest kontynuowane do momentu, gdy zostanie osiągnięty koniec wejścia lub wywołanie MQPUT nie powiedzie się. Jeśli program osiągnie koniec danych wejściowych, zamyka kolejkę za pomocą wywołania MQCLOSE.

*Uruchamianie przykładowych programów Put*

## Uruchamianie przykładów amqsput i amqsputc

ALW

Przykład **amqqsput** jest programem do umieszczania komunikatów przy użyciu powiązań lokalnych, a przykład **amqsputc** jest programem do umieszczania komunikatów przy użyciu powiązań klienta. Każdy z tych programów ma następujące parametry pozycyjne:

1. Nazwa kolejki docelowej (wymagane)
2. Nazwa menedżera kolejek (opcjonalna)

   Jeśli menedżer kolejek nie jest określony, program **amqsput** nawiązuje połączenie z domyślnym menedżerem kolejek, a program **amqsputc** nawiązuje połączenie z menedżerem kolejek identyfikowanym przez zmienną środowiskową *MQSERVER* lub plik definicji kanału klienta.
3. Opcje otwierania (opcjonalnie)

   Jeśli opcje otwarcia nie są określone, w przykładzie używana jest wartość 8208, która jest kombinacją tych dwóch opcji:

   - MQOO_OUTPUT

- MQOO_FAIL_IF_QUIESCING,

4. Opcje zamykania (opcjonalnie)

   Jeśli opcje zamykania nie są określone, przykład używa wartości 0, która jest równa MQCO_NONE.

5. Nazwa docelowego menedżera kolejek (opcjonalna)

   Jeśli docelowy menedżer kolejek nie zostanie określony, pole `ObjectQMgrName` w MQOD pozostanie puste.

6. Nazwa kolejki dynamicznej (opcjonalna)

   Jeśli nazwa kolejki dynamicznej nie zostanie określona, pole `DynamicQName` w programie MQOD pozostanie puste.

Użyj następujących zmiennych środowiskowych, aby podać referencje używane do uwierzytelniania w menedżerze kolejek:

**ID_UŻYTKOWNIKA MQSAMP_ID**
Ustaw identyfikator użytkownika, który ma być używany do uwierzytelniania połączenia, jeśli do uwierzytelniania w menedżerze kolejek ma być używany identyfikator użytkownika i hasło. Program poprosi o podanie hasła, które ma być dołączone do identyfikatora użytkownika.

V 9.3.4 Linux AIX **MQSAMP_TOKEN**
Ustaw niepustą wartość, aby podać znacznik uwierzytelniania w celu uwierzytelnienia w menedżerze kolejek. Program pyta o znacznik uwierzytelniania. Znaczniki uwierzytelniania mogą być używane tylko przez przykład **amqsputc** , który używa powiązań klienta.

Aby uruchomić te programy, wprowadź jedną z następujących komend:

- `amqsput` *`myqueue qmanagername`*
- `amqsputc` *`myqueue qmanagername`*

gdzie *moja_kolejka* jest nazwą kolejki, w której zostaną umieszczone komunikaty, a *qmanagername* jest menedżerem kolejek, do którego należy *moja_kolejka*.

## Uruchamianie przykładu amq0put

ALW

Wersja języka COBOL nie ma żadnych parametrów. Zostanie nawiązane połączenie z domyślnym menedżerem kolejek, a po jego uruchomieniu zostanie wyświetlone zapytanie:

```
Please enter the name of the target queue
```

Pobiera on dane wejściowe z kolejki StdIn i dodaje każdy wiersz danych wejściowych do kolejki docelowej. Pusty wiersz wskazuje, że nie ma więcej danych.

## Uruchamianie przykładu C programu AMQSPUT4 ( IBM i)

IBM i

Program w języku C AMQSPUT4, dostępny tylko dla platformy IBM i , tworzy komunikaty, odczytując dane z podzbioru zbioru źródłowego.

Podczas uruchamiania programu należy określić nazwę pliku jako parametr. Struktura pliku musi być następująca:

```
queue name
text of message 1
text of message 2
:
text of message n
blank line
```

Przykład danych wejściowych dla próbek umieszczania jest dostarczany w bibliotece QMQMSAMP file AMQSDATA member PUT.

**Uwaga:** Należy pamiętać, że w nazwach kolejek rozróżniana jest wielkość liter. Wszystkie kolejki utworzone przez przykładowy program tworzenia plików AMQSAMP4 mają nazwy utworzone wielkimi literami.

Program w języku C umieszcza komunikaty w kolejce o nazwie określonej w pierwszym wierszu pliku. Można użyć dostarczonej kolejki SYSTEM.SAMPLE.LOCAL. Program umieszcza tekst każdego z poniższych wierszy pliku w oddzielnych komunikatach datagramu i zatrzymuje się, gdy odczyta pusty wiersz na końcu pliku.

Przy użyciu przykładowego pliku danych komenda wygląda następująco:

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

## Uruchamianie przykładu COBOL AMQ0PUT4 ( IBM i)

IBM i

Program w języku COBOL AMQ0PUT4, dostępny tylko na platformie IBM i , tworzy komunikaty, akceptując dane z klawiatury.

Aby uruchomić program, wywołaj program i podaj nazwę kolejki docelowej jako parametr programu. Program przyjmuje dane wejściowe z klawiatury do buforu i tworzy komunikat datagramu dla każdego wiersza tekstu. Program zostanie zatrzymany po wpisaniu pustego wiersza na klawiaturze.

### *Przykładowe programy komunikatów referencyjnych*

Przykłady komunikatów odniesienia umożliwiają przesyłanie dużych obiektów z jednego węzła do innego (zwykle w różnych systemach) bez konieczności zapisywania obiektu w kolejkach IBM MQ w węźle źródłowym lub docelowym.

Udostępniono zestaw przykładowych programów demonstrujących, w jaki sposób komunikaty odniesienia mogą być umieszczane w kolejce, odbierane przez wyjścia komunikatów i pobierane z kolejki. Programy przykładowe używają komunikatów odniesienia do przenoszenia plików. Jeśli chcesz przenieść inne obiekty, takie jak bazy danych, lub jeśli chcesz wykonać sprawdzenia bezpieczeństwa, zdefiniuj własne wyjście, w oparciu o przykład, amqsxrm.

Wersja przykładowego programu obsługi wyjścia komunikatu odniesienia, który ma być używany, zależy od platformy, na której działa kanał:

- Na wszystkich platformach należy użyć komendy amqsxrma na końcu wysyłania.
- Jeśli odbiornik działa na dowolnej platformie z wyjątkiem platformy IBM i, należy użyć komendy amqsxrma na odbierającym końcu.
- IBM i Jeśli dziennik działa w systemie IBM i, użyj komendy amqsxrm4.

.

IBM i *Uwagi dla użytkowników programu IBM i*

Aby pobrać komunikat odniesienia przy użyciu przykładowego wyjścia komunikatu, należy określić plik w głównym systemie plików IFS lub w dowolnym podkatalogu, aby można było utworzyć plik strumieniowy.

Przykładowe wyjście komunikatu w systemie IBM i tworzy plik, przekształca dane do formatu EBCDIC i ustawia stronę kodową na stronę kodową systemu. Następnie można skopiować ten plik do biblioteki QSYS.LIB za pomocą komendy CPYFRMSTMF. Na przykład:

```
CPYFRMSTMF FROMSTMF('JANEP/TEST.TXT')
TOMBR('qsys.lib.janep.lib/test.fie/test.mbr') MBROPT(*REPLACE)
CVTDTA(*NONE)
```

Komenda CPYFRMSTMF nie tworzy zbioru. Należy go utworzyć przed uruchomieniem tej komendy.

Jeśli plik jest wysyłany z biblioteki QSYS.LIB, w przykładach nie są wymagane żadne zmiany. W przypadku każdego innego systemu plików upewnij się, że identyfikator CCSID określony w polu CodedCharSetId w strukturze MQRMH jest zgodny z wysyłanymi danymi masowymi.

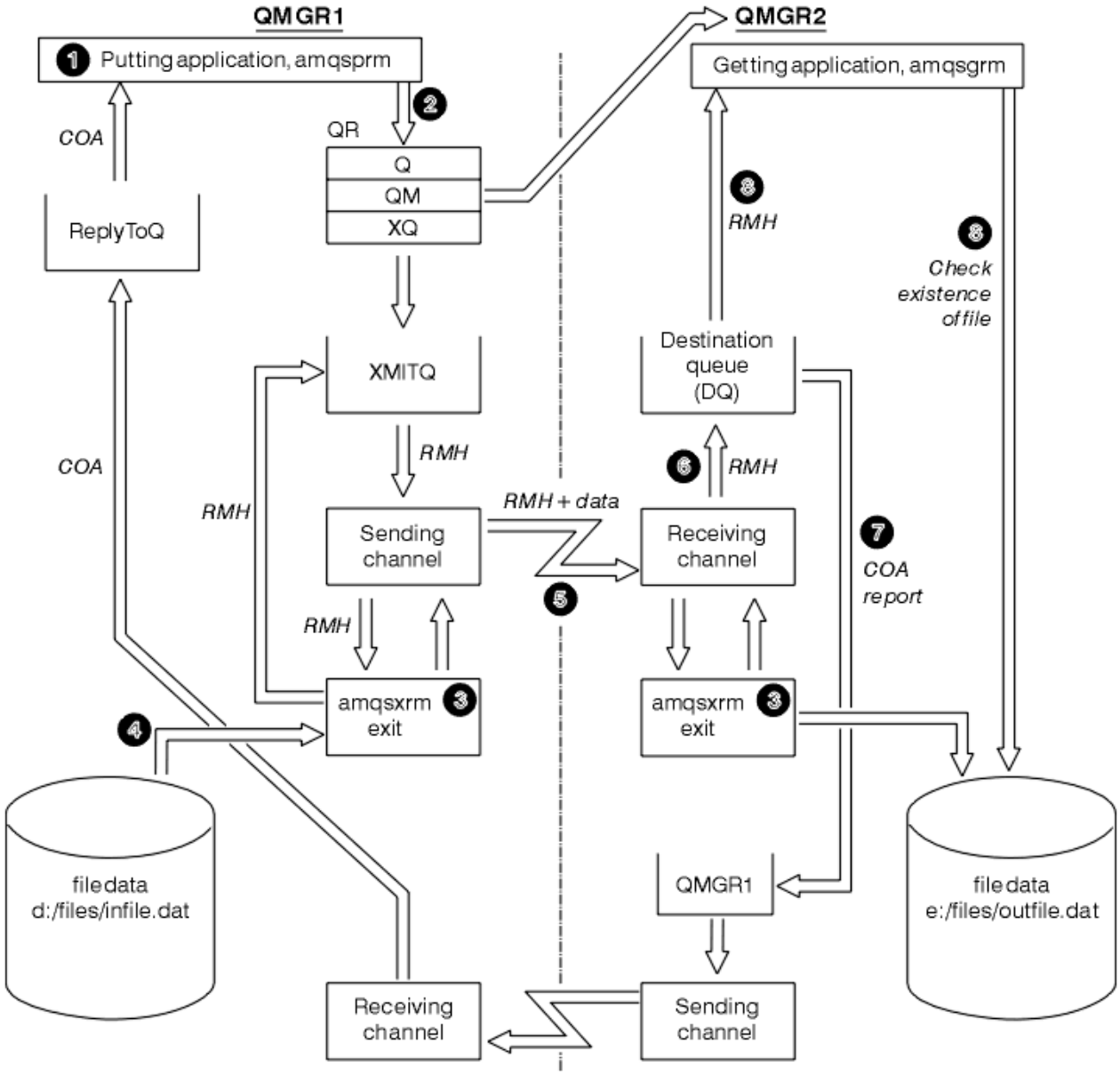
Jeśli używany jest zintegrowany system plików, należy utworzyć moduły programu z ustawioną opcją SYSIFCOPT (*IFSIO). Aby przenieść bazę danych lub pliki rekordów o stałej długości, należy zdefiniować własne wyjście na podstawie dostarczonego przykładu AMQSXRM4.

Zalecaną metodą przesyłania zbioru bazy danych jest przekształcenie go do struktury IFS za pomocą komendy CPYTOSTMF, a następnie wysłanie komunikatu referencyjnego przyłączającego zbiór IFS. Jeśli zbiór bazy danych ma być przesyłany z systemu plików IFS, ale nie ma być przekształcany w strukturę IFS, należy podać nazwę podzbioru. Jeśli zostanie wybrana ta metoda, integralność danych nie będzie zagwarantowana.

Multi *Uruchamianie przykładowych komunikatów referencyjnych*

W tym przykładzie przedstawiono sposób uruchomienia przykładowej aplikacji komunikatu referencyjnego AMQSPRM w systemie AIX, Linux, and Windowslub AMQSPRMA w systemie IBM i. Przykład przedstawia, w jaki sposób komunikaty odniesienia mogą być umieszczane w kolejce, odbierane przez wyjścia komunikatów i pobierane z kolejki.

Przykłady komunikatów referencyjnych są uruchamiane w następujący sposób:




*Rysunek 131. Uruchamianie przykładowych komunikatów referencyjnych*

1. Skonfiguruj środowisko do uruchamiania programów nasłuchujących, kanałów i monitorów wyzwalaczy oraz zdefiniuj kanały i kolejki.

   Na potrzeby opisu sposobu konfigurowania komunikatu referencyjnego w tym przykładzie komputer nadawczy jest nazywany MACHINE1 z menedżerem kolejek o nazwie QMGR1 i komputerem odbiorczym jako MACHINE2 z menedżerem kolejek o nazwie QMGR2.

   **Uwaga:** Poniższe definicje umożliwiają zbudowanie komunikatu odniesienia w celu wysłania pliku typu FLATFILE z menedżera kolejek QMGR1 do QMGR2 i ponownego utworzenia pliku zgodnie z definicją w wywołaniu komendy AMQSPRM (lub AMQSPRMA w systemie IBM i). Komunikat odniesienia (w tym dane zbioru) jest wysyłany przy użyciu kanału CHL1 i kolejki transmisji XMITQ oraz umieszczany w kolejce DQ. Raporty wyjątków i COA są wysyłane z powrotem do QMGR1 za pomocą kanału REPORT i kolejki transmisji QMGR1.

   Aplikacja, która odbiera komunikat odniesienia (AMQSGRM lub AMQSGRMA w systemie IBM i ), jest wyzwalana przy użyciu kolejki inicjującej INITQ i procesu PROC. Upewnij się, że pola CONNAME są ustawione poprawnie, a pole MSGEXIT odzwierciedla strukturę katalogów, w zależności od typu komputera i miejsca, w którym jest zainstalowany produkt IBM MQ .

IBM i W definicjach MQSC używane są style AIX do definiowania wyjść, więc jeśli używane są komendy MQSC w systemie IBM i, należy je odpowiednio zmodyfikować. Należy pamiętać, że w danych komunikatu FLATFILE jest rozróżniana wielkość liter i przykład nie będzie działać, jeśli nie zostanie zapisany wielkimi literami.

Na komputerze MACHINE1, menedżer kolejek QMGR1

**Składnia komend MQSC**

```
define chl(chl1) chltype(sdr) trptype(tcp) conname('machine2') xmitq(xmitq)
msgdata(FLATFILE) msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)
')

define ql(xmitq) usage(xmitq)

define chl(report) chltype(rcvr) trptype(tcp) replace

define qr(qr) rname(dq) rqmname(qmgr2) xmitq(xmitq) replace
```

IBM i **Składnia komendy IBM i**

**Uwaga:** Jeśli nazwa menedżera kolejek nie zostanie określona, system użyje domyślnego menedżera kolejek.

```
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)
```

Na komputerze MACHINE2, menedżer kolejek QMGR2

**Składnia komend MQSC**

```
define chl(chl1) chltype(rcvr) trptype(tcp)
msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)')
        msgdata(flatfile)

define chl(report) chltype(sdr) trptype(tcp) conname('MACHINE1')
        xmitq(qmgr1)

define ql(initq)

define ql(qmgr1) usage(xmitq)

define pro(proc) applicid('/usr/lpp/mqm/samp/bin/amqsgrm')

define ql(dq) initq(initq) process(proc) trigger trigtype(first)
```

IBM i **IBM i składnia komendy**

**Uwaga:** IBM i W systemie IBM i, jeśli nie zostanie podana nazwa menedżera kolejek, system użyje domyślnego menedżera kolejek.

```
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. Po utworzeniu obiektów IBM MQ :
   a. Jeśli ma to zastosowanie do platformy, uruchom program nasłuchujący dla nadawczego i odbiorczego menedżera kolejek.
   b. Uruchom kanały CHL1 i REPORT
   c. W odbierającym menedżerze kolejek uruchom monitor wyzwalacza dla kolejki inicjującej INITQ
3. Wywołaj przykładowy program put Reference Message AMQSPRM (AMQSPRMA w systemie IBM i ) z wiersza komend, używając następujących parametrów:

   **-m**
   Nazwa lokalnego menedżera kolejek. Wartość domyślna to domyślny menedżer kolejek.

   **-i**
   Nazwa i położenie pliku źródłowego

   **-o**
   Nazwa i położenie pliku docelowego

   **-q**
   Nazwa kolejki.

   **-g**
   Nazwa menedżera kolejek, w którym istnieje kolejka zdefiniowana w parametrze -q. Wartością domyślną jest menedżer kolejek określony w parametrze -m.

   **-t**
   Typ obiektu

   **-w**
   Przedział czasu oczekiwania, czyli czas oczekiwania na raporty wyjątków i COA z odbierającego menedżera kolejek

   Na przykład, aby użyć przykładu z wcześniej zdefiniowanymi obiektami, należy użyć następujących parametrów:

```
-mQMGR1 -iInput File -oOutput File -qQR -tFLATFILE -w120
```

   Zwiększenie czasu oczekiwania umożliwia wysłanie dużego pliku przez sieć przed przekroczeniem limitu czasu przez program umieszczający komunikaty.

```
amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
```

   **Użytkownicy systemu IBM i:** 

W systemie IBM iwykonaj następujące kroki:

   a. Użyj następującej komendy:

```
CALL    PGM(QMQM/AMQSPRM4) PARM('-mQMGR1' +
'-i/refmsgs/rmsg1' +
'-o/refmsgs/rmsgx' '-qQR' +
'-gQMGR1' '-tFLATFILE' '-w15')
```

Zakłada się, że oryginalny plik `rmsg1` znajduje się w katalogu IFS `/refmsgs` i że plik docelowy ma być `rmsgx` w katalogu IFS `/refmsgs` w systemie docelowym.

b. Utwórz własny katalog przy użyciu komendy CRTDIR, a nie katalogu głównego.

c. Podczas wywoływania programu, który umieszcza dane, należy pamiętać, że nazwa pliku wyjściowego musi odzwierciedlać konwencję nazewnictwa IFS; na przykład /TEST/FILENAME tworzy plik o nazwie FILENAME w katalogu TEST.

**Uwaga:**

IBM i W systemie IBM ipodczas określania parametrów można używać ukośnika (/) lub myślnika (-). Na przykład:

```
amqsprm /i d:\files\infile.dat /o e:\files\outfile.dat /q QR
/m QMGR1 /w 30 /t FLATFILE
```

Linux AIX W przypadku platform AIX and Linux należy użyć dwóch ukośników odwrotnych (\\) zamiast jednego, aby oznaczyć docelowy katalog plików. Dlatego komenda **amqsprm** wygląda następująco:

```
amqsprm -i /files/infile.dat -o e:\\files\\outfile.dat -q QR
-m QMGR1 -w 30 -t FLATFILE
```

Uruchomienie programu put Reference Message powoduje wykonanie następujących czynności:

- Komunikat odniesienia jest umieszczany w kolejce QR w menedżerze kolejek QMGR1.
- Plik źródłowy i ścieżka znajdują się w katalogu `d:\files\infile.dat` i istnieją w systemie, w którym wprowadzono przykładową komendę.
- Jeśli kolejka QR jest kolejką zdalną, komunikat odniesienia jest wysyłany do innego menedżera kolejek w innym systemie, w którym został utworzony plik o nazwie i ścieżce `e:\files\outfile.dat`. Treść tego pliku jest taka sama, jak plik źródłowy.
- amqsprm czeka 30 sekund na raport COA z docelowego menedżera kolejek.
- Typem obiektu jest `flatfile`, dlatego kanał używany do przenoszenia komunikatów z kolejki QR musi być określony w polu *MsgData* .

4. Po zdefiniowaniu kanałów należy wybrać wyjście komunikatu na końcu wysyłającym i odbierającym jako amqsxrm.

Windows Jest ona definiowana w systemie Windows w następujący sposób:

```
msgexit(' pathname\amqsxrm.dll(MsgExit)')
```

Linux AIX Jest ona definiowana w systemie AIX and Linux w następujący sposób:

```
msgexit(' pathname/amqsxrm(MsgExit)')
```

Jeśli określono nazwę ścieżki, należy podać pełną nazwę. Jeśli nazwa ścieżki zostanie pominięta, zakłada się, że program znajduje się w ścieżce określonej w pliku `qm.ini` (lub, w systemie IBM MQ for Windows, w ścieżce określonej w rejestrze).

5. Wyjście kanału odczytuje nagłówek komunikatu odniesienia i znajduje plik, do którego się odwołuje.
6. Wyjście kanału może następnie segmentować plik przed wysłaniem go w dół kanału wraz z nagłówkiem.

Linux AIX W systemie AIX and Linuxzmień właściciela grupy katalogu docelowego na mqm, aby przykładowe wyjście komunikatu może utworzyć plik w tym katalogu. Zmień również

uprawnienia katalogu docelowego, aby umożliwić członkom grupy mqm zapis w tym katalogu. Dane pliku nie są przechowywane w kolejkach systemu IBM MQ .

7. Gdy ostatni segment pliku jest przetwarzany przez wyjście odbierającego komunikatu, komunikat odniesienia jest umieszczany w kolejce docelowej określonej przez parametr amqsprm. Jeśli ta kolejka jest wyzwalana (czyli definicja określa atrybuty kolejki **Trigger**, **InitQ**i **Process** ), wyzwalany jest program określony przez parametr PROC kolejki docelowej. Program, który ma być wyzwalany, musi być zdefiniowany w polu ApplId atrybutu **Process** .
8. Gdy komunikat odniesienia dotrze do kolejki docelowej (DQ), raport COA jest wysyłany z powrotem do aplikacji umieszczającej (amqsprm).
9. Przykład Get Reference Message, amqsgrm, pobiera komunikaty z kolejki określonej w wejściowym komunikacie wyzwalacza i sprawdza, czy plik istnieje.

*Projekt przykładu Put Reference Message (amqsprma.c, AMQSPRM4)*
Ten temat zawiera szczegółowy opis przykładowego komunikatu z odwołaniem do umieszczania (Put Reference Message).

W tym przykładzie tworzony jest komunikat odniesienia, który odwołuje się do pliku i umieszcza go w określonej kolejce:

1. Przykład nawiązuje połączenie z lokalnym menedżerem kolejek przy użyciu wywołania MQCONN.
2. Następnie otwiera (MQOPEN) kolejkę modelową, która jest używana do odbierania komunikatów raportu.
3. Przykład tworzy komunikat odniesienia zawierający wartości wymagane do przeniesienia pliku, na przykład nazwy pliku źródłowego i docelowego oraz typ obiektu. Na przykład w przykładzie dostarczonym z produktem IBM MQ jest tworzony komunikat odniesienia w celu wysłania pliku `d:\x\file.in` z katalogu QMGR1 do katalogu QMGR2 i ponownego utworzenia pliku jako `d:\y\file.out` przy użyciu następujących parametrów:

   ```
   amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
   ```

   Gdzie QR jest definicją kolejki zdalnej, która odwołuje się do kolejki docelowej w systemie QMGR2.

   **Uwaga:** W przypadku platform AIX and Linux należy użyć dwóch ukośników odwrotnych (\\) zamiast jednego, aby oznaczyć docelowy katalog plików. Dlatego komenda **amqsprm** wygląda następująco:

   ```
   amqsprm -q QR -m QMGR1 -i /x/file.in -o d:\\y\\file.out -t FLATFILE
   ```

4. Komunikat odniesienia jest umieszczany (bez danych zbioru) w kolejce określonej przez parametr /q. Jeśli jest to kolejka zdalna, komunikat jest umieszczany w odpowiedniej kolejce transmisji.
5. Próbka oczekuje przez czas określony w parametrze /w (domyślnie 15 sekund) dla raportów COA, które wraz z raportami o wyjątkach są wysyłane z powrotem do kolejki dynamicznej utworzonej w lokalnym menedżerze kolejek (QMGR1).

*Projekt przykładu Reference Message Exit (amqsxrma.c, AMQSXRM4)*
W tym przykładzie rozpoznawane są komunikaty odniesienia z typem obiektu, który jest zgodny z typem obiektu w polu danych użytkownika wyjścia komunikatu w definicji kanału.

W przypadku tych komunikatów mają miejsce następujące zdarzenia:

- W kanale nadawcy lub serwera określona długość danych jest kopiowana z określonego przesunięcia określonego pliku do miejsca pozostałego w buforze agenta po komunikacie referencyjnym. Jeśli koniec pliku nie zostanie osiągnięty, komunikat odniesienia jest umieszczany z powrotem w kolejce transmisji po zaktualizowaniu pola *DataLogicalOffset* .
- Jeśli w kanale requestera lub odbiorcy pole *DataLogicalOffset* ma wartość zero, a określony plik nie istnieje, zostanie utworzony. Dane następujące po komunikacie odniesienia są dodawane na końcu określonego zbioru. Jeśli komunikat odniesienia nie jest ostatnim komunikatem dla określonego zbioru, jest on odrzucany. W przeciwnym razie jest on zwracany do wyjścia kanału, bez dołączonych danych, w celu umieszczenia go w kolejce docelowej.

W przypadku kanałów nadawczych i kanałów serwera, jeśli pole *DataLogicalLength* w wejściowym komunikacie odniesienia ma wartość zero, pozostała część pliku, od *DataLogicalOffset* do końca pliku, ma zostać wysłana wzdłuż kanału. Jeśli wartość jest różna od zera, wysyłana jest tylko określona długość.

Jeśli wystąpi błąd (na przykład, jeśli przykład nie może otworzyć pliku), MQCXP. Parametr *ExitResponse* jest ustawiony na wartość MQXCC_SUPPRESS_FUNCTION, aby przetwarzany komunikat był umieszczany w kolejce niedostarczonych komunikatów, a nie w kolejce docelowej. Kod sprzężenia zwrotnego jest zwracany w MQCXP. *Feedback* i powrócić do aplikacji, która umieściła komunikat w polu *Feedback* deskryptora komunikatu raportu. Jest to spowodowane tym, że aplikacja umieszczała żądane raporty o wyjątkach, ustawiając wartość MQRO_EXCEPTION w polu *Report* deskryptora MQMD.

Jeśli kodowanie lub identyfikator *CodedCharacterSetId* (CCSID) komunikatu odniesienia różni się od kodowania w menedżerze kolejek, komunikat odniesienia jest konwertowany na kodowanie lokalne i identyfikator CCSID. W tym przykładzie, amqsprm, format obiektu to MQFMT_STRING, więc amqsxrm przekształca dane obiektu na lokalny identyfikator CCSID na końcu odbierającym przed zapisaniem danych do pliku.

Nie należy określać formatu przesyłanego pliku jako MQFMT_STRING, jeśli plik zawiera znaki wielobajtowe (na przykład DBCS lub Unicode). Jest to spowodowane tym, że znak wielobajtowy może zostać podzielony, gdy plik jest segmentowany na końcu wysyłania. Aby przesłać i przekształcić taki plik, należy określić format inny niż MQFMT_STRING, tak aby wyjście komunikatu odwołania nie dokonywało jego konwersji i przekształcić plik po zakończeniu operacji przesyłania.

*Kompilowanie przykładu wyjścia komunikatu referencyjnego*
Aby skompilować przykład Reference Message Exit, należy użyć komendy dla platformy, na której zainstalowano produkt IBM MQ .

*MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Aby skompilować amqsxrma, należy użyć następujących komend:

### wł.AIX

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 amqsqrma.c
```

### wł.IBM i

IBM i

```
CRTCMOD MODULE(MYLIB/AMQSXRMA) SRCFILE(QMQMSAMP/QCSRC)
TERASPACE(*YES *TSIFC)
```

**Uwaga:**

1. Aby utworzyć moduł w taki sposób, aby korzystał z systemu plików IFS, należy dodać opcję `SYSIFCOPT(*IFSIO)`
2. Aby utworzyć program do użycia z kanałami niewątkowymi, należy użyć następującej komendy: `CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM)`
3. Aby utworzyć program do użycia z kanałami wątkowymi, należy użyć następującej komendy: `CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM_R)`

### wł.Linux

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
```

### wł.Windows

Windows Produkt IBM MQ dostarcza teraz bibliotekę mqm z pakietami klienckimi, a także pakietami serwera, dlatego w poniższym przykładzie zamiast pliku `mqmvx.lib`jest używany plik `mqm.lib` :

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

**Pojęcia pokrewne**

“Zapisywanie programów obsługi wyjścia kanału” na stronie 993
Poniższe informacje mogą być pomocne podczas pisania programów obsługi wyjścia kanału.

*Projekt przykładu Get Reference Message (amqsgrma.c, AMQSGRM4)*
W tym temacie opisano projekt przykładu Get Reference Message.

Logika programu jest następująca:

1. Przykład jest wyzwalany i wyodrębnia nazwy kolejek i menedżerów kolejek z wejściowego komunikatu wyzwalacza.
2. Następnie łączy się on z określonym menedżerem kolejek za pomocą komendy MQCONN i otwiera określoną kolejkę za pomocą komendy MQOPEN.
3. W tym przykładzie komenda MQGET jest wykonywana z odstępem czasu oczekiwania wynoszącym 15 sekund w pętli w celu pobrania komunikatów z kolejki.
4. Jeśli komunikat jest komunikatem referencyjnym, przykład sprawdza, czy istnieje plik, który został przesłany.
5. Następnie zamyka kolejkę i rozłącza się z menedżerem kolejek.

### *Przykładowe programy żądania*

Przykładowe programy żądania demonstrują przetwarzanie klient/serwer. Przykłady to klienty, które umieszczają komunikaty żądań w kolejce serwera docelowego, która jest przetwarzana przez program serwera. Oczekują, aż program serwera umieścił komunikat odpowiedzi w kolejce odpowiedzi.

Próbki żądań umieszczają serię komunikatów żądań w kolejce serwera docelowego przy użyciu wywołania MQPUT. Te komunikaty określają kolejkę lokalną, SYSTEM.SAMPLE.REPLY jako kolejka odpowiedzi, która może być kolejką lokalną lub zdalną. Programy oczekują na komunikaty odpowiedzi, a następnie je wyświetlają. Odpowiedzi są wysyłane tylko wtedy, gdy kolejka serwera docelowego jest przetwarzana przez aplikację serwera lub gdy aplikacja jest wyzwalana w tym celu (programy przykładowe Inquire, Set i Echo są przeznaczone do wyzwalania). Próbka w języku C oczekuje 1 minutę (próbka w języku COBOL czeka 5 minut), na nadejście pierwszej odpowiedzi (w celu umożliwienia wyzwolenia aplikacji serwera) i 15 sekund na kolejne odpowiedzi, ale obie próbki mogą zakończyć się bez uzyskania żadnych odpowiedzi. Nazwy przykładowych programów Request znajdują się w sekcji “Funkcje demonstrowane w przykładowych programach w środowisku wieloplatformowym” na stronie 1092 .

*Uruchamianie przykładowych programów żądania*

## Uruchamianie przykładów amqsreq0.c, amqsreq i amqsreqc

Wersja C programu przyjmuje trzy parametry:

1. Nazwa kolejki serwera docelowego (niezbędne)
2. Nazwa menedżera kolejek (opcjonalna)

3. Kolejka odpowiedzi (opcjonalnie)

Na przykład wprowadź jedną z następujących wartości:

- `amqsreq myqueue qmanagername replyqueue`
- `amqsreqc myqueue qmanagername`
- `amq0req0 myqueue`

gdzie `myqueue` jest nazwą kolejki serwera docelowego, `qmanagername` jest nazwą menedżera kolejek, który jest właścicielem produktu `myqueue`, a `replyqueue` jest nazwą kolejki odpowiedzi.

Jeśli nazwa menedżera kolejek zostanie pominięta, przyjmuje się, że właścicielem kolejki jest domyślny menedżer kolejek. Jeśli nazwa kolejki odpowiedzi zostanie pominięta, zostanie udostępniona domyślna kolejka odpowiedzi.

## Uruchamianie przykładu amq0req0.cbl

Wersja języka COBOL nie ma żadnych parametrów. Zostanie nawiązane połączenie z domyślnym menedżerem kolejek, a po jego uruchomieniu zostanie wyświetlone zapytanie:

```
Please enter the name of the target server queue
```

Program pobiera dane wejściowe ze zmiennej StdIn i dodaje każdy wiersz do kolejki serwera docelowego, pobierając każdy wiersz tekstu jako treść komunikatu żądania. Program kończy działanie, gdy odczytywany jest pusty wiersz.

## Uruchamianie przykładu AMQSREQ4

Program w języku C tworzy komunikaty, pobierając dane ze standardowego wejścia (klawiatury) z pustym wejściem kończącym. Program przyjmuje maksymalnie trzy parametry: nazwa kolejki docelowej (wymagana), nazwa menedżera kolejek (opcjonalna) i nazwa kolejki zwrotnej (opcjonalna). Jeśli nie określono nazwy menedżera kolejek, używany jest domyślny menedżer kolejek. Jeśli nie określono kolejki odpowiedzi, należy podać wartość SYSTEM.SAMPLE.REPLY REPLY.

Poniżej przedstawiono przykład wywoływania przykładowego programu w języku C, który określa kolejkę odpowiedzi, ale pozwala menedżerowi kolejek na ustawienie domyślne:

```
CALL PGM(QMQM/AMQSREQ4) PARM('SYSTEM.SAMPLE.LOCAL' '' 'SYSTEM.SAMPLE.REPLY')
```

**Uwaga:** Należy pamiętać, że w nazwach kolejek rozróżniana jest wielkość liter. Wszystkie kolejki utworzone przez przykładowy program tworzenia plików AMQSAMP4 mają nazwy utworzone wielkimi literami.

## Uruchamianie przykładu AMQ0REQ4

Program w języku COBOL tworzy komunikaty, akceptując dane z klawiatury. Aby uruchomić program, wywołaj program i określ nazwę kolejki docelowej jako parametr. Program przyjmuje dane wejściowe z klawiatury do buforu i tworzy komunikat żądania dla każdego wiersza tekstu. Program zostanie zatrzymany po wpisaniu pustego wiersza na klawiaturze.

*Uruchamianie przykładu żądania przy użyciu wyzwalania*

Jeśli przykład jest używany z wyzwalaniem i jednym z przykładowych programów Inquire, Set lub Echo, wiersz danych wejściowych musi być nazwą kolejki, do której program wyzwalany ma uzyskać dostęp.

ALW *Uruchamianie przykładu żądania przy użyciu wyzwalania w systemie AIX, Linux, and Windows*

W systemie AIX, Linux, and Windowsuruchom program monitora wyzwalacza RUNMQTRM w jednej sesji, a następnie uruchom program amqsreq w innej sesji.

Aby uruchomić przykłady za pomocą wyzwalania:

1. Uruchom program monitora wyzwalacza RUNMQTRM w jednej sesji (kolejka inicjująca SYSTEM.SAMPLE.TRIGGER jest dostępna do użycia).
2. Uruchom program amqsreq w innej sesji.
3. Upewnij się, że zdefiniowano kolejkę serwera docelowego.

   Przykładowe kolejki dostępne do użycia jako kolejka serwera docelowego dla próbki żądania do umieszczania komunikatów są następujące:

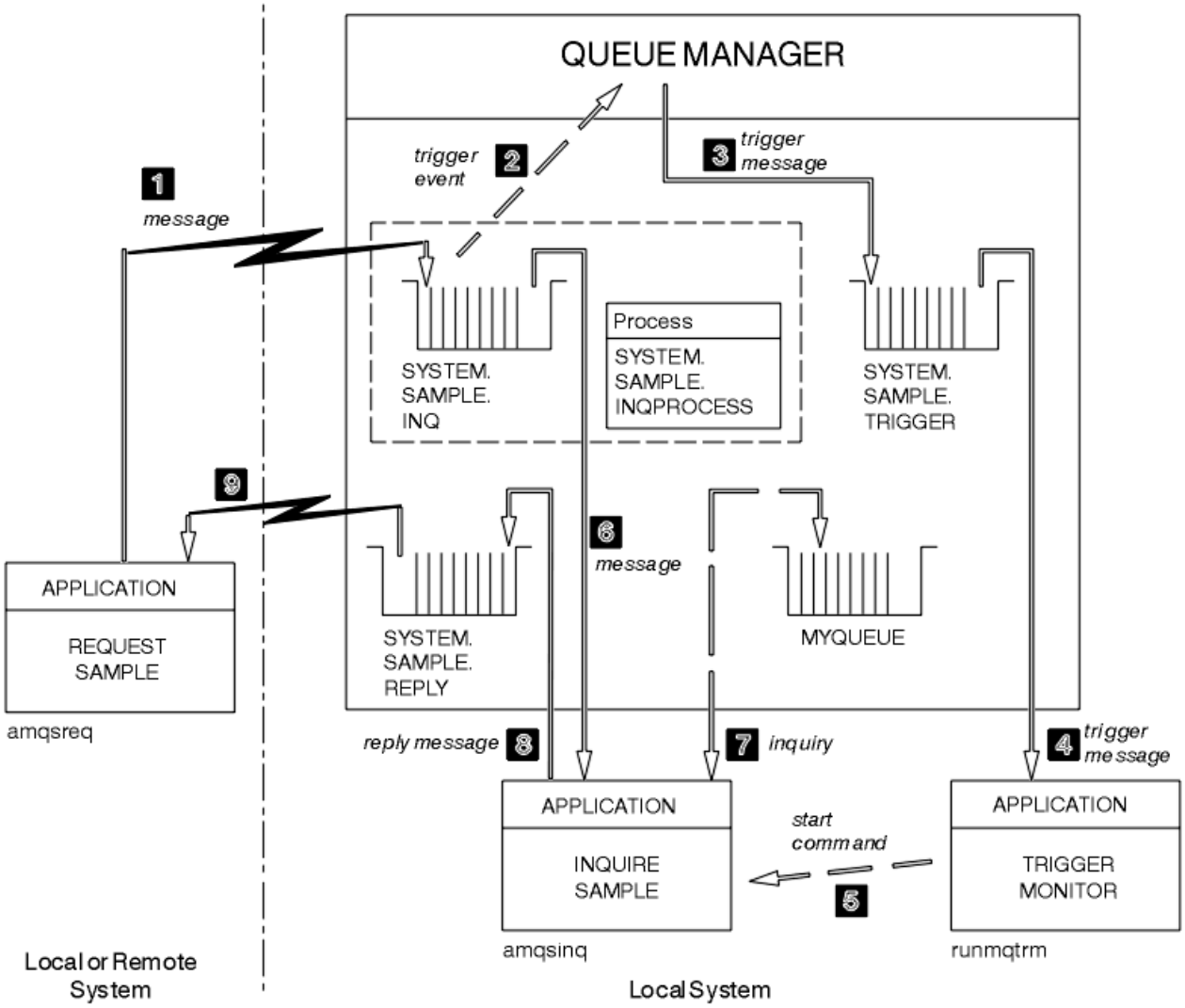
   - SYSTEM.SAMPLE.INQ -dla przykładowego programu Inquire
   - SYSTEM.SAMPLE.SET -dla przykładowego programu Set
   - SYSTEM.SAMPLE.ECHO -dla przykładowego programu Echo

   Kolejki te mają wyzwalacz typu FIRST, więc jeśli przed uruchomieniem przykładu żądania w kolejkach znajdują się już komunikaty, aplikacje serwera nie są wyzwalane przez wysyłane komunikaty.
4. Upewnij się, że zdefiniowano kolejkę dla programu przykładowego Inquire, Set lub Echo, który ma być używany.

Oznacza to, że monitor wyzwalacza jest gotowy, gdy próbka żądania wysyła komunikat.

**Uwaga:** Przykładowe definicje procesów utworzone za pomocą komend RUNMQSC i pliku amqscos0.tst wyzwalają przykłady w języku C. Zmień definicje procesów w pliku amqscos0.tst i użyj komendy RUNMQSC z tym zaktualizowanym plikiem, aby używać wersji COBOL.

Rysunek 132 na stronie 1148 przedstawia sposób użycia próbek Request i Inquire razem.




*Rysunek 132. Żądaj i sprawdzaj próbki za pomocą wyzwalania*

W pliku Rysunek 132 na stronie 1148 próbka żądania umieszcza komunikaty w kolejce serwera docelowego, SYSTEM.SAMPLE.INQi przykład Inquire wysyłają zapytanie do kolejki MYQUEUE. Alternatywnie można użyć jednej z przykładowych kolejek zdefiniowanych podczas uruchamiania komendy amqscos0.tstlub dowolnej innej kolejki zdefiniowanej przez użytkownika dla przykładu Inquire.

**Uwaga:** Liczby w polu Rysunek 132 na stronie 1148 przedstawiają sekwencję zdarzeń.

Aby uruchomić próbki żądań i zapytań przy użyciu wyzwalania:

1. Sprawdź, czy kolejki, które mają być używane, są zdefiniowane. Uruchom komendę amqscos0.tst, aby zdefiniować przykładowe kolejki i zdefiniować kolejkę MYQUEUE.
2. Uruchom komendę monitora wyzwalacza RUNMQTRM:

   ```
   RUNMQTRM -m qmanagername -q SYSTEM.SAMPLE.TRIGGER
   ```

3. Uruchom przykład żądania

   ```
   amqsreq SYSTEM.SAMPLE.INQ
   ```

   **Uwaga:** Obiekt procesu definiuje, co ma być wyzwalane. Jeśli klient i serwer nie działają na tej samej platformie, wszystkie procesy uruchomione przez monitor wyzwalacza muszą definiować *ApplType*,

w przeciwnym razie serwer przyjmuje swoje domyślne definicje (tj. typ aplikacji, która jest zwykle powiązana z serwerem) i powoduje awarię.

Listę typów aplikacji zawiera sekcja ApplType.

4. Wprowadź nazwę kolejki, która ma być używana przez przykład zapytania:

```
MYQUEUE
```

5. Wprowadź pusty wiersz (aby zakończyć program żądania).
6. Następnie w przykładzie żądania zostanie wyświetlony komunikat zawierający dane programu zapytania uzyskane z kolejki MYQUEUE.

Można użyć więcej niż jednej kolejki. W tym przypadku należy wprowadzić nazwy innych kolejek w kroku “4” na stronie 1149.

Więcej informacji na temat wyzwalania zawiera sekcja “Uruchamianie aplikacji IBM MQ przy użyciu wyzwalaczy” na stronie 891.

IBM i *Uruchamianie przykładu żądania przy użyciu wyzwalania w systemie IBM i*

W systemie IBM iuruchom przykładowy serwer wyzwalacza AMQSERV4w jednym zadaniu, a następnie uruchom AMQSREQ4 w innym. Oznacza to, że serwer wyzwalacza jest gotowy, gdy przykładowy program żądania wysyła komunikat.

**Uwaga:**

1. Przykładowe definicje utworzone przez program AMQSAMP4 wyzwalają wersje C przykładów. Aby wyzwolić wersje języka COBOL, należy zmienić definicje procesów SYSTEM.SAMPLE.ECHOPROCESS, SYSTEM.SAMPLE.INQPROCESSi SYSTEM.SAMPLE.SETPROCESS. Można użyć komendy CHGMQMPRC (szczegółowe informacje na ten temat zawiera sekcja Zmiana procesu MQ (CHGMQMPRC) ). W tym celu należy dokonać edycji lub uruchomić własną wersję programu AMQSAMP4.
2. Kod źródłowy dla AMQSERV4 jest dostarczany tylko dla języka C. Jednak skompilowana wersja (której można użyć z przykładami języka COBOL) jest dostarczana w bibliotece QMQM.

Komunikaty żądań można umieścić w następujących przykładowych kolejkach serwera:

- SYSTEM.SAMPLE.ECHO (dla przykładowych programów Echo)
- SYSTEM.SAMPLE.INQ (dla programów przykładowych Inquire)
- SYSTEM.SAMPLE.SET (dla programów przykładowych Set)

Schemat blokowy dla pliku SYSTEM.SAMPLE.ECHO ECHO jest pokazany na Rysunek 133 na stronie 1151. Przy użyciu przykładowego pliku danych komenda wysyłająca do tego serwera żądanie programu w języku C wygląda następująco:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(ECHO)')
```

**Uwaga:** Ta przykładowa kolejka ma wyzwalacz typu FIRST, więc jeśli w kolejce znajdują się już komunikaty przed uruchomieniem przykładu żądania, aplikacje serwera nie są wyzwalane przez wysyłane komunikaty.

Jeśli chcesz spróbować dalszych przykładów, możesz wypróbować następujące warianty:

- Użyj komendy AMQSTRG4 (lub jej odpowiednika w wierszu komend STRMQMTRM), aby uzyskać szczegółowe informacje, patrz sekcja Uruchamianie monitora wyzwalacza MQ (Start MQ Trigger Monitor-STRMQMTRM) ) Zamiast wartości AMQSERV4 można wprowadzić zadanie, ale potencjalne opóźnienia wprowadzania zadań mogą spowodować, że śledzenie tego, co się dzieje, będzie mniej łatwe.
- Uruchom plik SYSTEM.SAMPLE.INQUIRE i SYSTEM.SAMPLE.SET . Używając przykładowego pliku danych, komendy służące do wysyłania żądań programu w języku C do tych serwerów to odpowiednio:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(INQ)')
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(SET)')
```

Te przykładowe kolejki mają również wyzwalacz typu FIRST.

*Projekt przykładowego programu żądania*
Program otwiera kolejkę serwera docelowego, aby mógł umieszczać komunikaty. Używa wywołania MQOPEN z opcją MQOO_OUTPUT. Jeśli nie można otworzyć kolejki, program wyświetla komunikat o błędzie zawierający kod przyczyny zwrócony przez wywołanie MQOPEN.

Następnie program otwiera kolejkę odpowiedzi o nazwie SYSTEM.SAMPLE.REPLY , aby mogła otrzymać komunikaty odpowiedzi. W tym celu program używa wywołania MQOPEN z opcją MQOO_INPUT_EXCLUSIVE. Jeśli nie można otworzyć kolejki, program wyświetla komunikat o błędzie zawierający kod przyczyny zwrócony przez wywołanie MQOPEN.

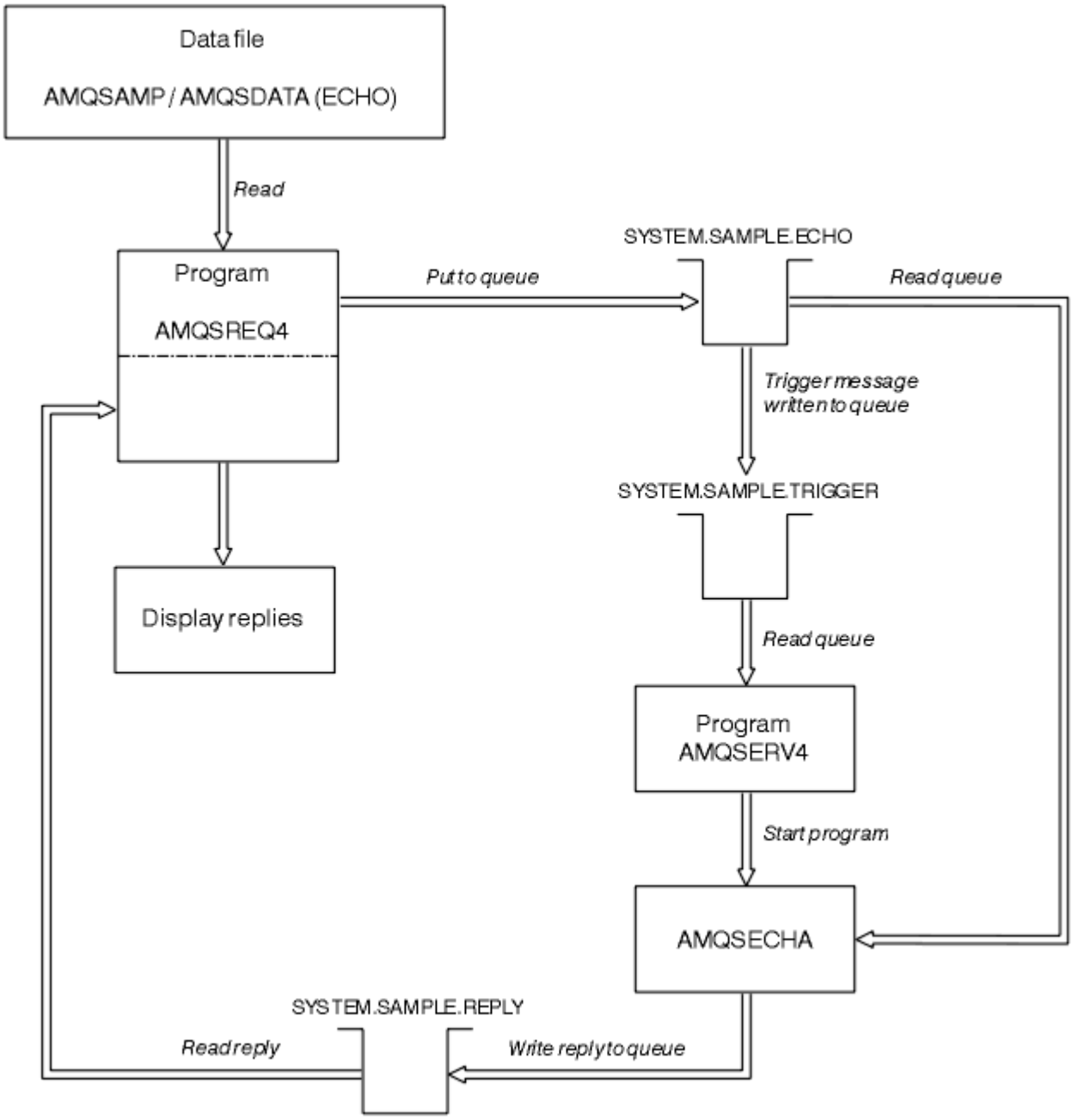
Dla każdego wiersza danych wejściowych program odczytuje tekst do buforu i używa wywołania MQPUT do utworzenia komunikatu żądania zawierającego tekst tego wiersza. W tym wywołaniu program używa opcji raportu MQRO_EXCEPTION_WITH_DATA, aby zażądać, aby wszystkie komunikaty raportu wysłane w związku z komunikatem żądania zawierały pierwsze 100 bajtów danych komunikatu. Działanie programu jest kontynuowane do momentu, gdy zostanie osiągnięty koniec wejścia lub wywołanie MQPUT nie powiedzie się.

Następnie program używa wywołania MQGET do usunięcia komunikatów odpowiedzi z kolejki i wyświetla dane zawarte w odpowiedziach. Wywołanie MQGET używa opcji MQGMO_WAIT, MQGMO_CONVERT i MQGMO_ACCEPT_TRUNCATED. Wartość *WaitInterval* wynosi 5 minut w wersji COBOL i 1 minutę w wersji C dla pierwszej odpowiedzi (w celu umożliwienia wyzwolenia aplikacji serwera) i 15 sekund dla kolejnych odpowiedzi. Program oczekuje przez te okresy, jeśli w kolejce nie ma żadnego komunikatu. Jeśli przed upływem tego okresu nie nadejdzie żaden komunikat, wywołanie nie powiedzie się i zostanie zwrócony kod przyczyny MQRC_NO_MSG_AVAILABLE. Wywołanie używa również opcji MQGMO_ACCEPT_TRUNCATED_MSG, dzięki czemu komunikaty dłuższe niż zadeklarowana wielkość buforu są obcinane.

Program demonstruje sposób czyszczenia pól *MsgId* i *CorrelId* struktury MQMD po każdym wywołaniu MQGET, ponieważ wywołanie to ustawia te pola na wartości zawarte w pobieranym komunikacie. Usunięcie zaznaczenia tych pól oznacza, że kolejne wywołania MQGET pobierają komunikaty w kolejności, w jakiej są przechowywane w kolejce.

Działanie programu jest kontynuowane do momentu, gdy wywołanie MQGET zwróci kod przyczyny MQRC_NO_MSG_AVAILABLE lub wywołanie MQGET nie powiedzie się. Jeśli wywołanie nie powiedzie się, program wyświetli komunikat o błędzie zawierający kod przyczyny.

Następnie program zamyka zarówno kolejkę serwera docelowego, jak i kolejkę zwrotną za pomocą wywołania MQCLOSE.




*Rysunek 133. Przykładowy schemat blokowy programu IBM i Client/Server (Echo)*

## *Przykładowe programy Set*

Programy przykładowe Set nie mogą wykonywać operacji umieszczania w kolejce przy użyciu wywołania MQSET w celu zmiany atrybutu **InhibitPut** kolejki. Informacje o projektowaniu przykładowych programów Set.

Nazwy tych programów można znaleźć w sekcji "Funkcje demonstrowane w przykładowych programach w środowisku wieloplatformowym" na stronie 1092 .

Programy są przeznaczone do uruchamiania jako programy wyzwalane, dlatego ich jedynym wejściem jest struktura MQTMC2 (komunikat wyzwalacza), która zawiera nazwę kolejki docelowej z atrybutami, do których ma zostać skierowane zapytanie. W wersji C używana jest również nazwa menedżera kolejek. Wersja języka COBOL używa domyślnego menedżera kolejek.

Aby proces wyzwalania działał, upewnij się, że program przykładowy Set, który ma być używany, jest wyzwalany przez komunikaty przychodzące do kolejki SYSTEM.SAMPLE.SET. W tym celu należy w polu

*ApplicId* definicji procesu SYSTEM.SAMPLE.SETPROCESSokreślić nazwę przykładowego programu, który ma zostać użyty. Przykładowa kolejka ma wyzwalacz typu FIRST. Jeśli przed uruchomieniem próbki żądania w kolejce znajdują się już komunikaty, próbka Set nie jest wyzwalana przez wysyłane komunikaty.

Po poprawnym ustawieniu definicji:

- ALW W systemach AIX, Linux, and Windows uruchom program **runmqtrm** w jednej sesji, a następnie uruchom program amqsreq w innej sesji.
- IBM i W systemie IBM iuruchom program AMQSERV4 w jednej sesji, a następnie program AMQSREQ4 w innej sesji. Zamiast wartości AMQSERV4można użyć wartości AMQSTRG4 , ale potencjalne opóźnienia wprowadzania zadań mogą ułatwić śledzenie tego, co się dzieje.

Przykładowe programy żądania służą do wysyłania komunikatów żądania, z których każdy zawiera tylko nazwę kolejki, do kolejki SYSTEM.SAMPLE.SET. Dla każdego komunikatu żądania programy przykładowe Set wysyłają komunikat odpowiedzi zawierający potwierdzenie, że operacje umieszczania zostały zablokowane w określonej kolejce. Odpowiedzi są wysyłane do kolejki odpowiedzi określonej w komunikacie żądania.

## Projekt przykładowego programu Set

Program otwiera kolejkę o nazwie określonej w strukturze komunikatu wyzwalacza, która została przekazana podczas uruchamiania. (Dla jasności będzie to *kolejka żądań*). Program używa wywołania MQOPEN do otwarcia tej kolejki dla wejścia współużytkowanego.

Program korzysta z wywołania MQGET w celu usunięcia komunikatów z tej kolejki. To wywołanie używa opcji MQGMO_ACCEPT_TRUNCATED_MSG i MQGMO_WAIT z odstępem czasu oczekiwania wynoszącym 5 sekund. Program testuje deskryptor każdego komunikatu, aby sprawdzić, czy jest to komunikat żądania; jeśli nie, program usuwa komunikat i wyświetla komunikat ostrzegawczy.

Dla każdego komunikatu żądania usuniętego z kolejki żądań program odczytuje nazwę kolejki (która będzie zwana *kolejką docelową* ). zawarte w danych i otwiera tę kolejkę za pomocą wywołania MQOPEN z opcją MQOO_SET. Następnie program używa wywołania MQSET do ustawienia wartości atrybutu **InhibitPut** kolejki docelowej na MQQA_PUT_INHIBITED.

Jeśli wywołanie MQSET powiedzie się, program użyje wywołania MQPUT1 do umieszczenia komunikatu odpowiedzi w kolejce odpowiedzi. Ten komunikat zawiera łańcuch `PUT inhibited`.

Jeśli wywołanie MQOPEN lub MQSET nie powiedzie się, program użyje wywołania MQPUT1 do umieszczenia komunikatu `report` w kolejce odpowiedzi. W polu *Feedback* deskryptora komunikatu tego komunikatu raportu znajduje się kod przyczyny zwrócony przez wywołanie MQOPEN lub MQSET, w zależności od tego, które wywołanie nie powiodło się.

Po wywołaniu MQSET program zamyka kolejkę docelową za pomocą wywołania MQCLOSE.

Jeśli w kolejce żądań nie ma już żadnych komunikatów, program zamyka tę kolejkę i rozłącza się z menedżerem kolejek.

### *Przykładowy program TLS*

AMQSSSLC jest przykładowym programem w języku C, który demonstruje sposób użycia struktur MQCNO i MQSCO do dostarczenia informacji o połączeniu klienta TLS w wywołaniu MQCONNX. Umożliwia to aplikacji MQI klienta udostępnianie w czasie wykonywania definicji kanału połączenia klienta i ustawień TLS bez tabeli definicji kanału klienta (CCDT).

Jeśli zostanie podana nazwa połączenia, program tworzy definicję kanału połączenia klienta w strukturze MQCD.

Jeśli podano nazwę rdzenia pliku repozytorium kluczy, program tworzy strukturę MQSCO. Jeśli podano także adres URL programu odpowiadającego OCSP, program tworzy strukturę MQAIR rekordu informacji uwierzytelniającej.

Następnie program łączy się z menedżerem kolejek za pomocą komendy MQCONNX. Następnie wysyła zapytanie i drukuje nazwę menedżera kolejek, z którym nawiązał połączenie.

Ten program jest przeznaczony do połączenia jako aplikacja kliencka MQI. Można ją jednak połączyć jako zwykłą aplikację MQI. W takim przypadku po prostu łączy się z lokalnym menedżerem kolejek i ignoruje informacje o połączeniu klienta.

V 9.3.0 V 9.3.0 Jeśli fraza hasła używana do uzyskania dostępu do repozytorium kluczy nie jest ukryta w pliku, należy ją podać w pliku **amqssslc** podczas działania aplikacji. Hasło można podać w następujący sposób:

- Żądanie **amqssslc** , aby zapytać o frazę hasła, lub
- Używając zmiennej środowiskowej *MQKEYRPWD* lub
- Używanie atrybutu **SSLKeyRepositoryPassword** w pliku konfiguracyjnym klienta

Więcej informacji na temat dostarczania hasła repozytorium kluczy do aplikacji IBM MQ MQI client zawiera sekcja Podawanie hasła repozytorium kluczy dla IBM MQ MQI client w systemie AIX, Linux, and Windows.

**amqssslc** akceptuje następujące parametry, z których wszystkie są opcjonalne:

**-m QmgrName**
Nazwa menedżera kolejek, z którym ma zostać nawiązane połączenie

**-c ChannelName**
Nazwa kanału, który ma być używany

**-x ConnName**
Nazwa połączenia z serwerem

Parametry TLS:

V 9.3.0 V 9.3.0 **-k KeyReposFileName**
Nazwa pliku repozytorium kluczy. Jeśli rozszerzenie pliku nie zostanie podane, przyjmuje się, że ma ono rozszerzenie .kdb. Na przykład:

```
/home/user/client.kdb
C:\User\client.p12
```

**-s CipherSpec**
Łańcuch CipherSpec kanału TLS odpowiadający elementowi **SSLCIPH** w definicji kanału SVRCONN w menedżerze kolejek.

**-f**
Określa, że muszą być używane tylko algorytmy z certyfikatem FIPS 140-2.

**-b VALUE1[,VALUE2...]**
Określa, że muszą być używane tylko algorytmy zgodne ze standardem Suite B. Ten parametr jest rozdzielaną przecinkami listą co najmniej jednej z następujących wartości: NONE,128_BIT,192_BIT. Wartości te mają takie samo znaczenie jak w przypadku zmiennej środowiskowej **MQSUITEB** i równoważnego ustawienia **EncryptionPolicySuiteB** w sekcji SSL pliku konfiguracyjnego klienta.

**-p Strategia**
Określa strategię sprawdzania poprawności certyfikatu, która ma być używana. Może to być jedna z następujących wartości:

**ANY**
Zastosuj wszystkie strategie sprawdzania poprawności certyfikatów obsługiwane przez bibliotekę bezpiecznych gniazd i zaakceptuj łańcuch certyfikatów, jeśli dowolna ze strategii uzna, że łańcuch certyfikatów jest poprawny. To ustawienie może być używane w celu zapewnienia maksymalnej kompatybilności wstecznej ze starszymi certyfikatami cyfrowymi, które nie są zgodne z nowoczesnymi standardami certyfikatów.

**RFC5280**
Zastosuj tylko strategię sprawdzania poprawności certyfikatu zgodną ze standardem RFC 5280. To ustawienie zapewnia bardziej rygorystyczne sprawdzanie poprawności niż ustawienie ANY, ale odrzuca niektóre starsze certyfikaty cyfrowe.

Wartością domyślną jest ANY.

**-l CertLabel**
Etykieta certyfikatu, która ma być używana dla bezpiecznego połączenia.

**Uwaga:** Należy podać wartość, używając małych liter.

V 9.3.0 V 9.3.0 **-w**
Określa, że program **amqsssl c** będzie pytany o hasło do repozytorium kluczy, które ma zostać podane.

V 9.3.0 V 9.3.0 **-i**
Określa, że program **amqsssl c** będzie pytał o klucz początkowy używany do szyfrowania frazy hasła repozytorium kluczy, która ma zostać podana.

Podaj tę opcję, jeśli początkowy plik kluczy został określony podczas szyfrowania frazy hasła repozytorium kluczy za pomocą programu narzędziowego **runmqicred** .

Parametr odwołania certyfikatu OCSP:

**-o URL**
URL programu odpowiadającego OCSP

Można również ustawić jedną z następujących zmiennych środowiskowych, aby podać referencje używane do uwierzytelniania w menedżerze kolejek:

**ID_UŻYTKOWNIKA MQSAMP_ID**
Ustaw identyfikator użytkownika, który ma być używany do uwierzytelniania połączenia, jeśli do uwierzytelniania w menedżerze kolejek ma być używany identyfikator użytkownika i hasło. Program poprosi o podanie hasła, które ma być dołączone do identyfikatora użytkownika.

V 9.3.4 Linux AIX **MQSAMP_TOKEN**
Ustaw niepustą wartość, aby podać znacznik uwierzytelniania w celu uwierzytelnienia w menedżerze kolejek. Program pyta o znacznik uwierzytelniania.

*Uruchamianie przykładowego programu TLS*
Aby uruchomić przykładowy program TLS, należy najpierw skonfigurować środowisko TLS. Następnie należy uruchomić przykład z wiersza komend, podając pewną liczbę parametrów.

## O tym zadaniu

Poniższe instrukcje uruchamiają przykładowy program przy użyciu certyfikatów osobistych. Zmieniając komendę, można na przykład użyć certyfikatów CA i sprawdzić ich status przy użyciu programu odpowiadającego OCSP. Zapoznaj się z instrukcjami w przykładzie.

## Procedura

1. Utwórz menedżer kolejek o nazwie QM1. Więcej informacji na ten temat zawiera sekcja crtmqm.
2. Utwórz repozytorium kluczy dla menedżera kolejek. Więcej informacji na ten temat zawiera sekcja Konfigurowanie repozytorium kluczy w systemie AIX, Linux, and Windows.
3. Utwórz repozytorium kluczy dla klienta. Nazwij to *clientkey.kdb*.

   Hasło repozytorium kluczy należy zapisać w pliku podczas tworzenia repozytorium kluczy.
4. Utwórz certyfikat osobisty dla menedżera kolejek. Więcej informacji na ten temat zawiera sekcja Tworzenie samopodpisanego certyfikatu osobistego w systemie AIX, Linux, and Windows.
5. Utwórz certyfikat osobisty dla klienta.
6. Wyodrębnij certyfikat osobisty z repozytorium kluczy serwera i dodaj go do repozytorium klienta. Więcej informacji na ten temat zawiera sekcja Wyodrębnianie publicznej części certyfikatu samopodpisanego z repozytorium kluczy w systemie AIX, Linux, and Windowsoraz sekcja Dodawanie certyfikatu ośrodka CA (lub publicznej części certyfikatu samopodpisanego) do repozytorium kluczy w systemach AIX, Linux, and Windows.

7. Wyodrębnij certyfikat osobisty z repozytorium kluczy klienta i dodaj go do repozytorium kluczy serwera.
8. Utwórz kanał połączenia z serwerem za pomocą komendy MQSC:

```
DEFINE CHANNEL(QM1SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256)
```

   Więcej informacji na ten temat zawiera sekcja Kanał połączenia z serwerem .
9. Zdefiniuj i uruchom program nasłuchujący kanału w menedżerze kolejek. Więcej informacji na ten temat zawiera sekcja DEFINE LISTENER (Zdefiniuj program nasłuchujący) i sekcja START LISTENER(Uruchom program nasłuchujący).
10. Uruchom przykładowy program za pomocą następującej komendy:

V 9.3.0 V 9.3.0

```
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
```

## Wyniki

Program przykładowy wykonuje następujące działania:

1. Umożliwia nawiązanie połączenia z dowolnym określonym menedżerem kolejek lub z domyślnym menedżerem kolejek przy użyciu dowolnych określonych opcji.
2. Otwiera menedżer kolejek i pyta o jego nazwę.
3. Zamyka menedżer kolejek.
4. Rozłącza się z menedżerem kolejek.

Jeśli program przykładowy zostanie uruchomiony pomyślnie, zostaną wyświetlone dane wyjściowe podobne do poniższych:

```
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
```

Jeśli program przykładowy napotka problem, zostanie wyświetlony odpowiedni komunikat o błędzie, na przykład w przypadku określenia niepoprawnego programu odpowiadającego OCSP URL, zostanie wyświetlony następujący komunikat:

```
MQCONNX ended with reason code 2553
```

Listę kodów przyczyny można znaleźć w sekcji Kody zakończenia i kody przyczyny funkcji API.

## *Przykładowe programy wyzwalające*

Funkcja udostępniona w przykładzie wyzwalania jest podzbiorem funkcji udostępnionych w monitorze wyzwalacza w programie **runmqtrm** .

Nazwy tych programów można znaleźć w sekcji “Funkcje demonstrowane w przykładowych programach w środowisku wieloplatformowym” na stronie 1092 .

## Projekt próbki wyzwalającej

Wyzwalający program przykładowy otwiera kolejkę inicjującą za pomocą wywołania MQOPEN z opcją MQOO_INPUT_AS_Q_DEF. Pobiera on komunikaty z kolejki inicjującej przy użyciu wywołania MQGET z opcjami MQGMO_ACCEPT_TRUNCATED_MSG i MQGMO_WAIT, określając nieograniczony przedział czasu oczekiwania. Program usuwa zawartość pól *MsgId* i *CorrelId* przed każdym wywołaniem MQGET w celu pobrania komunikatów w kolejności.

Po pobraniu komunikatu z kolejki inicjującej program testuje komunikat, sprawdzając wielkość komunikatu, aby upewnić się, że ma on taką samą wielkość jak struktura MQTM. Jeśli ten test nie powiedzie się, program wyświetli ostrzeżenie.

W przypadku poprawnych komunikatów wyzwalacza próbka wyzwalania kopiuje dane z następujących pól: *ApplicId*, *EnvrData*, *Versio*n i *ApplType*. Ostatnie dwa z tych pól są polami liczbowymi, dlatego program tworzy zastąpienia znaków, które mają być używane w strukturze MQTMC2 dla systemów IBM i, AIX, Linux, and Windows .

Przykład wyzwalania wysyła komendę start do aplikacji określonej w polu *ApplicId* komunikatu wyzwalacza i przekazuje strukturę MQTMC2 lub MQTMC (znakowa wersja komunikatu wyzwalacza).

- ALW W systemach AIX, Linux, and Windows pole *EnvData* jest używane jako rozszerzenie łańcucha komendy wywołującej.
- IBM i W systemie IBM ijest on używany jako parametry wprowadzania zadania, na przykład priorytet zadania lub opis zadania.

Na koniec program zamyka kolejkę inicjującą.

## Kończenie uruchamiania przykładowych programów w systemie IBM i

IBM i

Program monitora wyzwalacza można zakończyć za pomocą opcji 2 żądania systemowego (ENDRQS) lub przez zablokowanie pobierania z kolejki wyzwalacza.

Jeśli używana jest przykładowa kolejka wyzwalaczy, komenda jest następująca:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') MQMNAME GETENBL(*NO)
```

**Ważne:** Przed ponownym uruchomieniem wyzwalania w tej kolejce należy wprowadzić następującą komendę:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*YES)
```

*Uruchamianie przykładowych programów wyzwalających*
Ten temat zawiera informacje dotyczące uruchamiania przykładowych programów wyzwalających.

## Uruchamianie przykładów amqstrg0.c, amqstrg i amqstrgc

Program przyjmuje 2 parametry:

1. Nazwa kolejki inicjującej (niezbędne)
2. Nazwa menedżera kolejek (opcjonalna)

Jeśli menedżer kolejek nie jest określony, nawiązywane jest połączenie z menedżerem domyślnym. Przykładowa kolejka inicjująca zostanie zdefiniowana po uruchomieniu komendy amqscos0.tst; nazwa tej kolejki to SYSTEM.SAMPLE.TRIGGER, i można go użyć podczas uruchamiania tego programu.

**Uwaga:** Funkcja w tym przykładzie jest podzbiorem pełnej funkcji wyzwalania, która jest dostarczana w programie runmqtrm.

## Uruchamianie przykładu AMQSTRG4

IBM i

Jest to monitor wyzwalacza dla środowiska IBM i . Wprowadza jedno zadanie IBM i dla każdej aplikacji, która ma zostać uruchomiona. Oznacza to, że istnieje dodatkowe przetwarzanie powiązane z każdym komunikatem wyzwalacza.

Komenda AMQSTRG4 (w produkcie QCSRC) przyjmuje dwa parametry: nazwę kolejki inicjującej, która ma być używana, oraz nazwę menedżera kolejek (opcjonalnie). AMQSAMP4 (w QCLSRC) definiuje przykładową kolejkę inicjującą, SYSTEM.SAMPLE.TRIGGER, którego można użyć podczas wypróbowania programów przykładowych.

Przy użyciu przykładowej kolejki wyzwalacza należy wydać następującą komendę:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

Alternatywnie można użyć równoważnej komendy CL STRMQMTRM. Szczegółowe informacje na ten temat zawiera sekcja Uruchamianie monitora wyzwalacza MQ (Start MQ Trigger Monitor-STRMQMTRM).

## Uruchamianie przykładu AMQSERV4

IBM i

Jest to serwer wyzwalacza dla środowiska IBM i . Dla każdego komunikatu wyzwalacza ten serwer uruchamia komendę uruchamiania w swoim własnym zadaniu w celu uruchomienia określonej aplikacji. Serwer wyzwalacza może wywoływać transakcje CICS .

Parametr AMQSERV4 przyjmuje dwa parametry: nazwę kolejki inicjującej, która ma być używana, oraz nazwę menedżera kolejek (opcjonalnie). AMQSAMP4 definiuje przykładową kolejkę inicjowania, SYSTEM.SAMPLE.TRIGGER, którego można użyć podczas wypróbowania programów przykładowych.

W przypadku użycia przykładowej kolejki wyzwalacza komenda do wykonania jest następująca:

```
CALL PGM(QMQM/AMQSERV4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

*Projekt serwera wyzwalaczy*
Projekt serwera wyzwalacza jest podobny do projektu monitora wyzwalacza, z kilkoma wyjątkami

Projekt serwera wyzwalaczy jest podobny do projektu monitora wyzwalaczy, z tą różnicą, że serwer wyzwalaczy:

- Umożliwia korzystanie z aplikacji MQAT_CICS oraz MQAT_OS400 .
- IBM i Wywołuje aplikacje IBM i we własnym zadaniu (lub używa komendy STRCICSUSR do uruchamiania aplikacji CICS ) zamiast wprowadzania zadania IBM i .
- W przypadku aplikacji CICS podstawiany jest *EnvData*, na przykład w celu określenia regionu CICS , z komunikatu wyzwalacza w komendzie STRCICSUSR.
- Otwiera kolejkę inicjującą dla współużytkowanych danych wejściowych, dzięki czemu wiele serwerów wyzwalaczy może działać w tym samym czasie.

**Uwaga:** Programy uruchomione przez komendę AMQSERV4 nie mogą używać wywołania MQDISC, ponieważ spowoduje to zatrzymanie serwera wyzwalaczy. Jeśli programy uruchomione przez komendę AMQSERV4 używają wywołania MQCONN, są wyświetlane kod przyczyny MQRC_ALREADY_CONNECTED.

## ALW *Korzystanie z przykładów TUXEDO w systemie AIX, Linux, and Windows*

Zapoznaj się z przykładowymi programami Put i Get dla TUXEDO i zbuduj środowisko serwera w TUXEDO.

## Zanim rozpoczniesz

Przed uruchomieniem tych przykładów należy zbudować środowisko serwera.

## O tym zadaniu

**Uwaga:** W tej sekcji znak ukośnika odwrotnego (\) jest używany do dzielenia długich komend w więcej niż jednym wierszu. Nie należy wprowadzać tego znaku. Wprowadź każdą komendę w jednym wierszu.

ALW *Budowanie środowiska serwera*
Informacje na temat budowania środowiska serwera dla systemu IBM MQ na różnych platformach.

## Zanim rozpoczniesz

Zakłada się, że używane jest działające środowisko TUXEDO.

AIX *Budowanie środowiska serwera dla systemu AIX (32-bitowego)*
Sposób budowania środowiska serwera dla systemu IBM MQ for AIX (32-bitowego).

## Procedura

1. Utwórz katalog (na przykład APPDIR), w którym zostanie zbudowane środowisko serwera, i wykonaj wszystkie komendy w tym katalogu.
2. Wyeksportuj następujące zmienne środowiskowe, gdzie TUXDIR jest katalogiem głównym dla TUXEDO, a *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ :

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib:/lib
```

3. Dodaj następujący wiersz do pliku TUXEDO udataobj/RM:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa -lmqm
```

4. Uruchom komendy:

```
$ 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. Dokonaj edycji pliku ubbstxcx.cfg i w razie potrzeby dodaj szczegóły dotyczące nazwy komputera, katalogów roboczych i menedżera kolejek:

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. Utwórz TLOGDEVICE:

```
$tmadmin -c
```

Zostanie wyświetlona zachęta. Po wyświetleniu tego monitu wpisz:

```
> crdl -z /APPDIR/TLOG1
```

7. Uruchom menedżer kolejek:

```
$ strmqm
```

8. Uruchom Tuxedo:

```
$ tmboot -y
```

## Co dalej

Za pomocą programów do obsługi zadań i zadań można teraz umieszczać komunikaty w kolejce i pobierać je z kolejki.

AIX *Budowanie środowiska serwera dla systemu AIX (64-bitowego)*
Informacje na temat budowania środowiska serwera dla systemu IBM MQ for AIX (64-bitowego).

## Procedura

1. Utwórz katalog (na przykład APPDIR), w którym zostanie zbudowane środowisko serwera, i wykonaj wszystkie komendy w tym katalogu.
2. Wyeksportuj następujące zmienne środowiskowe, gdzie TUXDIR reprezentuje katalog główny TUXEDO, a *MQ_INSTALLATION_PATH* reprezentuje katalog wysokiego poziomu, w którym zainstalowano produkt IBM MQ installed.:

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib64"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib64: MQ_INSTALLATION_PATH/lib64:/lib64
```

3. Dodaj następujący wiersz do pliku TUXEDO udataobj/RM:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa64 -lmqm
```

4. Uruchom komendy:

```
$ 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. Dokonaj edycji pliku ubbstxcx.cfg i w razie potrzeby dodaj szczegóły dotyczące nazwy komputera, katalogów roboczych i menedżera kolejek:

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. Utwórz TLOGDEVICE:

```
$tmadmin -c
```

   Zostanie wyświetlona zachęta. Po wyświetleniu tego monitu wpisz:

```
> crdl -z /APPDIR/TLOG1
```

7. Uruchom menedżer kolejek:

```
$ strmqm
```

8. Uruchom Tuxedo:

```
$ tmboot -y
```

## Co dalej

Za pomocą programów do obsługi zadań i zadań można teraz umieszczać komunikaty w kolejce i pobierać je z kolejki.

Windows *Budowanie środowiska serwera dla systemu Windows (32-bitowego)*
Budowanie środowiska serwera dla systemu IBM MQ for Windows (32-bitowego).

## O tym zadaniu

**Uwaga:** Zmień następujące pola określone jako *VARIABLES* na ścieżki katalogów:

*Tabela 164. Pola, które mają zostać zmienione na ścieżki do katalogów*

| **Pole** | **Ścieżka katalogu** |
|---|---|
| *MQMDIR (katalog MQMDIR)* | Ścieżka do katalogu określona podczas instalowania programu IBM MQ , na przykład `g:\Program Files\IBM\MQ`. |
| *KATALOG_TUXDIR* | Ścieżka katalogu określona podczas instalowania TUXEDO, na przykład `f:\tuxedo`. |
| *KATALOG_APLIKACJI* | Ścieżka do katalogu, która ma być używana dla przykładowej aplikacji, na przykład `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
```

*Rysunek 134. Przykład pliku ubbstxcn.cfg dla systemu IBM MQ for Windows*

**Uwaga:** Zmień nazwę komputera *MachineName*i ścieżki do katalogów, aby były zgodne z instalacją. Zmień również nazwę menedżera kolejek *MYQUEUEMANAGER* na nazwę menedżera kolejek, z którym ma zostać nawiązane połączenie.

Przykładowy plik `ubbconfig` dla systemu IBM MQ for Windows jest wymieniony w sekcji Rysunek 134 na stronie 1161. Jest on dostarczany jako plik `ubbstxcn.cfg` w katalogu przykładów IBM MQ .

Przykładowy plik makefile (patrz Rysunek 135 na stronie 1162) dostarczany dla produktu IBM MQ for Windows nosi nazwę `ubbstxmn.mak`i jest przechowywany w katalogu przykładów IBM MQ .

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
$(TUXDIR)\bin\buildclient  -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

*Rysunek 135. Przykładowy plik makefile TUXEDO dla systemu IBM MQ for Windows*

Aby zbudować środowisko serwera i przykłady, wykonaj następujące kroki.

## Procedura

1. Utwórz katalog aplikacji, w którym ma zostać zbudowana przykładowa aplikacja, na przykład:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Skopiuj następujące pliki przykładowe z katalogu przykładów IBM MQ do katalogu aplikacji:
   - `amqstxmn.mak`
   - `amqstxen.env`
   - `ubbstxcn.cfg`
3. Zmodyfikuj każdy z tych plików, aby ustawić nazwy katalogów i ścieżki do katalogów używane w danej instalacji.
4. Zmodyfikuj plik `ubbstxcn.cfg` (patrz sekcja Rysunek 134 na stronie 1161), aby dodać szczegóły dotyczące nazwy komputera i menedżera kolejek, z którym ma zostać nawiązane połączenie.
5. Dodaj następujący wiersz do pliku TUXEDO *TUXDIR*`udataobj\rm`:

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib\mqmxa.lib MQMDIR\tools\lib\mqm.lib
   ```

   Nowy wpis musi zawierać jeden wiersz w pliku.
6. Ustaw następujące zmienne środowiskowe:

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. Utwórz urządzenie TLOG dla TUXEDO.

   W tym celu należy wywołać funkcję `tmadmin -c`i wprowadzić następującą komendę:

   ```
   crdl -z APPDIR\TLOG
   ```

8. Ustaw katalog bieżący na *APPDIR*i wywołaj przykładowy plik makefile `amqstxmn.mak` jako zewnętrzny plik makefile projektu. Na przykład w programie Microsoft Visual C++ wydaj następującą komendę:

   ```
   msvc amqstxmn.mak
   ```

   Wybierz opcję **build** (buduj), aby zbudować wszystkie programy przykładowe.

Windows *Budowanie środowiska serwera dla systemu Windows (64-bitowego)*
Informacje na temat budowania środowiska serwera dla systemu IBM MQ for Windows (64-bitowego).

## O tym zadaniu

**Uwaga:** Zmień następujące pola określone jako *VARIABLES* na ścieżki katalogów:

*Tabela 165. Pola, które mają zostać zmienione na ścieżki do katalogów*

| **Pole** | **Ścieżka katalogu** |
|---|---|
| *MQMDIR (katalog MQMDIR)* | Ścieżka do katalogu określona podczas instalowania programu IBM MQ , na przykład `g:\Program Files\IBM\MQ`. |
| *KATALOG_TUXDIR* | Ścieżka katalogu określona podczas instalowania TUXEDO, na przykład `f:\tuxedo`. |
| *KATALOG_APLIKACJI* | Ścieżka do katalogu, która ma być używana dla przykładowej aplikacji, na przykład `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:\Programï¿½Files\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
```

*Rysunek 136. Przykład pliku ubbstxcn.cfg dla systemu IBM MQ for Windows*

**Uwaga:** Zmień nazwę komputera *MachineName*i ścieżki do katalogów, aby były zgodne z instalacją. Zmień również nazwę menedżera kolejek *MYQUEUEMANAGER* na nazwę menedżera kolejek, z którym ma zostać nawiązane połączenie.

Przykładowy plik ubbconfig dla IBM MQ for Windows jest wymieniony w sekcji Rysunek 136 na stronie 1164. Jest on dostarczany jako plik `ubbstxcn.cfg` w katalogu przykładów IBM MQ .

Przykładowy plik makefile (patrz sekcja Rysunek 137 na stronie 1165 ) Parametr IBM MQ for Windows ma nazwę `ubbstxmn.mak`i jest przechowywany w katalogu przykładów IBM MQ .

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib64
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
 $(TUXDIR)\bin\buildclient -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

*Rysunek 137. Przykładowy plik makefile TUXEDO dla systemu IBM MQ for Windows*

Aby zbudować środowisko serwera i przykłady, wykonaj następujące kroki.

## Procedura

1. Utwórz katalog aplikacji, w którym ma zostać zbudowana przykładowa aplikacja, na przykład:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Skopiuj następujące pliki przykładowe z katalogu przykładów IBM MQ do katalogu aplikacji:
   - `amqstxmn.mak`
   - `amqstxen.env`
   - `ubbstxcn.cfg`
3. Zmodyfikuj każdy z tych plików, aby ustawić nazwy katalogów i ścieżki do katalogów używane w danej instalacji.
4. Zmodyfikuj plik `ubbstxcn.cfg` (patrz sekcja Rysunek 136 na stronie 1164 ). aby dodać szczegóły dotyczące nazwy komputera i menedżera kolejek, z którym ma zostać nawiązane połączenie.
5. Dodaj następujący wiersz do pliku TUXEDO *TUXDIR*`udataobj\rm`

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib64\mqmxa64.lib
   MQMDIR\tools\lib64\mqm.lib
   ```

   Nowy wpis musi zawierać jeden wiersz w pliku.
6. Ustaw następujące zmienne środowiskowe:

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. Utwórz urządzenie TLOG dla TUXEDO. W tym celu wywołaj komendę `tmadmin -c` i wprowadź następującą komendę:

   ```
   crdl -z APPDIR\TLOG
   ```

8. Ustaw katalog bieżący na *APPDIR*i wywołaj przykładowy plik makefile `amqstxmn.mak` jako zewnętrzny plik makefile projektu. Na przykład w programie Microsoft Visual C++ wydaj następującą komendę:

```
msvc amqstxmn.mak
```

   Wybierz opcję **build** (buduj), aby zbudować wszystkie programy przykładowe.

ALW *Przykładowy program serwera dla TUXEDO*

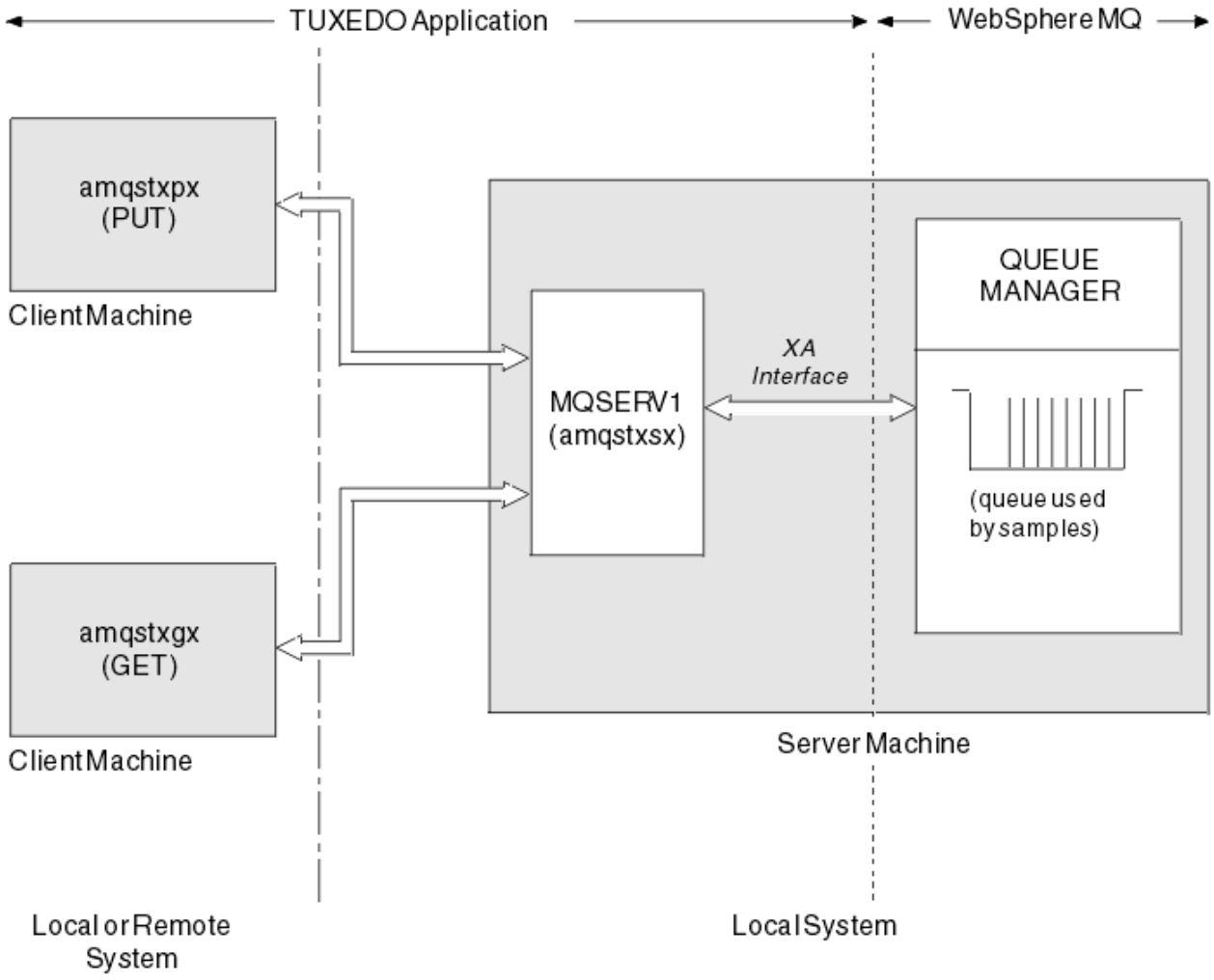
Przykładowy program serwera (amqstxsx) został zaprojektowany do działania z programami przykładowymi Put (amqstxpx.c) i Get (amqstxgx.c). Przykładowy program serwera jest uruchamiany automatycznie po uruchomieniu TUXEDO.

**Uwaga:** Przed uruchomieniem TUXEDO należy uruchomić menedżer kolejek.

Przykładowy serwer udostępnia dwie usługi TUXEDO: MPUT1 i MGET1:

- Usługa MPUT1 jest sterowana przez przykład PUT i używa MQPUT1 w punkcie synchronizacji w celu umieszczenia komunikatu w jednostce pracy sterowanej przez TUXEDO. Pobiera ona parametry QName i Message Text, które są dostarczane przez przykład PUT.
- Usługa MGET1 jest otwierana i zamykana za każdym razem, gdy pojawia się komunikat. Pobiera ona parametry QName i Message Text, które są dostarczane przez przykład GET.

Wszystkie komunikaty o błędach, kody przyczyny i komunikaty o statusie są zapisywane w pliku dziennika TUXEDO.




*Rysunek 138. Jak współpracują ze sobą przykłady TUXEDO*

ALW *Umieść przykładowy program dla TUXEDO*

Ten przykład umożliwia wielokrotne umieszczanie komunikatu w kolejce, w partiach, demonstrując zsynchronizowanie przy użyciu TUXEDO jako menedżera zasobów.

Przykładowy program serwera `amqstxsx` musi być uruchomiony, aby próba wykonania komendy put zakończyła się pomyślnie. Przykładowy program serwera nawiązuje połączenie z menedżerem kolejek i używa interfejsu XA. Aby uruchomić przykład, wpisz:

- `doputs -n queuename -b batchsize -c trancount -t message`

Na przykład:

- `doputs -n myqueue -b 5 -c 6 -t "Hello World"`

Spowoduje to umieszczenie 30 komunikatów w kolejce o nazwie `myqueue`, w sześciu partiach, z których każdy zawiera pięć komunikatów. Jeśli wystąpią jakiekolwiek problemy, zostanie wyświetlone zadanie wsadowe komunikatów, w przeciwnym razie zostaną one zatwierdzone.

Wszystkie komunikaty o błędach są zapisywane do pliku dziennika TUXEDO i do standardowego wyjścia błędów. Wszystkie kody przyczyny są zapisywane na standardowym wyjściu błędów.

ALW *Pobierz przykład dla TUXEDO*

Ten przykład umożliwia pobieranie komunikatów z kolejki w partiach.

Przykładowy program serwera `amqstxsx` musi być uruchomiony, aby przykład Get zakończył się pomyślnie. Przykładowy program serwera nawiązuje połączenie z menedżerem kolejek i używa interfejsu XA. Aby uruchomić przykład, wprowadź następującą komendę:

- `dogets -n queuename -b batchsize -c trancount`

Na przykład:

- `dogets -n myqueue -b 6 -c 4`

Spowoduje to wytworzenie 24 komunikatów z kolejki o nazwie `myqueue`, w sześciu partiach, z których każdy zawiera cztery komunikaty. Jeśli zostanie to wykonane po przykładzie umieszczania 30 komunikatów w katalogu `myqueue`, w katalogu `myqueue`będzie tylko sześć komunikatów. Liczba zadań wsadowych i ich wielkość mogą być różne w zależności od umieszczenia komunikatów i ich uzyskania.

Wszystkie komunikaty o błędach są zapisywane do pliku dziennika TUXEDO i do standardowego wyjścia błędów. Wszystkie kody przyczyny są zapisywane na standardowym wyjściu błędów.

## Windows *Korzystanie z wyjścia zabezpieczeń SSPI w systemie Windows*

W tej sekcji opisano sposób korzystania z programów obsługi wyjścia kanału SSPI w systemach Windows . Podany kod wyjścia ma dwa formaty: object i source.

### Kod obiektu

Plik z kodem obiektu nosi nazwę `amqrspin.dll`. Zarówno w przypadku klienta, jak i serwera, jest on instalowany jako standardowa część katalogu IBM MQ for Windows w folderze *MQ_INSTALLATION_PATH*`/exits/`*INSTALLATION_NAME* . Na przykład `C:\Program Files\IBM\MQ\exits\installation2`. Jest on ładowany jako standardowa procedura zewnętrzna. W definicji kanału można uruchomić dostarczone wyjście kanału zabezpieczeń i użyć usług uwierzytelniania.

W tym celu należy określić jedną z następujących opcji:

```
SCYEXIT('amqrspin(SCY_KERBEROS)')

SCYEXIT('amqrspin(SCY_NTLM)')
```

Aby zapewnić obsługę kanału zastrzeżonego, należy w kanale SVRCONN podać następujące informacje:

```
SCYDATA('remote_principal_name')
```

gdzie *nazwa_zdalnej_jednostki* ma postać *DOMENA**użytkownik*. Bezpieczny kanał jest ustanawiany tylko wtedy, gdy nazwa zdalnego użytkownika jest zgodna z nazwą *remote_principal_name*.

Aby użyć dostarczonych programów obsługi wyjścia kanału między systemami działającymi w domenie zabezpieczeń Kerberos , należy utworzyć **servicePrincipalName** dla menedżera kolejek.

## Kod źródłowy

Plik kodu źródłowego wyjścia ma nazwę `amqsspin.c`. Znajduje się on w katalogu `C:\Program Files\IBM\MQ\Tools\c\Samples`.

Jeśli kod źródłowy zostanie zmodyfikowany, należy ponownie skompilować zmodyfikowany kod źródłowy.

Jest on kompilowany i dowiązany w taki sam sposób, jak każde inne wyjście kanału dla odpowiedniej platformy, z wyjątkiem tego, że nagłówki SSPI muszą być dostępne w czasie kompilacji, a biblioteki bezpieczeństwa SSPI, wraz z zalecanymi powiązanymi bibliotekami, muszą być dostępne w czasie konsolidacji.

Przed wykonaniem poniższej komendy należy upewnić się, że w ścieżce jest dostępny katalog `cl.exe`, biblioteka Visual C++ oraz folder `include` . Na przykład:

```
cl /VERBOSE /LD /MT /Ipath_to_Microsoft_platform_SDK\include
/Ipath_to_IBM_MQ\tools\c\include amqsspin.c /DSECURITY_WIN32
-link /DLL /EXPORT:SCY_KERBEROS /EXPORT:SCY_NTLM STACK:8192
```

**Uwaga:** Kod źródłowy nie zawiera żadnych przepisów dotyczących śledzenia ani obsługi błędów. Jeśli kod źródłowy jest modyfikowany i używany, należy dodać własne procedury śledzenia i obsługi błędów.

### *Uruchamianie przykładów przy użyciu kolejek zdalnych*

Zdalne kolejkowanie można zademonstrować, uruchamiając przykłady w połączonych menedżerach kolejek.

Program amqscos0.tst udostępnia lokalną definicję kolejki zdalnej (SYSTEM.SAMPLE.REMOTE), który używa zdalnego menedżera kolejek o nazwie OTHER. Aby użyć tej przykładowej definicji, należy zmienić wartość OTHER na nazwę drugiego menedżera kolejek, który ma być używany. Należy również skonfigurować kanał komunikatów między dwoma menedżerami kolejek. Więcej informacji na ten temat zawiera sekcja Definiowanie kanałów.

Przykładowe programy żądania umieszczają własną nazwę lokalnego menedżera kolejek w polu *ReplyToQMgr* wysyłanych przez nie komunikatów. Przykłady zapytań i ustawiania wysyłają komunikaty odpowiedzi do kolejki i menedżera kolejek komunikatów, których nazwy znajdują się w polach *ReplyToQ* i *ReplyToQMgr* przetwarzanych przez nie komunikatów żądań.

### *Przykładowy program monitorowania kolejki klastra (AMQSCLM)*

W tym przykładzie użyto wbudowanych funkcji równoważenia obciążenia klastra IBM MQ do kierowania komunikatów do instancji kolejek, które mają przyłączone aplikacje konsumujące. Ten automatyczny kierunek zapobiega gromadzeniu się komunikatów w instancji kolejki klastra, do której nie jest przyłączona żadna aplikacja konsumująca.

## Przegląd

Można skonfigurować klaster, który ma więcej niż jedną definicję dla tej samej kolejki w różnych menedżerach kolejek. Ta konfiguracja zapewnia korzyści związane ze zwiększoną dostępnością i równoważeniem obciążenia. Jednak w produkcie IBM MQ nie ma możliwości dynamicznego modyfikowania dystrybucji komunikatów w klastrze na podstawie stanu przyłączonych aplikacji. Z tego powodu aplikacja konsumująca musi być zawsze przyłączona do każdej instancji kolejki, aby zapewnić przetwarzanie komunikatów.

Program przykładowy monitorowania kolejki klastra monitoruje stan przyłączonych aplikacji. Program dynamicznie dopasowuje wbudowaną konfigurację równoważenia obciążenia, aby kierować komunikaty do instancji kolejki klastrowej z przyłączonymi aplikacjami konsumującymi. W pewnych sytuacjach tego programu można użyć do złagodzenia potrzeby, aby aplikacja konsumująca była zawsze połączona z każdą instancją kolejki. Ponadto ponownie wysyła komunikaty, które zostały umieszczone w kolejce w instancji kolejki bez przyłączonych aplikacji konsumujących. Ponowne wysyłanie komunikatów umożliwia kierowanie komunikatów wokół aplikacji konsumującej, która jest tymczasowo zamknięta.

Program jest przeznaczony do użycia w przypadku, gdy aplikacje konsumujące są długo działającymi aplikacjami, a nie często podłączanymi i odłączanymi aplikacjami.

Przykładowy program monitorowania kolejki klastra jest skompilowanym programem wykonywalnym przykładowego pliku w języku C `amqsclma.c`.

Więcej informacji na temat klastrów i obciążeń można znaleźć w sekcji Używanie klastrów do zarządzania obciążeniem .

*AMQSCLM: projektowanie i planowanie użycia przykładu*
Informacje na temat sposobu działania programu przykładowego monitorowania kolejki klastra, wskazuje, że należy wziąć pod uwagę podczas konfigurowania systemu, w którym ma działać program przykładowy, oraz modyfikacje, które można wprowadzić w przykładowym kodzie źródłowym.

## Konstrukcja

Program przykładowy monitorowania kolejki klastra monitoruje lokalne kolejki klastrowe, które mają przyłączone aplikacje konsumujące. Program monitoruje kolejki określone przez użytkownika. Nazwa kolejki może być specyficzna, na przykład `APP.TEST01`, lub ogólna. Nazwy ogólne muszą być w formacie zgodnym z formatem PCF (Programmable Command Format). Przykładami nazw ogólnych są `APP.TEST*`lub `APP*`.

Każdy menedżer kolejek w klastrze, który jest właścicielem instancji kolejki lokalnej do monitorowania, wymaga połączenia z każdym programem przykładowym monitorowania kolejki klastra.

## Dynamiczne kierowanie komunikatów

Przykładowy program monitorowania kolejki klastra używa wartości **IPPROCS** (otwartej dla wejściowej liczby procesów) kolejki w celu określenia, czy ta kolejka ma konsumentów. Wartość większa niż 0 wskazuje, że do kolejki przyłączona jest co najmniej jedna aplikacja konsumująca. Takie kolejki są aktywne. Wartość 0 wskazuje, że kolejka nie ma przyłączonych programów konsumujących. Takie kolejki są nieaktywne.

W przypadku kolejki klastrowej z wieloma instancjami w klastrze IBM MQ używa właściwości priorytetu obciążenia klastra **CLWLPRTY** dla każdej instancji kolejki w celu określenia, do których instancji mają być wysyłane komunikaty. IBM MQ wysyła komunikaty do dostępnych instancji kolejki o najwyższej wartości **CLWLPRTY** .

Przykładowy program monitorowania kolejki klastra aktywuje kolejkę klastra, ustawiając lokalną wartość parametru **CLWLPRTY** na 1. Program dezaktywuje kolejkę klastra, ustawiając wartość parametru **CLWLPRTY** na 0.

Technologia klastrowa IBM MQ propaguje zaktualizowaną właściwość **CLWLPRTY** kolejki klastrowej do wszystkich odpowiednich menedżerów kolejek w klastrze. Na przykład składnia

- Menedżer kolejek z połączoną aplikacją, który umieszcza komunikaty w kolejce.
- Menedżer kolejek, który jest właścicielem kolejki lokalnej o tej samej nazwie w tym samym klastrze.

Propagacja jest wykonywana przy użyciu menedżerów kolejek pełnego repozytorium klastra. Nowe komunikaty dla kolejki klastra są kierowane do instancji o najwyższej wartości **CLWLPRTY** w klastrze.

## Przesyłanie komunikatów w kolejce

Dynamiczna modyfikacja wartości parametru **CLWLPRTY** wpływa na kierowanie nowych komunikatów. Ta dynamiczna modyfikacja nie ma wpływu na komunikaty już umieszczone w kolejce w instancji kolejki bez przyłączonych konsumentów ani na komunikaty, które były przesyłane przez mechanizm równoważenia obciążenia przed propagowaniem zmodyfikowanej wartości **CLWLPRTY** w klastrze. W wyniku tego komunikaty pozostają w dowolnej nieaktywnej kolejce i nie są przetwarzane przez aplikację konsumującą. Aby rozwiązać ten problem, przykładowy program monitorowania kolejki klastra może pobierać komunikaty z kolejki lokalnej bez konsumentów i wysyłać te komunikaty do zdalnych instancji tej samej kolejki, do której przyłączeni są konsumenci.

Program przykładowy monitorowania kolejki klastra przesyła komunikaty z nieaktywnej kolejki lokalnej do jednej lub większej liczby aktywnych kolejek zdalnych, pobierając komunikaty (za pomocą komendy **MQGET** ). i umieszczanie komunikatów (za pomocą **MQPUT** ) do tej samej kolejki klastrowej. Powoduje to, że funkcja zarządzania obciążeniem klastra IBM MQ wybiera inną instancję docelową w oparciu o wyższą wartość CLWLPRTY niż instancja kolejki lokalnej. Trwałość komunikatu i kontekst są zachowywane podczas przesyłania komunikatów. Kolejność komunikatów i wszystkie opcje powiązania nie są zachowywane.

## Planowanie

Przykładowy program monitorowania kolejki klastra modyfikuje konfigurację klastra, gdy występuje zmiana w połączeniach aplikacji konsumujących. Modyfikacje są przesyłane z menedżerów kolejek, w których program przykładowy monitorowania kolejki klastra monitoruje kolejki, do menedżerów kolejek pełnego repozytorium w klastrze. Menedżery kolejek pełnego repozytorium przetwarzają aktualizacje konfiguracji i ponownie je przetwarzają do wszystkich odpowiednich menedżerów kolejek w klastrze. Odpowiednie menedżery kolejek obejmują te menedżery kolejek, które są właścicielami kolejek klastrowych o takiej samej nazwie (w których działa instancja programu przykładowego monitorowania kolejki klastra), oraz wszystkie menedżery kolejek, w których aplikacja otworzyła kolejkę klastra w celu umieszczenia w niej komunikatów w ciągu ostatnich 30 dni.

Zmiany są przetwarzane asynchronicznie w klastrze. Dlatego po każdej zmianie różne menedżery kolejek w klastrze mogą mieć różne widoki konfiguracji przez pewien czas.

Przykładowy program monitorowania kolejki klastra jest odpowiedni tylko dla systemów, w których aplikacje konsumujące są rzadko podłączane lub odłączane, na przykład aplikacje długotrwałe. W przypadku monitorowania systemów, w których aplikacje konsumujące są przyłączone tylko przez krótki czas, opóźnienie związane z dystrybucją aktualizacji konfiguracji może spowodować, że menedżery kolejek w klastrze będą mieć niepoprawny widok kolejek, w których podłączono konsumentów. To opóźnienie może spowodować niepoprawne kierowanie komunikatów.

W przypadku monitorowania wielu kolejek stosunkowo niska szybkość zmian w przyłączonych konsumentach we wszystkich kolejkach może zwiększyć ruch w konfiguracji klastra w klastrze. Zwiększenie ruchu w konfiguracji klastra może spowodować nadmierne obciążenie co najmniej jednego z następujących menedżerów kolejek.

- Menedżery kolejek, w których działa przykładowy program monitorowania kolejki klastra
- Menedżery kolejek pełnego repozytorium
- Menedżer kolejek z połączoną aplikacją, który umieszcza komunikaty w kolejce
- Menedżer kolejek, który jest właścicielem kolejki lokalnej o tej samej nazwie w tym samym klastrze

Należy ocenić wykorzystanie procesora w menedżerach kolejek pełnego repozytorium. Dodatkowe wykorzystanie procesora jest widoczne jako ruch komunikatów w kolejce pełnego repozytorium SYSTEM.CLUSTER.COMMAND.QUEUE. Jeśli w tej kolejce znajdują się komunikaty, oznacza to, że menedżery kolejek pełnego repozytorium nie są w stanie nadążyć za szybkością zmiany konfiguracji klastra w systemie.

Jeśli program przykładowy monitorowania kolejek klastra monitoruje wiele kolejek, program przykładowy i menedżer kolejek wykonują dużo pracy. Ta praca jest wykonywana, nawet jeśli nie ma zmian

w przyłączonych konsumentach. Argument **-i** można zmodyfikować, aby zmniejszyć użycie procesora przez przykładowy program w systemie lokalnym, zmniejszając częstotliwość cyklu monitorowania.

Aby ułatwić wykrywanie nadmiernej aktywności, przykładowy program monitorujący kolejkę klastra zgłasza średni czas przetwarzania w okresie odpytywania, czas przetwarzania oraz liczbę zmian w konfiguracji. Raporty są dostarczane w postaci komunikatu informacyjnego, **CLM0045I**, co 30 minut lub co 600 odstępów czasu odpytywania, w zależności od tego, co nastąpi wcześniej.

## Wymagania dotyczące użycia monitorowania kolejki klastra

Program przykładowy monitorowania kolejki klastra ma wymagania i ograniczenia. Udostępniony przykładowy kod źródłowy można zmodyfikować, aby zmienić niektóre z tych ograniczeń w sposobie jego używania. Przykłady wymienione w tej sekcji zawierają szczegółowe modyfikacje, które można wprowadzić.

- Przykładowy program monitorowania kolejki klastra jest przeznaczony do monitorowania kolejek, w których aplikacje konsumujące są przyłączone lub nie są przyłączone. Jeśli system wykorzystuje aplikacje, które są często podłączane i odłączane, przykładowy program może generować nadmierną aktywność konfiguracji klastra w całym klastrze. Może to mieć wpływ na wydajność menedżerów kolejek w klastrze.
- Przykładowy program monitorowania kolejki klastra zależy od bazowego systemu IBM MQ i technologii klastrowej. Liczba monitorowanych kolejek, częstotliwość monitorowania i częstotliwość zmiany stanu każdej kolejki mają wpływ na obciążenie całego systemu. Czynniki te muszą być brane pod uwagę podczas wybierania kolejek, które mają być monitorowane, oraz okresu odpytywania monitorowania.
- Instancja programu przykładowego monitorowania kolejki klastra musi być połączona z każdym menedżerem kolejek w klastrze, który jest właścicielem instancji kolejki, która ma być monitorowana. Nie jest konieczne łączenie programu przykładowego z menedżerami kolejek w klastrze, które nie są właścicielami kolejek.
- Przykładowy program monitorowania kolejki klastra musi być uruchomiony z odpowiednią autoryzacją, aby uzyskać dostęp do wszystkich wymaganych zasobów IBM MQ . Na przykład składnia
  - Menedżer kolejek, z którym ma zostać nawiązane połączenie
  - SYSTEM SYSTEM.ADMIN.COMMAND.QUEUE
  - Wszystkie kolejki, które mają być monitorowane podczas przesyłania komunikatów
- Serwer komend musi być uruchomiony dla każdego menedżera kolejek z połączonym programem przykładowym monitorowania kolejki klastra.
- Każda instancja programu przykładowego monitorowania kolejki klastra wymaga wyłącznego użycia lokalnej (nieklastrowej) kolejki w menedżerze kolejek, z którym jest połączona. Ta kolejka lokalna jest używana do sterowania programem przykładowym i odbierania komunikatów odpowiedzi z zapytań wysłanych do serwera komend menedżera kolejek.
- Wszystkie kolejki, które mają być monitorowane przez pojedynczą instancję programu przykładowego monitorowania kolejki klastra, muszą znajdować się w tym samym klastrze. Jeśli menedżer kolejek ma kolejki w wielu klastrach, które wymagają monitorowania, wymaganych jest wiele instancji programu przykładowego. Każda instancja wymaga lokalnej kolejki dla komunikatów sterowania i odpowiedzi.
- Wszystkie kolejki, które mają być monitorowane, muszą znajdować się w pojedynczym klastrze. Kolejki skonfigurowane do używania listy nazw klastrów nie są monitorowane.
- Włączenie przesyłania komunikatów z nieaktywnych kolejek jest opcjonalne. Dotyczy to wszystkich kolejek monitorowanych przez instancję programu przykładowego monitorowania kolejki klastra. Jeśli tylko podzbiór monitorowanych kolejek wymaga włączenia przesyłania komunikatów, potrzebne są dwie instancje przykładowego programu monitorującego kolejki klastra. Jeden przykładowy program ma włączoną funkcję przesyłania komunikatów, a drugi wyłączony. Każda instancja programu przykładowego wymaga lokalnej kolejki dla komunikatów sterowania i odpowiedzi.
- Funkcja równoważenia obciążenia klastra IBM MQ domyślnie wysyła komunikaty do instancji kolejek klastrowych znajdujących się w tym samym menedżerze kolejek, z którym połączona jest aplikacja

umieszczająca. Ta opcja musi być wyłączona, gdy kolejka lokalna jest nieaktywna w następujących okolicznościach:

  - Umieszczanie aplikacji łączących się z menedżerami kolejek, które są właścicielami instancji nieaktywnej kolejki, która jest monitorowana
  - Komunikaty w kolejce są przesyłane z nieaktywnych kolejek do aktywnych kolejek.

  Lokalną preferencję równoważenia obciążenia w kolejce można wyłączyć statycznie, ustawiając wartość CLWLUSEQ na ANY. W tej konfiguracji komunikaty umieszczane w kolejkach lokalnych są dystrybuowane do instancji kolejek lokalnych i zdalnych w celu zrównoważenia obciążenia, nawet jeśli istnieją lokalne aplikacje konsumujące. Alternatywnie przykładowy program monitorowania kolejki klastra można skonfigurować w taki sposób, aby tymczasowo ustawić wartość parametru **CLWLUSEQ** na ANY , gdy kolejka nie ma przyłączonych konsumentów, co spowoduje, że tylko komunikaty lokalne będą kierowane do lokalnych instancji kolejki, gdy ta kolejka jest aktywna.
- System IBM MQ i aplikacje nie mogą używać **CLWLPRTY** do monitorowania kolejek ani używanych kanałów. W przeciwnym razie działania przykładowego programu monitorowania kolejki klastra w atrybutach kolejki **CLWLPRTY** mogą mieć niepożądane skutki.
- Przykładowy program monitorujący kolejkę klastra rejestruje informacje o środowisku wykonawczym w zestawie plików raportu. Katalog, w którym mają być przechowywane te raporty, jest wymagany, a przykładowy program monitorowania kolejki klastra musi mieć uprawnienia do zapisu w tym katalogu.

*AMQSCLM: przygotowywanie i uruchamianie przykładu*
Przykład monitorowania kolejki klastra może być uruchomiony lokalnie z menedżerem kolejek lub jako klient połączony za pośrednictwem kanału. Przykład powinien być uruchamiany za każdym razem, gdy menedżer kolejek jest uruchomiony lokalnie, można go skonfigurować jako usługę menedżera kolejek w taki sposób, aby automatycznie uruchamiał i zatrzymywał przykład z menedżerem kolejek.

## Zanim rozpoczniesz

Przed uruchomieniem przykładu monitorowania kolejki klastra należy wykonać następujące kroki.

1. Utwórz kolejkę roboczą w każdym menedżerze kolejek na potrzeby wewnętrznego użycia przykładu.

   Każda instancja przykładu wymaga lokalnej kolejki innej niż klastrowa do wyłącznego użytku wewnętrznego. Można wybrać nazwę kolejki. W przykładzie użyto nazwy AMQSCLM.CONTROL.QUEUE. Na przykład w systemie Windowsmożna utworzyć tę kolejkę za pomocą następującej komendy **MQSC** :

   ```
   DEFINE QLOCAL(AMQSCLM.CONTROL.QUEUE)
   ```

   Wartości **MAXDEPTH** i **MAXMSGL** można pozostawić jako domyślne.
2. Utwórz katalog dla dzienników błędów i komunikatów informacyjnych.

   Przykład zapisuje komunikaty diagnostyczne w plikach raportów. Należy wybrać katalog, w którym mają być przechowywane pliki. Na przykład w systemie Windowsmożna utworzyć katalog za pomocą następującej komendy:

   ```
   mkdir C:\AMQSCLM\rpts
   ```

   Pliki raportów utworzone przez przykład mają następującą konwencję nazewnictwa:

   ```
   QmgrName.ClusterName.RPT0n.LOG
   ```
3. (Opcjonalnie) Zdefiniuj próbkę monitorowania kolejki klastra jako usługę IBM MQ .

   Aby monitorować kolejki, próbka musi być zawsze uruchomiona. Aby upewnić się, że przykład monitorowania kolejki klastra jest zawsze uruchomiony, można zdefiniować przykład jako usługę menedżera kolejek. Zdefiniowanie przykładu jako usługi oznacza, że aplikacja AMQSCLM jest

uruchamiana podczas uruchamiania menedżera kolejek. Poniższego przykładu można użyć do zdefiniowania próbki monitorowania kolejki klastra jako usługi systemu IBM MQ .

```
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')
```

| Definicja | Opis |
|---|---|
| **service** | Określa nazwę usługi. Można wybrać nazwę usługi. |
| **descr** | Określa tekstowy opis usługi. |
| **control** | Wskazuje, że usługa jest uruchamiana i zatrzymywana w tym samym czasie co menedżer kolejek. |
| **servtype** | Wskazuje, że obiekt usługi serwera, czyli tylko jedna instancja, może być wykonywany w danym momencie dla tego menedżera kolejek. |
| **startcmd** | Określa położenie i nazwę programu. |
| **startarg** | Określa argumenty próbki. Należy zwrócić uwagę na użycie *+ QMNAME +*. Nazwa menedżera kolejek jest zastępowana automatycznie. |
| **stdout** | Pełna nazwa pliku, do którego przekierowywane jest wyjście standardowe. Przykład zapisuje w tym pliku tylko komunikaty potwierdzające, że przykład został zakończony. Przykład wykonuje tę czynność, ponieważ plik standardowego wyjścia błędów został już zamknięty we wcześniejszym etapie procesu zakończenia próby. |
| **stderr** | Pełna nazwa pliku, do którego przekierowywane są standardowe wyjście błędów. Przykład zapisuje w pliku standardowego wyjścia błędów wszystkie komunikaty o błędach przed zakończeniem próbkowania. |

## O tym zadaniu

To zadanie umożliwia uruchamianie i zatrzymywanie próbki monitorowania kolejki klastra na różne sposoby. Umożliwia również uruchomienie przykładu w trybie, w którym generowane są pliki raportów zawierające informacje statystyczne o monitorowanych kolejkach.

Program przykładowy można uruchomić za pomocą następującej komendy.

```
AMQSCLM -m QMgrName -c ClusterName (-q QNameMask| -f QListFile) -r MonitorQName
[-l ReportDir] [-t] [-u ActiveVal] [-i Interval] [-d] [-s] [-v]
```

Tabela zawiera listę argumentów, których można użyć z próbką monitorowania kolejki klastra, wraz z dodatkowymi informacjami o każdej z nich.

| Argument | Zmienna | Dalsze informacje |
|---|---|---|
| **-m** | **QMgrName** | Menedżer kolejek, który ma być monitorowany. |
| **-c** | **ClusterName** | Klaster zawierający kolejki do monitorowania. |
| **-q** | **QNameMask** | Kolejka lub kolejki do monitorowania. Kończący * monitoruje wszystkie kolejki, których nazwy są zgodne z zerem lub większą liczbą znaków końcowych. |
| **-f** | **QListFile** | Pełna ścieżka i nazwa pliku zawierającego listę nazw kolejek lub masek nazw kolejek do monitorowania. Plik musi zawierać jedną nazwę kolejki/ maskę w wierszu. Można podać wartość **-q** lub **-f**, ale nie obie te wartości. |
| **-r** | **MonitorQName** | Kolejka lokalna używana wyłącznie przez przykład. |
| **-l** | **ReportDir** | Ścieżka do katalogu, w którym mają być zapisywane zarejestrowane komunikaty informacyjne w zestawie zawijania[9]plików raportów. |
| **-t** | | (Opcjonalnie) Włącza przesyłanie komunikatów z kolejki z nieaktywnych kolejek lokalnych do kolejek aktywnych. Jeśli ta opcja nie jest włączona, tylko nowe komunikaty wchodzące do klastra są dynamicznie kierowane do aktywnych instancji kolejki. |
| **-u** | **ActiveVal** | (Opcjonalnie) Automatycznie przełącza właściwość **CLWLUSEQ** monitorowanej instancji kolejki na wartość ANY , gdy jest ona nieaktywna, i na wartość **ActiveVal** , gdy jest aktywna. Parametr **ActiveVal** może mieć wartość LOCAL lub QMGR. Jeśli ten argument nie jest ustawiony w systemie, w którym aplikacje są umieszczane w tym samym menedżerze kolejek lub w którym włączona jest funkcja przesyłania komunikatów, monitorowane kolejki muszą mieć wartość **CLWLUSEQ** ANYlub QMGR , a menedżer kolejek musi mieć wartość ANY. |
| **-i** | **Interval** | (Opcjonalnie) Przedział czasu (w sekundach), w którym monitor sprawdza kolejki. Wartością domyślną jest 300 sekund (5 minut). |
| **-d** | | (Opcjonalne) Włącza dodatkowe dane diagnostyczne. Dane wyjściowe debugowania mogą być przydatne podczas początkowego konfigurowania systemu lub podczas pracy z przykładowym kodem. |
| **-s** | | (Opcjonalne) Włącza minimalny wynik statystyczny na przedział czasu. |
| **-v** | | (Opcjonalnie) oprócz plików raportów należy również rejestrować informacje o raportach w produkcie `standard out`. |

Przykłady listy argumentów:

```
-m QMGR1 -c CLUS1   -f c:\QList.txt -r CLMQ -l c:\amqsclm\rpts -s
-m QMGR2 -c CLUS1   -q ABC* -r CLMQ -l c:\amqsclm\rpts -i 600
-m QMGR1 -c CLUSDEV -q QUEUE.* -r CLMQ -l c:\amqsclm\rpts -t -u QMGR -d
```

Przykładowy plik listy kolejek:

```
Q1
QUEUE.*
```

[9] Dla każdej kombinacji menedżera kolejek i kolejki generowany jest plik dziennika o stałej wielkości, który po zapełnieniu jest nadpisywany. Program rejestrujący zawsze zapisuje dane w tym samym pliku, a także zachowuje dwie poprzednie wersje pliku.

```
ABC
ABD
```

## Procedura

1. Uruchom przykład monitorowania kolejki klastra. Przykład można uruchomić w jeden z następujących sposobów:
   - Użyj wiersza komend z odpowiednimi autoryzacjami użytkowników.
   - Użyj komendy MQSC **START SERVICE** , jeśli przykład jest skonfigurowany jako usługa systemu IBM MQ .

   Lista argumentów jest taka sama w obu przypadkach.

   Przykład nie uruchamia monitorowania kolejek przez 10 sekund po zainicjowaniu programu. To opóźnienie umożliwia aplikacjom konsumuającym łączenie się najpierw z monitorowanymi kolejkami, zapobiegając niepotrzebnym zmianom aktywnego stanu kolejki.
2. Zatrzymaj przykład monitorowania kolejki klastra. Przykład jest automatycznie zatrzymywany po zatrzymaniu, zatrzymaniu, wyciszeniu lub zerwaniu połączenia z menedżerem kolejek. Istnieją sposoby zatrzymania przykładu bez kończenia menedżera kolejek:
   - Skonfiguruj lokalną kolejkę używaną wyłącznie przez przykład, aby wyłączyć funkcję Get.
   - Wyślij komunikat z wartością **CorrelId** równą `"STOP CLUSTER MONITOR\0\0\0\0"`do kolejki lokalnej używanej wyłącznie przez przykład.
   - Przerwij przykładowy proces. Może to spowodować utratę nietrwałych komunikatów przesyłanych do aktywnych kolejek. Może to również spowodować, że kolejka lokalna używana przez próbkę będzie otwarta przez określoną liczbę sekund po zakończeniu. Ta sytuacja uniemożliwia natychmiastowe uruchomienie nowej instancji próbki monitorowania kolejki klastra.

   Jeśli przykład został uruchomiony jako usługa systemu IBM MQ , opcja **STOP SERVICE** nie ma żadnego wpływu. Istnieje możliwość użycia jednej z metod zakończenia opisanych jako skonfigurowany mechanizm produktu **STOP SERVICE** w menedżerze kolejek.

## Co dalej

Sprawdź status próbki.

Jeśli raportowanie jest włączone, można przejrzeć pliki raportów pod kątem statusu. Użyj następującej komendy, aby przejrzeć bieżący plik raportu:

```
QMgrName.ClusterName.RPT01.LOG
```

Aby przejrzeć starsze pliki raportów, użyj następujących komend:

```
QMgrName.ClusterName.RPT02.LOG
QMgrName.ClusterName.RPT03.LOG
```

Maksymalna wielkość plików raportów wynosi około 1 MB. Po zapełnieniu pliku RPT01 tworzony jest nowy plik RPT01 . Nazwa starego pliku RPT01 została zmieniona na RPT02. Nazwa pliku RPT02 została zmieniona na RPT03. Stara wartość RPT03 jest odrzucana.

Przykład tworzy komunikaty informacyjne w następujących sytuacjach:

- podczas uruchamiania
- w momencie zakończenia
- gdy oznacza kolejkę **ACTIVE** lub **INACTIVE**
- gdy ponownie kolejkowane są komunikaty z nieaktywnej kolejki do aktywnej instancji lub instancji

W przykładzie tworzony jest komunikat o błędzie *CLMnnnnE* w celu zgłoszenia problemu, który wymaga uwagi.

Co 30 minut próbka raportuje średni czas przetwarzania na okres odpytywania i czas przetwarzania. Informacje te są przechowywane w komunikacie CLM0045I.

Gdy komunikaty statystyczne są włączone **-s**, przykład raportuje następujące informacje statystyczne na temat każdego sprawdzenia kolejki:

- Czas przetwarzania kolejek (w milisekundach)
- Liczba sprawdzonych kolejek
- Liczba wprowadzonych aktywnych/nieaktywnych zmian
- Liczba przesłanych wiadomości

Informacje te są zgłaszane w komunikacie CLM0048I.

Pliki raportów mogą szybko rosnąć w trybie debugowania i szybko się zawijać. W takiej sytuacji może zostać przekroczony limit wielkości 1 MB dla poszczególnych plików.

*AMQSCLM: rozwiązywanie problemów*
Poniższe sekcje zawierają informacje na temat scenariuszy, które mogą wystąpić podczas korzystania z przykładu. Dostępne są informacje o potencjalnych wyjaśnieniach scenariusza oraz opcje dotyczące sposobu jego rozwiązania.

## Scenariusz: AMQSCLM nie jest uruchamiany

**Potencjalny opis:** Niepoprawna składnia.

**Działanie:** Sprawdź dane wyjściowe standardowego wyjścia błędów pod kątem poprawnej składni.

**Potencjalny opis:** Menedżer kolejek jest niedostępny.

**Działanie:** Sprawdź plik raportu pod kątem komunikatu o identyfikatorze CLM0010E.

**Potencjalna objaśnienie:** Nie można otworzyć lub utworzyć pliku lub plików raportu.

**Działanie:** Podczas inicjowania sprawdź komunikaty o błędach w standardowym wyjściu błędów.

## Scenariusz: AMQSCLM nie zmienia kolejki na ACTIVE lub INACTIVE

**Potencjalny opis:** Kolejka nie znajduje się na liście kolejek, które mają być monitorowane.

**Działanie:** Sprawdź wartości parametrów **-q** i **-f** .

**Potencjalny opis:** Kolejka nie jest kolejką lokalną we właściwym klastrze.

**Działanie:** Sprawdź, czy kolejka jest lokalna i czy znajduje się w poprawnym klastrze.

**Potencjalny problem:** Aplikacja AMQSCLM nie jest uruchomiona dla tego menedżera kolejek i klastra.

**Działanie:** Uruchom AMQSCLM dla odpowiedniego menedżera kolejek i klastra.

**Potencjalnym wyjaśnieniem:** Kolejka jest pozostawiona jako NIEAKTYWNA, **CLWLPRTY** = 0, ponieważ nie ma konsumentów. Można również pozostawić wartość ACTIVE **CLWLPRTY** > =1, ponieważ zawiera co najmniej 1 konsumenta.

**Działanie:** Sprawdź, czy aplikacje konsumujące są przyłączone do kolejki.

**Potencjalny opis:** Serwer komend menedżera kolejek nie jest uruchomiony.

**Działanie:** Sprawdź pliki raportów pod kątem błędów.

## Scenariusz: komunikaty nie są kierowane do kolejek INACTIVE

**Potencjalny Objaśnienie:** Komunikaty są umieszczane bezpośrednio w menedżerze kolejek, który jest właścicielem nieaktywnej kolejki, a wartość parametru **CLWLUSEQ** kolejki nie jest równa ANY, a argument **-u** nie jest używany dla AMQSCLM.

**Działanie:** Sprawdź wartość **CLWLUSEQ** odpowiedniego menedżera kolejek lub upewnij się, że argument **-u** jest używany dla AMQSCLM.

**Potencjalny opis:** Brak aktywnych kolejek w żadnym z menedżerów kolejek. Komunikaty są równomiernie rozkładane na wszystkie nieaktywne kolejki, dopóki kolejka nie stanie się aktywna.

**Działanie:** Sprawdź status kolejek we wszystkich menedżerach kolejek.

**Potencjalny Objaśnienie:** Komunikaty są umieszczane w innym menedżerze kolejek w klastrze niż ten, który jest właścicielem nieaktywnej kolejki, a zaktualizowana wartość parametru **CLWLPRTY** wynosząca 0 nie jest propagowana do menedżera kolejek aplikacji umieszczającej.

**Działanie:** Sprawdź, czy kanały klastra między monitorowanym menedżerem kolejek i menedżerem kolejek repozytorium pełnego są uruchomione. Sprawdź, czy kanały między umieszczaniem menedżera kolejek i menedżerem kolejek repozytorium pełnego są uruchomione. Sprawdź dzienniki błędów monitorowanych, umieszczanych i pełnych menedżerów kolejek repozytorium.

**Potencjalna przyczyna:** Instancje kolejek zdalnych są aktywne (CLWLPRTY=1), ale nie można kierować komunikatów do tych instancji kolejek, ponieważ kanał nadawczy klastra z lokalnego menedżera kolejek nie jest uruchomiony.

**Działanie:** Sprawdź status kanałów nadawczych klastra od lokalnego menedżera kolejek do zdalnego menedżera kolejek (lub menedżerów) z aktywną instancją kolejki.

## Scenariusz: AMQSCLM nie przesyła komunikatów z nieaktywnej kolejki

**Potencjalny opis:** Przesyłanie komunikatów nie jest włączone ( **-t** ).

**Działanie:** Upewnij się, że przesyłanie komunikatów jest włączone ( **-t** ).

**Potencjalny opis:** Kolejka nie znajduje się na liście kolejek, które mają być monitorowane.

**Działanie:** Sprawdź wartości parametrów **-q** i **-f** .

**Potencjalna objaśnienie:** Komenda AMQSCLM nie jest uruchomiona dla tego lub innych menedżerów kolejek w klastrze, które są właścicielami instancji tej samej kolejki.

**Działanie:** Uruchom AMQSCLM.

**Potencjalny opis:** Kolejka ma **CLWLUSEQ** = LOCAL lub **CLWLUSEQ** = QMGR, a argument **-u** nie jest ustawiony.

**Działanie:** Ustaw parametr **-u** lub zmień konfigurację kolejki lub menedżera kolejek na ANY.

**Potencjalny objaśnienie:** Brak aktywnych instancji kolejki w klastrze.

**Działanie:** Sprawdź, czy istnieją instancje kolejki o wartości **CLWLPRTY** 1 lub większej.

**Potencjalny opis:** Instancje kolejek zdalnych mają konsumentów ( **IPPROCS** > = 1), ale są nieaktywne w tych menedżerach kolejek ( **CLWLPRTY** = 0), ponieważ aplikacja AMQSCLM nie monitoruje tych instancji zdalnych.

**Działanie:** Upewnij się, że aplikacja AMQSCLM działa w tych menedżerach kolejek i/lub że kolejka znajduje się na liście kolejek, które mają być monitorowane, sprawdzając wartości parametrów **-q** i **-f** .

**Potencjalne wyjaśnienie:** Instancje kolejek zdalnych są aktywne ( **CLWLPRTY** = 1), ale są postrzegane jako nieaktywne w lokalnym menedżerze kolejek ( **CLWLPRTY** = 0). Ta sytuacja jest spowodowana tym, że zaktualizowana wartość parametru **CLWLPRTY** nie jest propagowana do tego menedżera kolejek.

**Działanie:** Upewnij się, że zdalne menedżery kolejek są połączone z co najmniej jednym menedżerem kolejek repozytorium pełnego w klastrze. Upewnij się, że menedżery kolejek repozytorium pełnego działają poprawnie. Sprawdź, czy kanały między menedżerami kolejek repozytorium pełnego i monitorowanymi menedżerami kolejek są uruchomione.

**Potencjalne wyjaśnienie:** Komunikaty nie są zatwierdzane, dlatego nie można ich pobrać.

**Działanie:** Sprawdź, czy aplikacja wysyłająca działa poprawnie.

**Potencjalnym wyjaśnieniem:** Aplikacja AMQSCLM nie ma dostępu do kolejki lokalnej, w której znajdują się komunikaty.

**Działanie:** Sprawdź, czy AMQSCLM działa jako użytkownik z wystarczającymi uprawnieniami, aby uzyskać dostęp do kolejki.

**Potencjalny opis:** Serwer komend menedżera kolejek nie jest uruchomiony.

**Działanie:** Uruchom serwer komend menedżera kolejek.

**Potencjalny błąd:** AMQSCLM napotkał błąd.

**Działanie:** Sprawdź pliki raportów pod kątem błędów.

**Potencjalna objaśnienie:** Instancje kolejek zdalnych są aktywne (CLWLPRTY=1), ale nie można przesłać komunikatów do tych instancji kolejek, ponieważ kanał nadawczy klastra z lokalnego menedżera kolejek nie jest uruchomiony. Często towarzyszy temu ostrzeżenie CLM0030W w dzienniku raportu amqsclm.

**Działanie:** Sprawdź status kanałów nadawczych klastra od lokalnego menedżera kolejek do zdalnego menedżera kolejek (lub menedżerów) z aktywną instancją kolejki.

## ALW *Przykładowy program do wyszukiwania punktów końcowych połączenia (Connection Endpoint Lookup-CEPL)*

Przykład wyszukiwania punktu końcowego połączenia IBM MQ udostępnia prosty, ale wydajny moduł wyjścia, który umożliwia użytkownikom IBM MQ pobieranie definicji połączeń z repozytorium LDAP, takiego jak Tivoli Directory Server.

Aby można było używać CEPL, musi być zainstalowany klient Tivoli Directory Server v6.3 .

Do korzystania z tego przykładu wymagana jest wiedza na temat administrowania produktem IBM MQ na obsługiwanych platformach.

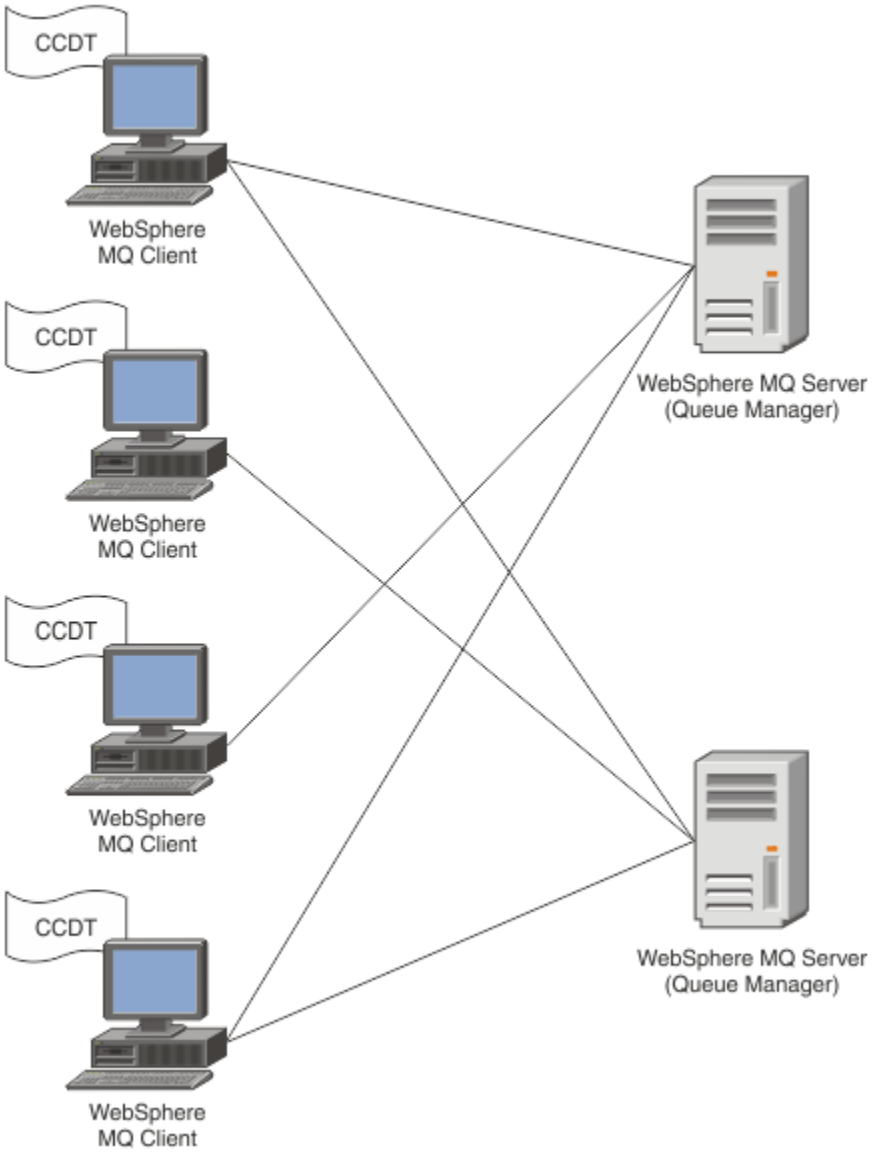
### Windows Linux AIX *Wprowadzenie*

Skonfiguruj repozytorium globalne, na przykład katalog LDAP (Lightweight Directory Access Protocol), w którym będą przechowywane definicje połączeń klientów, aby ułatwić konserwację i administrowanie.

Używanie aplikacji klienckiej IBM MQ do nawiązywania połączenia z menedżerem kolejek za pośrednictwem tabeli definicji połączenia klienta (CCDT).

CCDT jest tworzony za pomocą standardowego interfejsu IBM MQ MQSC Administration. Użytkownik musi być połączony z menedżerem kolejek w celu utworzenia definicji połączenia klienta, nawet jeśli dane zawarte w definicji nie są ograniczone do menedżera kolejek.

Wygenerowany plik CCDT musi być ręcznie rozproszony między komputerami klienckimi i aplikacjami.

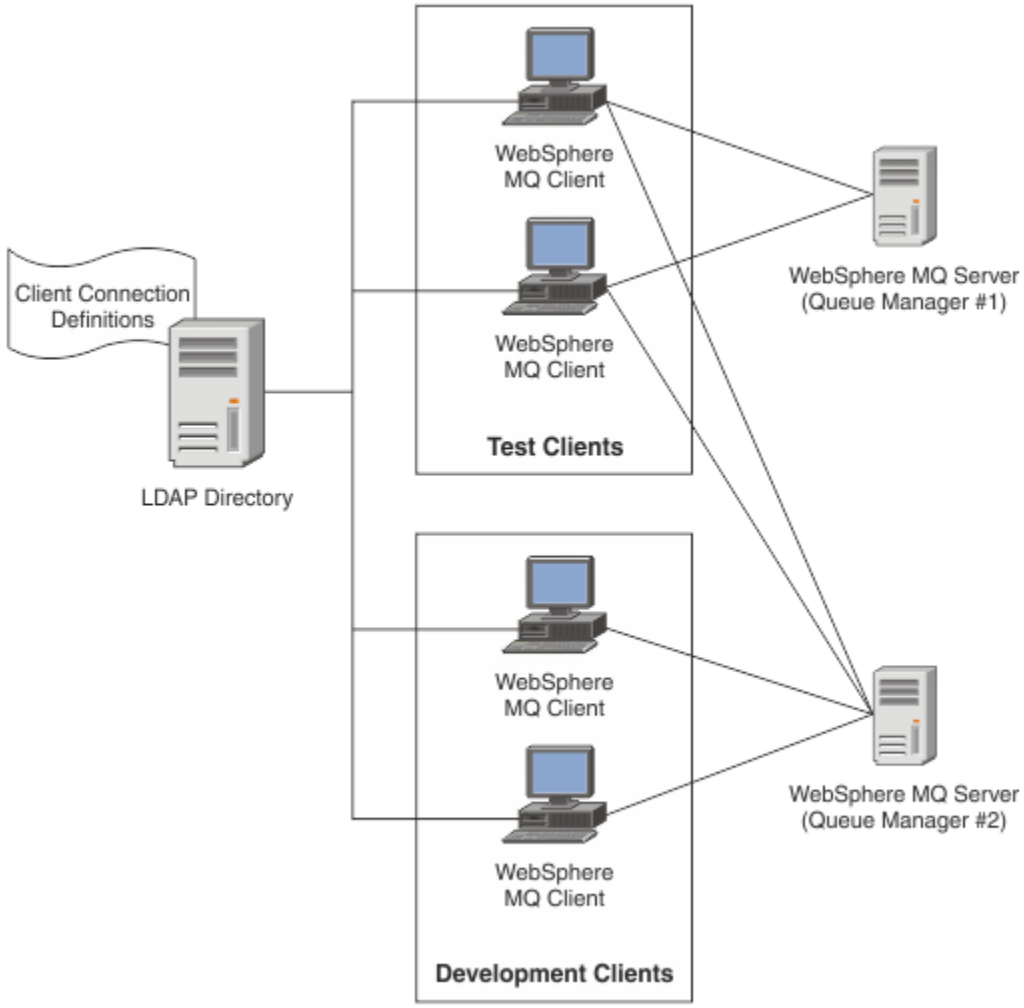



Plik CCDT musi być dystrybuowany do każdego klienta IBM MQ . Tam, gdzie tysiące klientów mogą istnieć lokalnie lub globalnie, wkrótce trudno będzie je utrzymać i administrować nimi. Potrzebne jest bardziej elastyczne podejście, aby zapewnić, że każdy klient ma dostęp do poprawnych definicji klientów.

Jednym z takich rozwiązań jest przechowywanie definicji połączeń klienta w repozytorium globalnym, takim jak katalog LDAP (Lightweight Directory Access Protocol). Katalog LDAP może również udostępniać dodatkowe funkcje ochrony, indeksowania i wyszukiwania, umożliwiając w ten sposób każdemu klientowi dostęp tylko do tych definicji połączeń, które się do niego dotyczą.

Katalog LDAP można skonfigurować w taki sposób, aby tylko konkretne definicje były dostępne dla określonych grup użytkowników. Na przykład klienty testowe mogą uzyskiwać dostęp zarówno do menedżera kolejek #1 , jak i do produktu #2, natomiast

klienty programistyczne mogą uzyskiwać dostęp tylko do menedżera kolejek #2 .




Moduł wyjścia może wyszukać repozytorium LDAP, na przykład IBM Tivoli Directory Server, w celu pobrania definicji kanałów. Za pomocą tych definicji połączeń aplikacja kliencka IBM MQ może nawiązać połączenie z menedżerem kolejek.

Moduł wyjścia jest modułem wyjścia przed nawiązaniem połączenia, który umożliwia uzyskanie definicji kanału podczas wywołania MQCONN/MQCONNX z repozytorium LDAP.

Moduł i schemat wyjścia mogą być implementowane przez:

- Klienci, którzy już zbudowali bazę umiejętności przy użyciu istniejącej technologii opartej na plikach CCDT i chcą zmniejszyć koszty administracyjne i dystrybucyjne.
- Obecni klienci, którzy już stosują własną technologię przyzwoitości do dystrybucji definicji połączeń klientów.
- Nowi lub istniejący klienci, którzy obecnie nie korzystają z żadnego typu połączenia klienckiego i chcą korzystać z funkcji oferowanych przez IBM MQ.
- Nowi lub istniejący klienci, którzy chcą bezpośrednio używać lub dostroić swój model przesyłania komunikatów zgodnie z bieżącą architekturą biznesową LDAP.

ALW *Obsługa różnych środowisk*

Przed uruchomieniem przykładu Wyszukiwanie punktu końcowego połączenia sprawdź, czy używany jest obsługiwany system operacyjny i odpowiednie oprogramowanie.

Przykładowy program do wyszukiwania punktów końcowych połączenia IBM MQ wymaga następującego oprogramowania:

- IBM WebSphere MQ 7.0lub nowsza wersja
- Tivoli Directory Server V6.3 Client lub nowszy

Obsługiwane systemy operacyjne:

1. Windows (7 /8/2008/2012)
2. AIX
3. Linux
   - RHEL v4 i v5 w systemie System p
   - SUSE v9 i v10 na platformie System p
   - RHEL v4 i v5 x86-64 (wersja 32-bitowa i 64-bitowa)
   - SUSE v9 i v10 x86-64 32-i 64-bitowy

**Uwaga:** Przykład nie jest dostępny dla następujących platform:

- z/OS
- IBM i

ALW *Instalowanie i konfigurowanie*
Instalowanie i konfigurowanie modułu wyjścia i schematu punktu końcowego połączenia.

## Instalowanie modułu wyjścia

Podczas instalowania produktu IBM MQmoduł wyjścia jest instalowany w katalogu `tools/samples/c/preconnexit/bin`. W przypadku platform 32-bitowych przed użyciem modułu wyjścia należy go skopiować do katalogu `exit/`*`installation_name`*`/` . W przypadku platform 64-bitowych moduł wyjścia musi zostać skopiowany do katalogu exit64/*nazwa_instalacji*/, zanim będzie można go użyć.

## Instalowanie schematu punktu końcowego połączenia

Wyjście używa schematu punktu końcowego połączenia `ibm-amq.schema`. Przed użyciem wyjścia należy zaimportować plik schematu do dowolnego serwera LDAP. Po zaimportowaniu schematu należy dodać wartości atrybutów.

Poniżej przedstawiono przykład importowania schematu punktu końcowego połączenia. W przykładzie założono, że używany jest serwer IBM Tivoli Directory Server (ITDS).

- Upewnij się, że serwer IBM Tivoli Directory Server jest uruchomiony, a następnie skopiuj lub wyślij za pomocą protokołu FTP plik `ibm-amq.schema` na serwer ITDS.
- Na serwerze ITDS wprowadź następującą komendę, aby zainstalować schemat w składnicy ITDS, gdzie *ID LDAP* i *hasło LDAP* są główną nazwą wyróżniającą i hasłem dla serwera LDAP:

  `ldapadd -D "`*`LDAP ID`*`" -w "`*`LDAP password`*`" -f ibm-amq.schema`
- W oknie komend wprowadź następującą komendę lub użyj narzędzia innej firmy, aby przejrzeć schemat w celu weryfikacji:

```
ldapsearch objectclass=ibm-amqClientConnection
```

Więcej informacji na temat importowania pliku schematu zawiera dokumentacja serwera LDAP.

## Konfiguracja

Do pliku konfiguracyjnego klienta należy dodać nową sekcję o nazwie PreConnect , na przykład `mqclient.ini`. Sekcja PreConnect zawiera następujące słowa kluczowe:

**Moduł**
Nazwa modułu zawierającego kod wyjścia funkcji API. Jeśli to pole zawiera pełną ścieżkę do modułu, jest ono używane w takim stanie, w jakim się znajduje ("as is"). W przeciwnym razie przeszukiwany jest folder `exit` lub `exit64` w instalacji IBM MQ .

**Funkcja**
Nazwa funkcjonalnego punktu wejścia do biblioteki zawierającej kod wyjścia `LdapPreConnect` . Definicja funkcji jest zgodna z prototypem funkcji w przedsiębiorstwie.

**Ostrzeżenie:** Podczas określania rzeczywistego punktu wejścia należy usunąć znaki cudzysłowu w instrukcji funkcji.

**Dane**
Identyfikator URI repozytorium LDAP zawierającego definicje kanałów.

Poniższy fragment kodu jest przykładem zmian wymaganych w pliku `mqclient.ini` .

```
PreConnect:
Module=amqlcelp
Function="LdapPreconnectExit"
Data=ldap:dap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
Sequence=1
```

ALW *Przegląd wyjścia i schematu*
Składnia i parametry używane do nawiązania połączenia z menedżerem kolejek.

IBM MQ 9.3 definiuje następującą składnię dla punktu wejścia w module wyjścia.

```
void MQENTRY MQ_PRECONNECT_EXIT ( PMQNXP pExitParms
                                , PMQCHAR pQMgrName
                                , PPMQCNO ppConnectOpts
                                , PMQLONG pCompCode
                                , PMQLONG pReason)
```

Podczas wykonywania wywołania MQCONN/X klient IBM MQ C ładuje moduł wyjścia zawierający implementację składni funkcji. Następnie wywołuje funkcję wyjścia w celu pobrania definicji kanału. Pobrane definicje kanałów są następnie używane do nawiązania połączenia z menedżerem kolejek.

## Parametry

**Parametry pExit**
Typ: wejście/wyjście PMQNXP

Struktura parametru wyjścia PreConnection . Struktura jest przydzielana i obsługiwana przez program wywołujący wyjście.

```
struct tagMQNXP
{
  MQCHAR4    StrucId;             /* Structure identifier */
  MQLONG     Version;             /* Structure version number */
  MQLONG     ExitId;              /* Type of exit */
  MQLONG     ExitReason;          /* Reason for invoking exit */
  MQLONG     ExitResponse;        /* Response from exit */
  MQLONG     ExitResponse2;       /* Secondary response from exit */
  MQLONG     Feedback;            /* Feedback code (reserved) */
  MQLONG     ExitDataLength;      /* Exit data length */
  PMQCHAR    pExitDataPtr;        /* Exit data */
  MQPTR      pExitUserAreaPtr;    /* Exit user area */
  PMQCD *    ppMQCDArrayPtr;      /* Array of pointers to MQCDs */
  MQLONG     MQCDArrayCount;      /* Number of entries found */
  MQLONG     MaxMQCDVersion;      /* Maximum MQCD version */
};
```

**pQMgrNazwa**
Typ: PMQCHAR input/output

Nazwa menedżera kolejek. W przypadku danych wejściowych ten parametr jest łańcuchem filtru dostarczanym do wywołania funkcji API MQCONN za pośrednictwem parametru **QMgrName** . To pole może być puste, jawne lub zawierać znaki wieloznaczne. Pole jest zmieniane przez wyjście. Parametr ma wartość NULL , gdy wyjście jest wywoływane z opcją MQXR_TERM.

**Opcje ppConnect**

Typ: ppConnectOpcja wejścia/wyjścia

Opcje sterujące działaniem komendy MQCONNX. Jest to wskaźnik do struktury opcji połączenia MQCNO, która steruje działaniem wywołania API MQCONN. Parametr ma wartość NULL , gdy wyjście jest wywoływane z opcją MQXR_TERM. Klient MQI zawsze udostępnia na potrzeby wyjścia strukturę MQCNO, nawet jeśli nie została ona pierwotnie udostępniona przez aplikację. Jeśli aplikacja udostępnia strukturę MQCNO, klient tworzy duplikat, aby przekazać go do wyjścia, w którym został zmodyfikowany. Klient zachowuje prawo własności do obiektu MQCNO. Dysk MQCD, do którego odwołuje się obiekt MQCNO, ma pierwszeństwo przed każdą definicją połączenia udostępnioną przez tablicę. Klient używa struktury MQCNO do nawiązania połączenia z menedżerem kolejek, a pozostałe są ignorowane.

**Kod pComp**

Typ: wejście/wyjście PMQLONG

Kod zakończenia. Wskaźnik do obiektu MQLONG, który odbiera kod zakończenia wyjść. Musi to być jedna z następujących wartości:

- MQCC_OK -pomyślne zakończenie
- MQCC_WARNING -ostrzeżenie (częściowe zakończenie)
- MQCC_FAILED -Wywołanie nie powiodło się

**pReason**

Typ: wejście/wyjście PMQLONG

Przyczyna kwalifikowania kodu pComp. Wskaźnik do obiektu MQLONG, który odbiera kod przyczyny wyjścia. Jeśli kod zakończenia to MQCC_OK, jedyną poprawną wartością jest: MQRC_NONE -(0, x '000') Brak powodu do raportowania.

Jeśli kod zakończenia to MQCC_FAILED lub MQCC_WARNING, funkcja wyjścia może ustawić w polu kodu przyczyny dowolną poprawną wartość MQRC_ *.

ALW *Informacje o kontekście LDAP produktu MQ*

Wyjście używa następującej struktury danych dla informacji o kontekście.

**MQNLDAPCTX**

Struktura MQNLDAPCTX ma następujący prototyp C.

```
typedef struct tagMQNLDAPCTX MQNLDAPCTX;
typedef MQNLDAPCTX MQPOINTER PMQNLDAPCTX;

struct tagMQNLDAPCTX
{
  MQCHAR4      StrucId;          /* Structure identifier */
  MQLONG       Version;          /* Structure version number */
  LDAP *       objectDirectory   /* LDAP Instance */
  MQLONG       ldapVersion;      /* Which LDAP version to use? */
  MQLONG       port;             /* Port number for LDAP server*/
  MQLONG       sizeLimit;        /* Size limit */
  MQBOOL       ssl;              /* SSL enabled? */
  MQCHAR *     host;             /* Hostname of LDAP server */
  MQCHAR *     password;         /* Password of LDAP server */
  MQCHAR *     searchFilter;     /* LDAP search filter */
  MQCHAR *     baseDN;           /* Base Distinguished Name */
  MQCHAR *     charSet;          /* Character set */
};
```

Windows Linux AIX *Przykładowy kod do budowania wyjścia wyszukiwania punktu końcowego połączenia*

Przykładowych fragmentów kodu można używać do kompilowania kodu źródłowego w systemie AIX, Linuxlub Windows.

## Kompilowanie źródła

Można skompilować kod źródłowy z dowolnymi bibliotekami klienta LDAP, na przykład z bibliotekami klienta IBM Tivoli Directory Server V6.3 . W tej dokumentacji założono, że używane są biblioteki klienta Tivoli Directory Server V6.3 .

**Uwaga:** Biblioteka wyjścia przed połączeniem jest obsługiwana przez następujące serwery LDAP:

- IBM Tivoli Directory Server V6.3
- Novell eDirectory V8.2

Poniższe fragmenty kodu opisują sposób kompilowania wyjść:

Windows **Kompilowanie wyjścia na platformie Windows**

Do skompilowania źródła wyjścia można użyć następującego fragmentu kodu:

```
CC=cl.exe
LL=link.exe
CCARGS=/c /I. /DWIN32 /W3 /DNDEBUG /EHsc /D_CRT_SECURE_NO_DEPRECATE /Zl

# The libraries to include
LDLIBS=Ws2_32.lib Advapi32.lib libibmldapstatic.lib libibmldapdbgstatic.lib \
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib \
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib msvcrt.lib

OBJS=amqlcel0.obj

all: amqlcelp.dll

amqlcelp.dll: $(OBJS)
        $(LL) /OUT:amqlcelp.dll /INCREMENTAL /NOLOGO /DLL /SUBSYSTEM:WINDOWS /MACHINE: X86 \
              /DEF:amqlcelp.def $(OBJS) $(LDLIBS) /NODEFAULTLIB:msvcrt.lib

# The exit source
amqlcel0.obj: amqlcel0.c
$(CC) $(CCARGS) $*.c
```

**Uwaga:** Jeśli używane są biblioteki klienta IBM Tivoli Directory Server V6.3 , które są kompilowane z kompilatorem Microsoft Visual Studio 2003 , mogą wystąpić ostrzeżenia podczas kompilowania bibliotek klienta IBM Tivoli Directory Server V6.3 z kompilatorem Microsoft Visual Studio 2012lub nowszym.

Linux AIX **Kompilowanie wyjścia w systemie AIX, Linux**

Poniższy fragment kodu jest przeznaczony do kompilowania źródła wyjścia w systemie Linux. Niektóre opcje kompilatora mogą się różnić w systemie AIX.

```
#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
```

Produkt IBM Tivoli Directory Server jest dostarczany razem z bibliotekami dołączanymi statycznie i dynamicznie, ale można użyć tylko jednego typu biblioteki. W tym skrypcie założono, że używane są biblioteki statyczne.

```
#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 *Wywołanie modułu wyjścia PreConnect*

Moduł wyjścia PreConnect można wywołać z trzema różnymi kodami przyczyny: kodem przyczyny MQXR_INIT dla inicjowania i nawiązywania połączenia z serwerem LDAP, kodem przyczyny MQXR_PRECONNECT dla pobierania definicji kanału z serwera LDAP lub kodem przyczyny MQXR_TERM dla czyszczenia wyjścia.

**MQXR_INIT**

Wyjście jest wywoływane z kodem przyczyny MQXR_INIT w celu zainicjowania i nawiązania połączenia z serwerem LDAP.

Przed wywołaniem MQXR_INIT pole pExitDataPtr struktury MQNXP jest zapełniane atrybutem Data z sekcji PreConnect w pliku `mqclient.ini` (czyli LDAP).

URL LDAP składa się co najmniej z protokołu, nazwy hosta, numeru portu i podstawowej nazwy wyróżniającej dla wyszukiwania. Wyjście analizuje URL LDAP zawarty w polu pExitDataPtr , przydziela strukturę kontekstu wyszukiwania LDAP MQNLDAPCTX i odpowiednio ją zapełnia. Adres tej struktury jest przechowywany w polu pExitUserAreaPtr. Niepowodzenie poprawnej analizy adresu LDAP URL powoduje wystąpienie błędu MQCC_FAILED.

W tym momencie wyjście łączy się i łączy z serwerem LDAP przy użyciu parametrów **MQNLDAPCTX** . Wynikowe uchwyty interfejsu API LDAP są również przechowywane w tej strukturze.

**MQXR_PRECONNECT**

Moduł wyjścia jest wywoływany z kodem przyczyny MQXR_PRECONNECT w celu pobrania definicji kanałów z serwera LDAP.

Wyjście przeszukuje serwer LDAP pod kątem definicji kanałów zgodnych z podanym filtrem. Jeśli plik **QMgrNameparameter** zawiera konkretną nazwę menedżera kolejek, wyszukiwanie zwróci wszystkie definicje kanałów, dla których wartość atrybutu LDAP **ibm-amqQueueManagerName** jest zgodna z podaną nazwą menedżera kolejek.

Jeśli parametr **QMgrName** ma wartość '*' lub ' ' (puste), wówczas wyszukiwanie zwróci wszystkie definicje kanałów, dla których atrybut punktu końcowego **ibm-amqIsClientDefault Connection** ma wartość TRUE.

Po pomyślnym wyszukiwaniu wyjście przygotowuje jedną lub tablicę definicji MQCD i powraca do programu wywołującego.

**MQXR_TERM**

Wyjście jest wywoływane z tym kodem przyczyny, gdy ma zostać wyczyszczone. Podczas tego czyszczenia wyjście odłącza się od serwera LDAP i zwalnia całą pamięć przydzieloną i utrzymywaną przez wyjście, w tym strukturę MQNLDAPCTX, tablicę wskaźników i wszystkie przywoływane przez nie MQCD. Wszystkie inne pola są ustawione na wartości domyślne. Parametry wyjścia **pQMgrName** i **ppConnectOpts** nie są używane podczas wyjścia z kodem przyczyny MQXR_TERM i mogą mieć wartość NULL.

**Odsyłacze pokrewne**

Sekcja PreConnect w pliku konfiguracyjnym klienta

ALW *Schematy LDAP*

Dane połączenia klienta są przechowywane w repozytorium globalnym nazywanym katalogiem LDAP (Lightweight Directory Access Protocol). Klient IBM MQ używa katalogu LDAP do uzyskania definicji połączenia. Struktura definicji połączeń klienta IBM MQ w katalogu LDAP jest nazywana schematem LDAP. Schemat LDAP jest kolekcją definicji typów atrybutów, definicji klas obiektów i innych informacji używanych przez serwer do określania, czy asercja wartości filtru lub atrybutu jest zgodna z atrybutami pozycji oraz czy dozwolone są operacje dodawania i modyfikowania.

## Zapisywanie danych w katalogu LDAP

Definicje połączeń klienta znajdują się w określonej gałęzi w drzewie katalogów nazywanej punktem połączenia. Podobnie jak wszystkie inne węzły w katalogu LDAP, z punktem połączenia jest powiązana nazwa wyróżniająca (DN). Węzła tego można użyć jako punktu początkowego dla wszystkich zapytań wykonywanych w katalogu. Użyj filtrowania podczas wysyłania zapytań do katalogu LDAP w celu zwrócenia podzbioru definicji połączeń klienta. Dostęp do poddrzew można ograniczyć na podstawie uprawnień nadanych w innych częściach drzewa katalogów-na przykład do użytkowników, działów lub grup.

**Definiowanie własnych atrybutów i klas**

Zapisz definicję kanału klienta, modyfikując schemat LDAP. Wszystkie definicje danych LDAP wymagają obiektów i atrybutów. Obiekty i atrybuty są identyfikowane przez identyfikator obiektu (OID), który jednoznacznie identyfikuje obiekt lub atrybut. Wszystkie klasy w schemacie LDAP dziedziczą bezpośrednio lub pośrednio z obiektu najwyższego poziomu. Obiekt definicji kanału klienta zawiera atrybuty obiektu najwyższego poziomu. Wszystkie definicje danych LDAP wymagają obiektów i atrybutów:

- Definicje obiektów są kolekcjami atrybutów LDAP.
- Atrybuty są typami danych LDAP.

Opis każdego atrybutu i sposób jego odwzorowania na normalne właściwości IBM MQ opisano w sekcji Atrybuty LDAP.

ALW *LDAP - atrybuty*

Zdefiniowane atrybuty LDAP są specyficzne dla systemu IBM MQ i są odwzorowywane bezpośrednio na właściwości połączenia klienta.

**IBM MQ Łańcuch katalogu kanału klienta-atrybuty**

W poniższej tabeli przedstawiono atrybuty łańcucha znaków wraz z ich odwzorowaniem na właściwości IBM MQ . Atrybuty mogą przechowywać wartości directoryString (kodowanego w formacie UnicodeUTF-8 , czyli systemu ze zmiennym kodowaniem bajtowym, który zawiera IA5/ASCII jako podzbiór). Składnia jest określana przez numer identyfikacyjny obiektu (OID).

*Tabela 166. Łańcuch katalogu kanału klienta IBM MQ -atrybuty*

| Atrybut LDAP | Opis | IBM MQ Właściwość |
|---|---|---|
| CN | Nazwa zwykła składająca się z nazwy kanału i nazwy definiującego menedżera kolejek. | |
| ibm-amqChannelNazwa | Nazwa definicji kanału. | CHANNEL |
| ibm-amqConnectionNazwa | Identyfikator połączenia komunikacyjnego. | CONNAME |
| ibm-amqDescription | Opis kanału. | DESCR |
| ibm-amqLocalAdres | Lokalny adres komunikacyjny kanału. | LOCLADDR |
| ibm-amqModeNazwa | Nazwa trybu LU 6.2. | MODENAME |
| ibm-amqPassword | Hasło, które może być używane. | PASSWORD |

| *Tabela 166. Łańcuch katalogu kanału klienta IBM MQ -atrybuty (kontynuacja)* | | |
|---|---|---|
| **Atrybut LDAP** | **Opis** | **IBM MQ Właściwość** |
| ibm-amqQueueManagerName | Nazwa menedżera kolejek lub grupy menedżerów kolejek, z którymi aplikacja kliencka IBM MQ może żądać połączenia. | QMNAME |
| ibm-amqSecurityExitUserDane | Dane użytkownika przekazywane do wyjścia zabezpieczeń. | SCYDATA |
| ibm-amqSecurityExitName | Nazwa programu obsługi wyjścia, który ma być uruchomiony przez wyjście zabezpieczeń kanału. | SCYEXIT |
| ibm-amqSslCipherSpec | Pojedyncza CipherSpec dla połączenia TLS. | SSLCIPH |
| ibm-amqSslPeerName | Sprawdza nazwę wyróżniającą (DN) certyfikatu pochodzącego od menedżera kolejek węzła sieci lub klienta znajdującego się na drugim końcu kanału produktu IBM MQ . | SSLPEER |
| ibm-amqTransactionProgramName | Nazwa programu transakcyjnego. | TPNAME |
| ibm-amqUserID | Identyfikator użytkownika, który ma być używany przez agent MCA podczas próby zainicjowania bezpiecznej sesji SNA za pomocą zdalnego agenta MCA. | USERID |

**Atrybuty liczby całkowitej połączenia klienta IBM MQ**

Atrybuty z predefiniowanymi wartościami (na przykład typ wyliczeniowy) są przechowywane jako standardowe liczby całkowite. Wartości te są przechowywane w katalogu LDAP jako wartości całkowite, a nie przy użyciu powiązanej nazwy stałej.

| *Tabela 167. Atrybuty liczby całkowitej katalogu kanału klienta IBM MQ* | | |
|---|---|---|
| **Atrybut LDAP** | **Opis** | **IBM MQ Właściwość** |
| ibm-amqConnectionpowinowactwo | Określa, czy aplikacje klienckie, które łączą się wiele razy za pomocą tej samej nazwy menedżera kolejek, używają tego samego kanału klienta. | AFFINITY |
| ibm-amqClientChannelWeight | Waga wpływająca na to, która definicja kanału połączenia klienckiego jest używana. | CLNTWGHT |
| ibm-amqHeartBeatInterval | Przybliżony czas między przepływami pulsu przekazywanymi od wysyłającego agenta MCA, kiedy nie ma żadnych komunikatów w kolejce wysyłania. | HBINT |
| ibm-amqKeepAliveInterval | Wartość limitu czasu dla kanału. | KAINT |
| ibm-amqMaximumMessageLength | Maksymalna długość komunikatu, który może zostać przesłany w kanale. | MAXMSGL |
| ibm-amqSharingKonwersacje | Maksymalna liczba konwersacji, które współużytkują każdą instancję kanału TCP/IP. | SHARECNV |
| ibm-amqTransportTyp | Typ transportu, który ma być używany. | TRPTYPE |

**Atrybut boolowski kanału klienta IBM MQ**

Ten atrybut boolowski nie jest odwzorowany na żadną właściwość IBM MQ . Składnia tego atrybutu wskazuje wartość boolowską.

*Tabela 168. Atrybut boolowski kanału klienta IBM MQ*

| Atrybut LDAP | Opis |
|---|---|
| ibm-amqIsClientDefault | Ten atrybut boolowski jest zdefiniowany w celu rozwiązania problemu z wyszukiwaniem pozycji, których atrybut ibm-amqQueueManagerName nie został zdefiniowany. |

**Atrybuty listy kanałów klienta IBM MQ**

Właściwości IBM MQ są zapisywane w katalogu LDAP jako jednowartościowy, rozdzielany przecinkami atrybut listy. Atrybuty są definiowane w taki sam sposób, jak inne atrybuty łańcucha katalogu.
W poniższej tabeli opisano atrybuty listy wraz z ich odwzorowaniem na właściwości IBM MQ .

*Tabela 169. Atrybuty listy kanałów klienta IBM MQ*

| Atrybut LDAP | Opis | IBM MQ Właściwość |
|---|---|---|
| ibm-amqHeaderKompresja | Lista technik kompresji danych nagłówka obsługiwanych przez kanał. | COMPHDR |
| ibm-amqMessageKompresja | Lista technik kompresji danych komunikatu obsługiwanych przez kanał. | COMPMSG |
| ibm-amqSendExitUserDane | Dane użytkownika przekazywane do wyjścia wysyłania. | SENDDATA |
| ibm-amqSendExitUserNazwa | Nazwa programu obsługi wyjścia, który ma być uruchomiony przez program obsługi wyjścia wysyłania kanału. | SENDEXIT |
| ibm-amqReceiveExitUserDane | Dane użytkownika przekazywane do wyjścia odbierania. | RCVDATA |
| ibm-amqReceiveExitName | Nazwa programu obsługi wyjścia użytkownika, który ma być uruchomiony przez program obsługi wyjścia odbierania kanału. | RCVEXIT |

ALW *Nazwa zwykła*

Nazwa zwykła (CN) składa się z nazwy kanału i nazwy definiującego menedżera kolejek.

Jest to atrybut istniejący wcześniej.

Format CN jest następujący:

```
CN=CHANNEL_NAME(DEFINING_Q_MGR_NAME)
```

Na przykład:

```
CN=TC1(QM_T1)
```

Dla tego atrybutu można podać tylko jedną wartość.

Ten atrybut jest atrybutem łańcuchowym i w wartościach nie jest rozróżniana wielkość liter. Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcucha jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania przy użyciu podłańcucha (na przykład CN=jim *, gdzie CN jest atrybutem) i zawiera jeden lub więcej znaków wieloznacznych.

ALW *ibm-amqChannelNazwa*
Ten atrybut określa nazwę definicji kanału.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 20 znaków, w której nie jest rozróżniana wielkość liter. Nie jest to atrybut istniejący wcześniej.

Dopasowanie podłańcucha jest ignorowane. Dopasowanie podłańcucha jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania przy użyciu podłańcucha i zawiera co najmniej jeden znak wieloznaczny.

ALW *ibm-amqDescription*
Ten atrybut LDAP udostępnia opis kanału.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 64 bajtów, w której nie jest rozróżniana wielkość liter. Nie jest to atrybut istniejący wcześniej.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ALW *ibm-amqConnectionNazwa*
Ten atrybut LDAP jest identyfikatorem połączenia komunikacyjnego. Określa on konkretne łącza komunikacyjne, które mają być używane przez ten kanał.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 264 znaków, w której nie jest rozróżniana wielkość liter. Nie jest to atrybut istniejący wcześniej.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ALW *ibm-amqLocal*
Ten atrybut określa lokalny adres komunikacji dla kanału.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 48 znaków, w której nie jest rozróżniana wielkość liter. Nie jest to atrybut istniejący wcześniej.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ALW *ibm-amqModeNazwa*
Ten atrybut jest używany dla połączeń LU 6.2. Dodatkowa definicja parametrów sesji dla połączenia, gdy wykonywana jest alokacja sesji komunikacyjnej.

Ten atrybut ma pojedynczą wartość łańcuchową o długości dokładnie 8 znaków, w której nie jest rozróżniana wielkość liter. Nie jest to atrybut istniejący wcześniej.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ALW *ibm-amqPassword*
Ten atrybut LDAP określa hasło, które może być używane przez agent MCA podczas próby zainicjowania bezpiecznej sesji LU 6.2 za pomocą zdalnego agenta MCA.

Ten atrybut ma pojedynczą wartość całkowitą z maksymalnie 12 cyframi. Nie jest to atrybut istniejący wcześniej.

ALW *ibm-amqQueueManagerName*
Ten atrybut określa nazwę menedżera kolejek lub grupy menedżerów kolejek, z którymi aplikacja kliencka IBM MQ może zażądać połączenia.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 48 znaków, w której nie jest rozróżniana wielkość liter. Nie jest to atrybut istniejący wcześniej.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

**Odsyłacze pokrewne**

“ibm-amqIsClientDefault” na stronie 1191
Ten atrybut boolowski rozwiązuje problem z wyszukiwaniem pozycji, w których atrybut ibm-amqQueueManagerName nie został zdefiniowany.


*ibm-daneamqSecurityExitUser*
Ten atrybut LDAP określa dane użytkownika, które są przekazywane do wyjścia zabezpieczeń.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 999 znaków, w której nie jest rozróżniana wielkość liter. Nie jest to atrybut istniejący wcześniej.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.


*ibm-amqSecurityExitName*
Ten atrybut LDAP określa nazwę programu obsługi wyjścia, który ma być uruchamiany przez wyjście zabezpieczeń kanału.

Pozostaw to pole puste, jeśli nie jest aktywne żadne wyjście zabezpieczeń kanału.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 999 znaków, w której nie jest rozróżniana wielkość liter. Ten atrybut nie jest atrybutem poprzedzającym wyjście.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.


*ibm-amqSslCipherSpec*
Ten atrybut LDAP określa pojedynczą CipherSpec dla połączenia TLS.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 32 znaków, w której nie jest rozróżniana wielkość liter. Nie jest to atrybut istniejący wcześniej.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.


*ibm-amqSslPeerName*
Ten atrybut LDAP jest używany do sprawdzania nazwy wyróżniającej certyfikatu pochodzącego od menedżera kolejek węzła sieci lub klienta na drugim końcu kanału produktu IBM MQ .

Ten atrybut LDAP ma pojedynczą wartość łańcuchową o maksymalnej wielkości 1024 bajtów, w której nie jest rozróżniana wielkość liter. Nie jest to już istniejące.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.


*ibm-amqTransactionProgramName*
Ten atrybut LDAP określa nazwę programu transakcyjnego. Jest on używany z połączeniami jednostki logicznej 6.2 .

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 64 znaków, w której nie jest rozróżniana wielkość liter. Nie jest to już istniejące.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.


*ibm-amqUserID*
Ten atrybut LDAP określa identyfikator użytkownika, który ma być używany przez agent MCA podczas próby zainicjowania bezpiecznej sesji SNA za pomocą zdalnego agenta MCA.

Ten atrybut ma pojedynczą wartość łańcuchową o długości dokładnie 12 znaków, w której nie jest rozróżniana wielkość liter. Nie jest to atrybut istniejący wcześniej.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ALW *ibm-amqConnectionpowinowactwo*
Ten atrybut LDAP określa, czy aplikacje klienckie, które łączą się wielokrotnie przy użyciu tej samej nazwy menedżera kolejek, używają tego samego kanału klienta.

Ten atrybut ma pojedynczą wartość całkowitą. Nie jest to atrybut istniejący wcześniej.

ALW *ibm-amqClientChannelWeight*
Ten atrybut LDAP określa wagę, która wpływa na używaną definicję kanału połączenia klienckiego.

Atrybut ważenia kanału klienta jest używany do określania wyboru definicji kanału klienta, gdy dostępna jest więcej niż jedna odpowiednia definicja.

Ten atrybut ma pojedynczą wartość całkowitą. Nie jest to atrybut istniejący wcześniej.

ALW *ibm-amqHeartBeatInterval*
Ten atrybut LDAP określa przybliżony czas między przepływami pulsu przekazywanymi od wysyłającego agenta MCA, gdy w kolejce transmisji nie ma żadnych komunikatów.

Ten atrybut ma pojedynczą wartość całkowitą. Nie jest to atrybut istniejący wcześniej. Wartością domyślną jest 1. Wartość domyślna jest ustawiana w bieżącej operacji zmiennej środowiskowej MQSERVER.

ALW *ibm-amqKeepAliveInterval*
Ten atrybut LDAP jest używany do określenia wartości limitu czasu dla kanału.

Wartość tego atrybutu jest przekazywana do stosu komunikacji, określając czas sprawdzania połączenia dla kanału. Za pomocą tej opcji można określić inną wartość keepalive dla każdego kanału.

Ten atrybut ma pojedynczą wartość całkowitą. Nie jest to atrybut istniejący wcześniej.

ALW *ibm-amqMaximumMessageLength*
Ten atrybut LDAP określa maksymalną długość komunikatu, który może być przesyłany przez kanał.

Wartością domyślną tego atrybutu jest 104857600 zgodnie z bieżącą operacją zmiennej środowiskowej MQSERVER. Ten atrybut ma pojedynczą wartość całkowitą i nie jest wcześniej istniejącym atrybutem.

ALW *ibm-amqSharingKonwersacje*
Ten atrybut LDAP określa maksymalną liczbę konwersacji, które współużytkują każdą instancję kanału TCP/IP.

Ten atrybut ma pojedynczą wartość całkowitą. Ten atrybut nie jest wcześniej istniejącym atrybutem.

ALW *ibm-amqTransport*
Ten atrybut LDAP określa typ transportu, który ma być używany.

Ten atrybut ma pojedynczą wartość całkowitą. Nie jest to atrybut istniejący wcześniej.

ALW *ibm-amqIsClientDefault*
Ten atrybut boolowski rozwiązuje problem z wyszukiwaniem pozycji, w których atrybut ibm-amqQueueManagerName nie został zdefiniowany.

Moduły wyjścia połączenia wstępnego generalnie przeszukują serwery LDAP z wartością atrybutu ibm-amqQueueManagerName jako kryterium wyszukiwania. Takie zapytanie zwróci wszystkie pozycje, w których wartość atrybutu ibm-amqQueueManagerName jest zgodna z nazwą menedżera kolejek określoną w wywołaniu MQCONN/X. Jeśli jednak używane są tabele definicji kanału klienta (CCDT),

można ustawić nazwę menedżera kolejek w wywołaniu MQCONN/X jako pustą lub poprzedzić nazwę gwiazdką (*). Jeśli nazwa menedżera kolejek jest pusta, klient nawiązuje połączenie z domyślnym menedżerem kolejek. Jeśli nazwa jest poprzedzona gwiazdką (*) w menedżerze kolejek, klient łączy się z dowolnym menedżerem kolejek.

Podobnie atrybut ibm-amqQueueManagerName w pozycji może pozostać niezdefiniowany. W takim przypadku oczekuje się, że klient korzystający z tych informacji o punkcie końcowym może nawiązać połączenie z dowolnym menedżerem kolejek. Na przykład wpis zawiera następujące wiersze:

```
ibm-amqChannelName = "CHANNEL1"
ibm-amqConnectionName = myhost(1414)
```

W tym przykładzie klient próbuje nawiązać połączenie z określonym menedżerem kolejek działającym w systemie `myhost`.

Jednak w przypadku serwerów LDAP wyszukiwanie nie jest wykonywane dla wartości atrybutu, która nie została zdefiniowana. Na przykład, jeśli pozycja zawiera informacje o połączeniu z wyjątkiem ibm-amqQueueManagerName, wówczas wyniki wyszukiwania nie będą zawierać tej pozycji. Aby rozwiązać ten problem, można ustawić wartość ibm-amqIsClientDefault. Jest to atrybut boolowski i przyjmuje się, że ma on wartość FALSE, jeśli nie jest zdefiniowany.

W przypadku pozycji, w których nie zdefiniowano ibm-amqQueueManagerName i które powinny być częścią wyszukiwania, należy ustawić wartość ibm-amqIsClientDefault na TRUE. Jeśli jako nazwa menedżera kolejek w wywołaniu komendy MQCONN/X zostanie podana wartość pusta lub gwiazdka (*), wyjście połączenia wstępnego przeszukuje serwer LDAP w poszukiwaniu wszystkich pozycji, dla których atrybut ibm-amqIsClientDefault ma wartość TRUE.

**Uwaga:** Nie należy ustawiać ani definiować atrybutu ibm-amqQueueManagerName , jeśli atrybut ibm-amqIsClientDefault ma wartość TRUE.

**Odsyłacze pokrewne**

“ibm-amqQueueManagerName” na stronie 1189
Ten atrybut określa nazwę menedżera kolejek lub grupy menedżerów kolejek, z którymi aplikacja kliencka IBM MQ może zażądać połączenia.

ALW *ibm-kompresjaamqHeader*
Ten atrybut LDAP jest listą technik kompresji danych nagłówka obsługiwanych przez kanał.

Maksymalna wielkość tego atrybutu wynosi 48 znaków. Nie jest to atrybut istniejący wcześniej.

Dla tego atrybutu można podać tylko jedną wartość.

Ten atrybut listy jest określany jako łańcuchy katalogów w formacie rozdzielanym przecinkami. Na przykład wartością określoną dla parametru **ibm-amqHeaderCompression** jest `0` , która jest odwzorowywana na wartość `NONE`. Wszystkie wartości przekraczające maksymalny dozwolony limit są ignorowane przez klienta. Na przykład kompresja ibm-amqHeaderzawiera na liście maksymalnie dwie liczby całkowite.

ALW *ibm-kompresjaamqMessage*
Ten atrybut LDAP jest listą technik kompresji danych komunikatu obsługiwanych przez kanał.

Maksymalna wielkość tego atrybutu wynosi 48 znaków. Nie jest to atrybut istniejący wcześniej.

Ten atrybut nie obsługuje wielu wartości.

Ten atrybut listy jest określany jako łańcuchy katalogów w formacie rozdzielanym przecinkami. Na przykład wartość określona dla tego atrybutu to 1,2,4, która jest odwzorowywana na bazowe sekwencje kompresji RLE, ZLIBFAST i ZLIBHIGH.

Wszystkie wartości przekraczające maksymalny dozwolony limit są ignorowane przez klienta. Na przykład kompresja ibm-amqMessagezawiera maksymalnie 16 liczb całkowitych na liście.

ALW *ibm-daneamqSendExitUser*
Ten atrybut LDAP określa dane użytkownika, które są przekazywane do wyjścia wysyłania.

Ten atrybut LDAP ma pojedynczą wartość łańcuchową o maksymalnej długości 999 znaków, w której nie jest rozróżniana wielkość liter. Nie jest to atrybut istniejący wcześniej.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

**Uwaga:** Systemy **ibm-amqSendExitName** i **ibm-amqSendExitUserData** muszą być synchronizowane parami. Dane użytkownika powinny być zsynchronizowane z nazwą wyjścia. Zatem jeśli jeden z nich jest określony, drugi również musi być określony symetrycznie, nawet jeśli nie zawiera żadnych danych.

ALW *ibm-amqSendExitName*
Ten atrybut LDAP określa nazwę programu obsługi wyjścia, który ma być uruchamiany przez program obsługi wyjścia wysyłania kanału.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 999 znaków, w której nie jest rozróżniana wielkość liter. Nie jest to atrybut istniejący wcześniej.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

**Uwaga:** Systemy **ibm-amqSendExitName** i **ibm-amqSendExitUserData** muszą być synchronizowane parami. Dane użytkownika muszą być zsynchronizowane z nazwą wyjścia. Tak więc, jeśli jeden z nich jest określony, drugi również musi być określony symetrycznie, nawet jeśli nie zawiera żadnych danych.

ALW *ibm-amqReceiveExitUserDane*
Ten atrybut LDAP określa dane użytkownika przekazywane do wyjścia odbierania.

Można uruchomić sekwencję wyjść odbierania. Łańcuch danych użytkownika dla serii wyjść jest oddzielony przecinkiem, spacjami lub obydwoma tymi znakami.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 999 znaków, w której nie jest rozróżniana wielkość liter. Nie jest to atrybut istniejący wcześniej.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

**Uwaga:** Systemy **ibm-amqReceiveExitName** i **ibm-amqReceiveExitUserData** muszą być synchronizowane parami. Dane użytkownika muszą być zsynchronizowane z nazwą wyjścia. Tak więc, jeśli jeden z nich jest określony, drugi również musi być określony symetrycznie, nawet jeśli nie zawiera żadnych danych.

ALW *ibm-amqReceiveExitName*
Ten atrybut LDAP określa nazwę programu użytkownika obsługi wyjścia, który ma być uruchomiony przez program obsługi wyjścia odbierania kanału.

Ten atrybut jest listą nazw programów, które mają być uruchamiane po sobie. Pozostaw to pole puste, jeśli nie jest aktywna żadna procedura zewnętrzna odbierania kanału.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 999 znaków, w której nie jest rozróżniana wielkość liter. Nie jest to atrybut istniejący wcześniej.

Dopasowanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

**Uwaga:** Systemy **ibm-amqReceiveExitName** i **ibm-amqReceiveExitUserData** muszą być synchronizowane parami. Dane użytkownika muszą być zsynchronizowane z nazwą wyjścia. Zatem jeśli jeden z nich jest określony, drugi musi być również określony symetrycznie, nawet jeśli nie zawiera żadnych danych.

# z/OS Korzystanie z przykładowych programów w systemie z/OS

Przykładowe aplikacje proceduralne dostarczane z produktem IBM MQ for z/OS demonstrują typowe zastosowania interfejsu MQI (Message Queue Interface).

## O tym zadaniu

IBM MQ for z/OS udostępnia również przykładowe wyjścia konwersji danych opisane w sekcji “Zapisywanie wyjść konwersji danych” na stronie 1013.

Wszystkie aplikacje przykładowe są dostarczane w postaci kodu źródłowego; kilka z nich jest również dostarczanych w postaci kodu wykonywalnego. Moduły źródłowe zawierają pseudokod opisujący logikę programu.

**Uwaga:** Chociaż niektóre przykładowe aplikacje mają podstawowe interfejsy sterowane panelem, nie mają one na celu zademonstrowania sposobu projektowania wyglądu i zachowania aplikacji. Więcej informacji na temat projektowania interfejsów sterowanych panelem dla terminali nieprogramowalnych zawiera publikacja *SAA Common User Access: Basic Interface Design Guide* (SC26-4583) i jej dodatek (GG22-9508). Zawierają one wytyczne ułatwiające projektowanie aplikacji, które są spójne zarówno w aplikacji, jak i w innych aplikacjach.

## Procedura

- Aby uzyskać więcej informacji na temat programów przykładowych, skorzystaj z następujących odsyłaczy:
  - “Funkcje przedstawione w przykładowych aplikacjach dla produktu z/OS” na stronie 1195
  - “Przygotowywanie i uruchamianie przykładowych aplikacji dla środowiska wsadowego w systemie z/OS” na stronie 1201
  - “Przygotowywanie przykładowych aplikacji dla środowiska TSO w systemie z/OS” na stronie 1204
  - “Przygotowywanie przykładowych aplikacji dla środowiska CICS w systemie z/OS” na stronie 1206
  - “Przygotowywanie przykładowej aplikacji dla środowiska IMS w systemie z/OS” na stronie 1210
  - “Przykłady Put w systemie z/OS” na stronie 1211
  - “Przykłady Get w serwisie z/OS” na stronie 1213
  - “Przykład przeglądania w systemie z/OS” na stronie 1216
  - “Przykład Print Message w systemie z/OS” na stronie 1218
  - “Przykład atrybutów kolejki w systemie z/OS” na stronie 1222
  - “Przykład menedżera poczty w systemie z/OS” na stronie 1223
  - “Przykład sprawdzania zdolności kredytowej w systemie z/OS” na stronie 1230
  - “Przykład procedury obsługi komunikatów w systemie z/OS” na stronie 1242
  - “Przykład asynchronicznego umieszczania w systemie z/OS” na stronie 1246
  - “Przykład wykorzystania asynchronicznego zadania wsadowego w systemie z/OS” na stronie 1247
  - “Przykład CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS” na stronie 1248
  - “Przykład publikowania/subskrypcji w serwisie z/OS” na stronie 1251
  - “Przykład właściwości Set and Inquire message property w systemie z/OS” na stronie 1253

**Zadania pokrewne**

“Korzystanie z przykładowych programów w systemie Multiplatforms” na stronie 1090
Te przykładowe programy proceduralne są dostarczane z produktem. Przykłady zostały napisane w języku C i COBOL i przedstawiają typowe zastosowania interfejsu kolejek komunikatów (Message Queue Interface-MQI).

### z/OS *Funkcje przedstawione w przykładowych aplikacjach dla produktu z/OS*

Ta sekcja zawiera podsumowanie funkcji interfejsu MQI demonstrowane w każdej z przykładowych aplikacji, przedstawia języki programowania, w których każda próbka jest napisana, oraz środowisko, w którym jest uruchamiana każda próbka.

#### z/OS *Umieść próbki w serwisie z/OS*

Przykłady umieszczania demonstrują sposób umieszczania komunikatów w kolejce przy użyciu wywołania MQPUT.

Aplikacja używa następujących wywołań MQI:

- ZMQCONN
- MQOPEN
- MQPUT
- MQCLOSE
- MQDISC

Program jest dostarczany w językach COBOL i C i działa w środowisku wsadowym i w środowisku CICS . Patrz Tabela 172 na stronie 1202 dla aplikacji wsadowej i Tabela 179 na stronie 1207 dla aplikacji CICS .

#### z/OS *Pobierz przykłady w serwisie z/OS*

Przykłady pobierania demonstrują sposób pobierania komunikatów z kolejki przy użyciu wywołania MQGET.

Aplikacja używa następujących wywołań MQI:

- ZMQCONN
- MQOPEN
- MQGET
- MQCLOSE
- MQDISC

Program jest dostarczany w językach COBOL i C i działa w środowisku wsadowym i w środowisku CICS . Patrz Tabela 172 na stronie 1202 dla aplikacji wsadowej i Tabela 179 na stronie 1207 dla aplikacji CICS .

#### z/OS *Przeglądanie przykładu w serwisie z/OS*

Przykład przeglądania demonstruje sposób użycia opcji Przeglądaj w celu znalezienia komunikatu, wydrukowania go, a następnie przejścia przez komunikaty w kolejce.

Aplikacja używa następujących wywołań MQI:

- ZMQCONN
- MQOPEN
- MQGET dla przeglądania komunikatów
- MQCLOSE
- MQDISC

Program jest dostarczany w językach COBOL, assembler, PL/I i C. Aplikacja jest uruchamiana w środowisku wsadowym. Informacje na temat aplikacji wsadowej zawiera sekcja Tabela 173 na stronie 1203 .

#### z/OS *Przykład komunikatu w systemie z/OS*

Przykład Drukowanie komunikatu demonstruje sposób usuwania komunikatu z kolejki i drukowania danych w komunikacie wraz ze wszystkimi polami jego deskryptora komunikatu. Opcjonalnie może on wyświetlić wszystkie właściwości komunikatu powiązane z każdym komunikatem.

Usuwając znaki komentarza z dwóch wierszy w module źródłowym, można zmienić program tak, aby przeglądał komunikaty w kolejce, a nie je usuwał. Ten program może być używany do diagnozowania problemów z aplikacją umieszczającą komunikaty w kolejce.

Aplikacja używa następujących wywołań MQI:

- ZMQCONN
- MQOPEN
- MQGET dla usuwania komunikatów z kolejki (z opcją przeglądania)
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

Program jest dostarczany w języku C. Aplikacja jest uruchamiana w środowisku wsadowym. Informacje na temat aplikacji wsadowej zawiera sekcja Tabela 174 na stronie 1203 .

z/OS *Przykład atrybutów kolejki w systemie z/OS*

Przykład atrybutów kolejki przedstawia sposób uzyskiwania informacji o atrybutach obiektu IBM MQ for z/OS i ustawiania ich wartości.

Aplikacja używa następujących wywołań MQI:

- MQOPEN
- MQINQ
- MQSET
- MQCLOSE

Program jest dostarczany w językach COBOL, assembler i C. Aplikacja działa w środowisku CICS . Informacje na temat aplikacji CICS zawiera sekcja Tabela 180 na stronie 1208 .

z/OS *Przykład programu Mail Manager w systemie z/OS*

Uwagi dotyczące używania przykładu programu Mail Manager.

Przykładowa aplikacja Mail Manager demonstruje następujące techniki:

- Korzystanie z kolejek aliasowych
- Używanie kolejki modelowej do tworzenia tymczasowej kolejki dynamicznej
- Korzystanie z kolejek odpowiedzi
- Korzystanie z punktów synchronizacji w środowisku CICS i środowisku wsadowym
- Wysyłanie komend do kolejki wejściowej komend systemowych
- Testowanie kodów powrotu
- Wysyłanie komunikatów do zdalnych menedżerów kolejek przy użyciu lokalnej definicji kolejki zdalnej oraz umieszczanie komunikatów bezpośrednio w nazwanej kolejce w zdalnym menedżerze kolejek

Aplikacja używa następujących wywołań MQI:

- ZMQCONN
- MQOPEN
- MQPUT1
- MQGET
- MQINQ
- MQCMIT

- MQCLOSE
- MQDISC

Dostępne są trzy wersje aplikacji:

- Aplikacja CICS napisana w języku COBOL
- Aplikacja TSO napisana w języku COBOL
- Aplikacja TSO napisana w języku C

Aplikacje TSO używają adaptera wsadowego IBM MQ for z/OS i zawierają niektóre panele ISPF .

Patrz Tabela 177 na stronie 1205 dla aplikacji TSO i Tabela 181 na stronie 1208 dla aplikacji CICS .

z/OS *Przykład sprawdzania zdolności kredytowej w systemie z/OS*

Te informacje zawierają punkty, które należy wziąć pod uwagę podczas korzystania z przykładu sprawdzania zdolności kredytowej.

Przykład Credit Check jest pakietem programów, które demonstrują następujące techniki:

- Tworzenie aplikacji działającej w więcej niż jednym środowisku
- Używanie kolejki modelowej do tworzenia tymczasowej kolejki dynamicznej
- Korzystanie z identyfikatora korelacji
- Ustawianie i przekazywanie informacji o kontekście
- Używanie priorytetu i trwałości komunikatu
- Uruchamianie programów za pomocą wyzwalania
- Korzystanie z kolejek odpowiedzi
- Korzystanie z kolejek aliasowych
- Korzystanie z kolejki niedostarczonych komunikatów
- Korzystanie z listy nazw
- Testowanie kodów powrotu

Aplikacja używa następujących wywołań MQI:

- MQOPEN
- MQPUT
- MQPUT1
- MQGET do przeglądania i pobierania komunikatów, korzystania z opcji oczekiwania i sygnału oraz do pobierania konkretnego komunikatu
- MQINQ
- MQSET
- MQCLOSE

Przykład można uruchomić jako autonomiczną aplikację CICS . Jednak aby zademonstrować sposób projektowania aplikacji kolejkowania komunikatów korzystającej z narzędzi udostępnianych przez środowiska CICS i IMS , jeden moduł jest również dostarczany jako program przetwarzania komunikatów wsadowych systemu IMS .

Programy CICS są dostarczane w języku C i COBOL. Pojedynczy program IMS jest dostarczany w języku C.

Patrz Tabela 182 na stronie 1209 dla aplikacji CICS i Tabela 184 na stronie 1211 dla aplikacji IMS .

z/OS *Przykład procedury obsługi komunikatów w systemie z/OS*

Przykład procedury obsługi komunikatów umożliwia przeglądanie, przekazywanie i usuwanie komunikatów w kolejce.

Aplikacja używa następujących wywołań MQI:

- ZMQCONN
- MQOPEN
- MQINQ
- MQPUT1
- MQCMIT
- MQBACK
- MQGET
- MQCLOSE
- MQDISC

Program jest dostarczany w językach programowania C i COBOL. Aplikacja działa w środowisku TSO. Informacje na temat aplikacji TSO zawiera sekcjaTabela 178 na stronie 1206.

z/OS *Przykłady rozproszonego wyjścia kolejkowania w systemie z/OS*

Tabela programów źródłowych przykładowych programów obsługi wyjścia kolejkowania rozproszonego.

Nazwy programów źródłowych przykładowych programów obsługi wyjścia rozproszonego kolejkowania są wymienione w poniższej tabeli:

*Tabela 170. Źródło dla przykładów rozproszonego wyjścia kolejkowania*

| **Nazwa elementu klastra** | **Dla języka** | **Opis** | **Dostarczane w bibliotece** |
|---|---|---|---|
| CSQ4BAX0 | Assembler | Program źródłowy | SCSQASMS, |
| CSQ4BCX1 | C | Program źródłowy | SCSQC37S |
| CSQ4BCX2 | C | Program źródłowy | SCSQC37S |
| CSQ4BCX4 | C | Program źródłowy | SCSQC37S |

**Uwaga:** Programy źródłowe są edytowane za pomocą odsyłacza CSQXSTUB.

z/OS *Przykłady wyjścia konwersji danych w systemie z/OS*

Dla procedury zewnętrznej konwersji danych udostępniono szkielet, a przykład jest dostarczany z produktem IBM MQ ilustrującym wywołanie MQXCNVC.

Nazwy programów źródłowych przykładowych programów obsługi wyjścia konwersji danych są wymienione w poniższej tabeli:

*Tabela 171. Źródło przykładów wyjścia konwersji danych (tylko w języku asemblera)*

| **Nazwa elementu klastra** | **Opis** | **Dostarczane w bibliotece** |
|---|---|---|
| CSQ4BAX8 | Program źródłowy | SCSQASMS, |
| CSQ4BAX9 | Program źródłowy | SCSQASMS, |
| CSQ4CAX9 | Program źródłowy | SCSQASMS, |

**Uwaga:** Programy źródłowe są połączone z CSQASTUB.

Więcej informacji zawiera sekcja “Zapisywanie wyjść konwersji danych” na stronie 1013.

z/OS *Przykłady publikowania/subskrypcji w systemie z/OS*

Przykładowe programy publikowania/subskrypcji demonstrują użycie funkcji publikowania i subskrypcji w produkcie IBM MQ.

Istnieją cztery przykładowe programy w języku programowania C i dwa w języku COBOL demonstrujące sposób programowania w interfejsie publikowania/subskrypcji produktu IBM MQ .

Aplikacje używają następujących wywołań MQI:

- ZMQCONN
- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

Programy przykładowe typu Public/Subscribe są dostarczane w językach programowania C i COBOL. Przykładowe aplikacje działają w środowisku wsadowym. Informacje na temat aplikacji wsadowych można znaleźć w sekcji Przykłady publikowania/subskrypcji .

## z/OS *Konfigurowanie menedżera kolejek w celu akceptowania połączeń klientów w systemie z/OS*

Przed uruchomieniem przykładowych aplikacji należy najpierw utworzyć menedżer kolejek. Następnie można skonfigurować menedżer kolejek w taki sposób, aby bezpiecznie akceptował przychodzące żądania połączeń z aplikacji działających w trybie klienta.

### Zanim rozpoczniesz

Upewnij się, że menedżer kolejek już istnieje i został uruchomiony. Określ, czy rekordy uwierzytelniania kanału są już włączone, wprowadzając komendę MQSC:

```
DISPLAY QMGR CHLAUTH
```

**Ważne:** To zadanie oczekuje, że rekordy uwierzytelniania kanału są włączone. Jeśli jest to menedżer kolejek używany przez innych użytkowników i aplikacje, zmiana tego ustawienia będzie miała wpływ na wszystkich innych użytkowników i aplikacje. Jeśli menedżer kolejek nie korzysta z rekordów uwierzytelniania kanału, krok 4 można zastąpić alternatywną metodą uwierzytelniania (na przykład wyjściem zabezpieczeń), która ustawia parametr MCAUSER na *identyfikator nieuprzywilejowanego użytkownika* , który zostanie uzyskany w kroku “1” na stronie 1199.

Aby aplikacja mogła korzystać z kanału, należy znać nazwę kanału, który ma być używany przez aplikację. Należy również wiedzieć, które obiekty, na przykład kolejki lub tematy, mają być używane przez aplikację, aby można było z nich korzystać.

### O tym zadaniu

To zadanie tworzy identyfikator użytkownika nieuprzywilejowanego, który ma być używany dla aplikacji klienckiej łączącej się z menedżerem kolejek. Dostęp jest przyznawany aplikacji klienckiej tylko po to, aby mogła używać kanału, którego potrzebuje, i kolejki, której potrzebuje, przy użyciu tego identyfikatora użytkownika.

### Procedura

1. Uzyskaj identyfikator użytkownika w systemie, w którym działa menedżer kolejek.

W przypadku tego zadania ten identyfikator użytkownika nie może być uprzywilejowanym użytkownikiem administracyjnym. Ten ID użytkownika jest uprawnieniem, z którego będzie uruchamiane połączenie klienta w menedżerze kolejek.

2. Uruchom program nasłuchujący.
   a) Upewnij się, że inicjator kanału jest uruchomiony. Jeśli nie, uruchom go za pomocą komendy **START CHINIT** .
   b) Uruchom program nasłuchujący, wydając następującą komendę:

```
START LISTENER TRPTYPE(TCP) PORT(nnnn)
```

   gdzie *nnnn* jest wybranym numerem portu.
3. Jeśli aplikacja używa systemu SYSTEM.DEF.SVRCONN oznacza, że ten kanał jest już zdefiniowany. Jeśli aplikacja używa innego kanału, utwórz go, wprowadzając komendę MQSC:

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

   *nazwa-kanału* jest nazwą kanału.
4. Utwórz regułę uwierzytelniania kanału umożliwiającą użycie kanału tylko adresowi IP systemu klienta, wydając komendę MQSC:

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

   where

   *nazwa-kanału* jest nazwą kanału.
   *client-machine-IP-address* to adres IP systemu klienta. Jeśli przykładowa aplikacja kliencka działa na tym samym komputerze, co menedżer kolejek, należy użyć adresu IP127.0.0.1, jeśli aplikacja ma nawiązywać połączenie przy użyciu hosta lokalnego. Jeśli kilka różnych komputerów klienckich ma się łączyć, można użyć wzorca lub zakresu zamiast pojedynczego adresu IP. Szczegółowe informacje na ten temat zawiera sekcja Ogólne adresy IP .
   *identyfikator nieuprzywilejowanego użytkownika* to identyfikator użytkownika uzyskany w kroku “1” na stronie 1199
5. Jeśli aplikacja używa systemu SYSTEM.DEFAULT.LOCAL.QUEUE-ta kolejka jest już zdefiniowana. Jeśli aplikacja używa innej kolejki, utwórz ją za pomocą komendy MQSC:

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

   gdzie *nazwa_kolejki* jest nazwą kolejki.
6. Przyznaj dostęp do połączenia z menedżerem kolejek i utwórz do niego zapytanie:
   a) Upewnij się, że inicjator kanału jest uruchomiony. Jeśli nie, uruchom inicjatora kanału, wydając komendę START CHINIT.
   b) Uruchom program nasłuchujący TCP, na przykład wydaj następującą komendę:

```
START LISTENER TRPTYPE(TCP) PORT(nnnn)
```

   gdzie *nnnn* jest wybranym numerem portu.
7. Jeśli aplikacja jest aplikacją typu punkt z punktem, to znaczy korzysta z kolejek, należy przyznać dostęp umożliwiający wysyłanie zapytań oraz umieszczanie i pobieranie komunikatów przy użyciu kolejki przy użyciu identyfikatora użytkownika, który ma być używany, za pomocą komend MQSC:
   Wykonaj komendy RACF :

```
RDEFINE MQQUEUE qmgr-name.QUEUE. queue-name UACC(NONE)
```

```
PERMIT qmgr-name.QUEUE. queue-name CLASS(MQQUEUE) ID(non-privileged-user-id) ACCESS(UPDATE)
```

where

*nazwa_menedżera_kolejek* to nazwa menedżera kolejek.
*nazwa-kolejki* jest nazwą kolejki.
*identyfikator nieuprzywilejowanego użytkownika* to identyfikator użytkownika uzyskany w kroku “1” na stronie 1199

8. Jeśli aplikacja jest aplikacją publikującą/subskrybującą, czyli korzysta z tematów, należy przyznać dostęp umożliwiający publikowanie i subskrybowanie przy użyciu tematu za pomocą identyfikatora użytkownika, który ma być używany, za pomocą następujących komend RACF :

```
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)
```

where

*nazwa_menedżera_kolejek* to nazwa menedżera kolejek.
*identyfikator nieuprzywilejowanego użytkownika* to identyfikator użytkownika uzyskany w kroku “1” na stronie 1199
Spowoduje to nadanie *nieuprawnionemu ID użytkownika* dostępu do dowolnego tematu w drzewie tematów. Można również zdefiniować obiekt tematu za pomocą komendy **DEFINE TOPIC** i nadać dostęp tylko do części drzewa tematów, do której odwołuje się ten obiekt tematu. Więcej informacji na ten temat zawiera sekcja Kontrolowanie dostępu użytkowników do tematów.

## Co dalej

Aplikacja kliencka może teraz nawiązać połączenie z menedżerem kolejek i umieszczać komunikaty przy użyciu kolejki.

**Pojęcia pokrewne**

z/OS Uprawnienia do pracy z obiektami IBM MQ w systemie z/OS

**Odsyłacze pokrewne**

USTAW CHLAURA
Zdefiniowanie kanału
ZDEFINIUJ QLOCAL
SET AUTHREC

## z/OS *Przygotowywanie i uruchamianie przykładowych aplikacji dla środowiska wsadowego w systemie z/OS*

Aby przygotować przykładową aplikację, która działa w środowisku wsadowym, wykonaj te same kroki, co podczas budowania dowolnej aplikacji wsadowej IBM MQ for z/OS .

Te kroki są wymienione w sekcji “Budowanie aplikacji wsadowych z/OS” na stronie 1054.

Alternatywnie, jeśli dostarczamy wykonywalną formę przykładu, można ją uruchomić z biblioteki dynamicznej thlqual.SCSQLOAD .

**Uwaga:** Wersja w języku asemblera przykładu Browse używa bloków sterujących danych (DCBs), dlatego należy ją dowiązać i edytować za pomocą RMODE(24).

Lista elementów biblioteki, które mają być używane, znajduje się w sekcji Tabela 172 na stronie 1202, Tabela 173 na stronie 1203, Tabela 174 na stronie 1203i Tabela 175 na stronie 1203.

Należy zmodyfikować kod JCL uruchamiania dostarczony dla przykładów, które mają być używane (patrz Tabela 172 na stronie 1202, Tabela 173 na stronie 1203, Tabela 174 na stronie 1203i Tabela 175 na stronie 1203 ).

Instrukcja PARM w dostarczonym JCL zawiera pewną liczbę parametrów, które należy zmodyfikować. Aby uruchomić przykładowe programy w języku C, należy rozdzielić parametry spacjami; aby uruchomić przykładowe programy w językach COBOL i PL/I, należy rozdzielić je przecinkami. Jeśli na przykład nazwa menedżera kolejek to CSQ1 , a użytkownik chce uruchomić aplikację z kolejką o nazwie LOCALQ1w kodzie JCL w języku COBOL, PL/I oraz w języku asembler, instrukcja PARM powinna wyglądać następująco:

```
PARM=(CSQ1,LOCALQ1)
```

W języku C JCL instrukcja PARM powinna wyglądać następująco:

```
PARM=('CSQ1 LOCALQ1')
```

Teraz można wprowadzić zadania.

z/OS *Nazwy przykładowych aplikacji wsadowych w systemie z/OS*

Podsumowanie programów dostarczanych dla przykładowych aplikacji wsadowych.

Aplikacje wsadowe są podsumowane w następujących tabelach:

- Tabela 172 na stronie 1202 Pobieranie i umieszczanie przykładów
- Tabela 173 na stronie 1203 Przykład przeglądania
- Tabela 174 na stronie 1203 Przykład drukowania komunikatu
- Przykłady publikowania/subskrypcji w systemie Tabela 175 na stronie 1203
- Tabela 176 na stronie 1204 Inne przykłady

*Tabela 172. Próbki wsadu i pobrania*

| **Nazwa elementu klastra** | **Dla języka** | **Opis** | **Zbiór źródłowy podany w bibliotece** | **Plik wykonywalny podany w bibliotece** |
|---|---|---|---|---|
| CSQ4BCJ1 | C | Pobierz program źródłowy | SCSQC37S | SCSQLOAD, |
| CSQ4BCK1 | C | Umieść program źródłowy | SCSQC37S | SCSQLOAD, |
| CSQ4BCJR | C | Przykładowe uruchomienie JCL dla CSQ4BCJ1 i CSQBCK1 | SCSQPROC | Brak |
| CSQ4BVJ1 | kompilatory | Pobierz program źródłowy | SCSQKOB | SCSQLOAD, |
| CSQ4BVK1 | kompilatory | Umieść program źródłowy | SCSQKOB | SCSQLOAD, |
| CSQ4BVJR | kompilatory | Przykładowe uruchomienie JCL dla CSQBVJ1 i CSQBVK1 | SCSQPROC | Brak |

*Tabela 173. Przykład przeglądania wsadowego*

| Nazwa elementu klastra | Dla języka | Opis | Zbiór źródłowy podany w bibliotece | Plik wykonywalny podany w bibliotece |
|---|---|---|---|---|
| CSQ4BVA1 | kompilatory | Program źródłowy | SCSQKOB | SCSQLOAD, |
| CSQ4BVAR | kompilatory | Przykładowe uruchomienie JCL dla CSQ4BVA1 | SCSQPROC | Brak |
| CSQ4BAA1 | Assembler | Program źródłowy | SCSQASMS, | SCSQLOAD, |
| CSQ4BAAR | Assembler | Przykładowe uruchomienie JCL dla CSQ4BAA1 | SCSQPROC | Brak |
| CSQ4BCA1 | C | Program źródłowy | SCSQC37S | SCSQLOAD, |
| CSQ4BCAR | C | Przykładowe uruchomienie JCL dla CSQ4BCA1 | SCSQPROC | Brak |
| CSQ4BPA1 | PL/I | Program źródłowy | SCSQPLIS, | SCSQLOAD, |
| CSQ4BPAR | PL/I | Przykładowe uruchomienie JCL dla CSQ4BPA1 | SCSQPROC | Brak |

*Tabela 174. Przykład komunikatu drukowania wsadowego (tylko w języku C)*

| Nazwa elementu klastra | Opis | Zbiór źródłowy podany w bibliotece | Plik wykonywalny podany w bibliotece |
|---|---|---|---|
| CSQ4BCG1 | Program źródłowy | SCSQC37S | SCSQLOAD, |
| CSQ4BCGR | Przykładowe uruchomienie JCL dla CSQ4BCG1 | SCSQPROC | Brak |
| CSQ4BCL1 | Przeglądaj program źródłowy | SCSQC37S | SCSQLOAD, |
| CSQ4BCLR | Przykładowe uruchomienie JCL dla CSQ4BCL1 | SCSQPROC | Brak |

*Tabela 175. Przykłady publikowania/subskrypcji*

| Nazwa elementu klastra | Dla języka | Opis | Zbiór źródłowy podany w bibliotece | JCL w SCSQPROC | Plik wykonywalny podany w bibliotece |
|---|---|---|---|---|---|
| CSQ4BCP1 | C | Publikuj do programu źródłowego tematu | SCSQC37S | CSQ4BCPP | SCSQLOAD, |
| CSQ4BCP2 | C | Subskrybuj temat i pobierz program źródłowy komunikatów | SCSQC37S | CSQ4BCPS | SCSQLOAD, |

*Tabela 175. Przykłady publikowania/subskrypcji (kontynuacja)*

| Nazwa elementu klastra | Dla języka | Opis | Zbiór źródłowy podany w bibliotece | JCL w SCSQPROC | Plik wykonywalny podany w bibliotece |
|---|---|---|---|---|---|
| CSQ4BCP3 | C | Subskrybuj temat przy użyciu miejsca docelowego podanego przez użytkownika i pobierz program źródłowy komunikatów | SCSQC37S | CSQ4BCPD | SCSQLOAD, |
| CSQ4BCP4 | C | Subskrybuj temat przy użyciu opcji rozszerzonych i pobierz program źródłowy komunikatów | SCSQC37S | CSQ4BCPE | SCSQLOAD, |
| CSQ4BVP1 | kompilatory | Publikuj do programu źródłowego tematu | SCSQKOB | CSQ4BVPP | SCSQLOAD, |
| CSQ4BVP2 | kompilatory | Subskrybuj temat i pobierz program źródłowy komunikatów | SCSQKOB | CSQ4BVPS | SCSQLOAD, |

*Tabela 176. Inne próbki*

| Nazwa elementu klastra | Dla języka | Opis | Zbiór źródłowy podany w bibliotece | JCL w SCSQPROC | Plik wykonywalny podany w bibliotece |
|---|---|---|---|---|---|
| CSQ4BCS1 | C | Program źródłowy wykorzystania asynchronicznego | SCSQC37S | CSQ4BCSC | SCSQLOAD, |
| CSQ4BCS2 | C | Asynchroniczne umieszczanie i sprawdzanie statusu programu źródłowego | SCSQC37S | CSQ4BCSP | SCSQLOAD, |
| CSQ4BCM1 | C | Sprawdź program źródłowy właściwości komunikatu | SCSQC37S | CSQ4BCMP | SCSQLOAD, |
| CSQ4BCM2 | C | Ustaw program źródłowy właściwości komunikatu | SCSQC37S | CSQ4BCMP | SCSQLOAD, |

### z/OS *Przygotowywanie przykładowych aplikacji dla środowiska TSO w systemie z/OS*

Aby przygotować przykładową aplikację, która działa w środowisku TSO, należy wykonać te same kroki, co podczas budowania dowolnej aplikacji wsadowej IBM MQ for z/OS .

Te kroki są wymienione w sekcji “Budowanie aplikacji wsadowych z/OS” na stronie 1054. Lista elementów biblioteki, które mają być używane, znajduje się w sekcji Tabela 177 na stronie 1205.

Alternatywnie, jeśli dostarczamy wykonywalną formę przykładu, można ją uruchomić z biblioteki dynamicznej thlqual.SCSQLOAD .

W przypadku przykładowej aplikacji Mail Manager upewnij się, że używane przez nią kolejki są dostępne w systemie. Są one zdefiniowane w elemencie **thlqual**.SCSQPROC (CSQ4CVD). Aby upewnić się, że te kolejki są zawsze dostępne, można dodać te elementy do wejściowego zestawu danych inicjowania CSQINP2 lub użyć programu CSQUTIL w celu załadowania tych definicji kolejek.

z/OS *Nazwy przykładowych aplikacji TSO w systemie z/OS*

Informacje o nazwach programów, które są dostarczane dla każdej przykładowej aplikacji TSO, oraz o bibliotekach, w których znajdują się pliki wykonywalne (kod źródłowy, kod JCL i tylko dla przykładowej procedury obsługi komunikatów).

Aplikacje TSO zostały podsumowane w następujących tabelach:

- Przykład menedżera poczty Tabela 177 na stronie 1205
- Tabela 178 na stronie 1206 Przykład procedury obsługi komunikatów

W tych przykładach używane są panele ISPF . Dlatego podczas konsolidacji programów należy uwzględnić kod pośredniczący ISPF , ISPLINK.

*Tabela 177. Przykład menedżera poczty TSO*

| **Nazwa elementu klastra** | **Dla języka** | **Opis** | **Zbiór źródłowy podany w bibliotece** |
|---|---|---|---|
| CSQ4CVD | Niezależne | Definicje obiektów IBM MQ for z/OS | SCSQPROC |
| CSQ40 | Niezależne | Komunikaty ISPF | SCSQMSGE |
| CSQ4RVD1 | kompilatory | CLIST w celu zainicjowania CSQ4TVD1 | SCSQCLST, |
| CSQ4TVD1 | kompilatory | Program źródłowy dla programu Menu | SCSQKOB |
| CSQ4TVD2 | kompilatory | Program źródłowy dla programu Get Mail | SCSQKOB |
| CSQ4TVD4 | kompilatory | Program źródłowy programu Send Mail | SCSQKOB |
| CSQ4TVD5 | kompilatory | Program źródłowy dla programu Nickname | SCSQKOB |
| CSQ4VDP1-6 | kompilatory | Definicje paneli | SCSQPNLA, |
| CSQ4VD0 | kompilatory | Definicja danych | SCSQCOBC, |
| CSQ4VD1 | kompilatory | Definicja danych | SCSQCOBC, |
| CSQ4VD2 | kompilatory | Definicja danych | SCSQCOBC, |
| CSQ4VD4 | kompilatory | Definicja danych | SCSQCOBC, |
| CSQ4RCD1 | C | CLIST w celu zainicjowania CSQ4TCD1 | SCSQCLST, |
| CSQ4TCD1 | C | Program źródłowy dla programu Menu | SCSQC37S |

| *Tabela 177. Przykład menedżera poczty TSO (kontynuacja)* | | | |
|---|---|---|---|
| **Nazwa elementu klastra** | **Dla języka** | **Opis** | **Zbiór źródłowy podany w bibliotece** |
| CSQ4TCD2 | C | Program źródłowy dla programu Get Mail | SCSQC37S |
| CSQ4TCD4 | C | Program źródłowy programu Send Mail | SCSQC37S |
| CSQ4TCD5 | C | Program źródłowy dla programu Nickname | SCSQC37S |
| CSQ4CDP1-6 | C | Definicje paneli | SCSQPNLA, |
| CSQ4TC0 | C | Plik włączkowy | SCSQC370 |

| *Tabela 178. Przykład procedury obsługi komunikatów TSO* | | | | |
|---|---|---|---|---|
| **Nazwa elementu klastra** | **Dla języka** | **Opis** | **Zbiór źródłowy podany w bibliotece** | **Plik wykonywalny podany w bibliotece** |
| CSQ4TCH0 | C | Definicja danych | SCSQC370 | Brak |
| CSQ4TCH1 | C | Program źródłowy | SCSQC37S | SCSQLOAD, |
| CSQ4TCH2 | C | Program źródłowy | SCSQC37S | SCSQLOAD, |
| CSQ4TCH3 | C | Program źródłowy | SCSQC37S | SCSQLOAD, |
| CSQ4RCH1 | C i COBOL | CLIST w celu zainicjowania CSQ4TCH1 lub CSQ4TVH1 | SCSQCLST, | Brak |
| CSQ4CHP1 | C i COBOL | Definicja panelu | SCSQPNLA, | Brak |
| CSQ4CHP2 | C i COBOL | Definicja panelu | SCSQPNLA, | Brak |
| CSQ4CHP3 | C i COBOL | Definicja panelu | SCSQPNLA, | Brak |
| CSQ4CHP9 | C i COBOL | Definicja panelu | SCSQPNLA, | Brak |
| CSQ4TVH0 | kompilatory | Definicja danych | SCSQCOBC, | Brak |
| CSQ4TVH1 | kompilatory | Program źródłowy | SCSQKOB | SCSQLOAD, |
| CSQ4TVH2 | kompilatory | Program źródłowy | SCSQKOB | SCSQLOAD, |
| CSQ4TVH3 | kompilatory | Program źródłowy | SCSQKOB | SCSQLOAD, |

## z/OS *Przygotowywanie przykładowych aplikacji dla środowiska CICS w systemie z/OS*

Przed uruchomieniem przykładowych programów CICS należy zalogować się do systemu CICS , używając parametru LOGMODE o wartości 32702. Wynika to z faktu, że przykładowe programy zostały napisane w taki sposób, aby korzystały z ekranu trybu 3270 2.

Aby przygotować przykładową aplikację, która działa w środowisku CICS , wykonaj następujące kroki:

1. Utwórz symboliczne odwzorowanie opisu i fizyczne odwzorowanie ekranu dla przykładu, składając źródło definicji ekranu BMS (dostarczane w bibliotece **thlqual**.SCSQMAPS, gdzie **thlqual** jest kwalifikatorem wysokiego poziomu używanym przez instalację). Podczas tworzenia nazw odwzorowań

należy użyć nazwy źródła definicji ekranu BMS (niedostępne dla programów przykładowych Put i Get), ale należy pominąć ostatni znak tej nazwy.

2. Należy wykonać te same kroki, które były wykonywane podczas budowania dowolnej aplikacji produktu CICS IBM MQ for z/OS . Te kroki są wymienione w sekcji “Budowanie aplikacji CICS w produkcie z/OS” na stronie 1057. Lista elementów biblioteki, które mają być używane, znajduje się w sekcji Tabela 179 na stronie 1207, Tabela 180 na stronie 1208, Tabela 181 na stronie 1208i Tabela 182 na stronie 1209.

   Alternatywnie, jeśli dostarczamy kod wykonywalny przykładu, można go uruchomić z biblioteki dynamicznej thlqual.SCSQCICS .

3. Zidentyfikuj zestaw map, programy i transakcje do CICS , aktualizując zestaw danych definicji systemu CICS (CSD). Wymagane definicje znajdują się w elemencie **thlqual**.SCSQPROC (CSQ4S100). Wskazówki na ten temat zawiera sekcja *The CICS-IBM MQ Adapter* w dokumentacji produktu CICS Transaction Server for z/OS 4.1 pod adresem: CICS Transaction Server for z/OS 4.1, The CICS-IBM MQ adapter.

   **Uwaga:** W przypadku przykładowej aplikacji sprawdzania zdolności kredytowej na tym etapie wyświetlany jest komunikat o błędzie, jeśli nie utworzono jeszcze zestawu danych VSAM używanego przez przykład.

4. W przypadku aplikacji przykładowych Credit Check i Mail Manager upewnij się, że używane przez nie kolejki są dostępne w systemie. W przypadku przykładu Credit Check są one zdefiniowane w podzbiorze **thlqual**.SCSQPROC (CSQ4CVB) dla języka COBOL i **thlqual**.SCSQPROC (CSQ4CCB) dla języka C. Dla przykładu Mail Manager są one zdefiniowane w elemencie **thlqual**.SCSQPROC (CSQ4CVD). Aby upewnić się, że te kolejki są zawsze dostępne, można dodać te elementy do wejściowego zestawu danych inicjowania CSQINP2 lub użyć programu CSQUTIL w celu załadowania tych definicji kolejek.

   W przypadku przykładowej aplikacji Atrybuty kolejki można użyć jednej lub większej liczby kolejek, które są dostarczane dla innych przykładowych aplikacji. Można również użyć własnych kolejek. Jednak w postaci, w jakiej został podany, ten przykład działa tylko z kolejkami, które zawierają znaki CSQ4SAMP w pierwszych ośmiu bajtach nazwy.

z/OS *Nazwy przykładowych aplikacji CICS w systemie z/OS*

Ten temat zawiera podsumowanie programów dostarczonych dla przykładowych aplikacji CICS .

Podsumowanie aplikacji CICS znajduje się w następujących tabelach:

- Tabela 179 na stronie 1207 Pobieranie i umieszczanie przykładów
- Tabela 180 na stronie 1208 Przykład atrybutów kolejki
- Tabela 181 na stronie 1208 Mail Manager-przykład (tylko w języku COBOL)
- Tabela 182 na stronie 1209 Przykład sprawdzania zdolności kredytowej
- Tabela 183 na stronie 1210 Przykłady wykorzystania asynchronicznego i publikowania/subskrypcji

*Tabela 179. CICS Pobieranie i umieszczanie przykładów*

| **Nazwa elementu klastra** | **Dla języka** | **Opis** | **Zbiór źródłowy podany w bibliotece** | **Plik wykonywalny podany w bibliotece** |
|---|---|---|---|---|
| CSQ4CCK1 | C | Umieść program źródłowy | SCSQC37S | SCSQ (SCSQ)CICS |
| CSQ4CCJ1 | C | Pobierz program źródłowy | SCSQC37S | SCSQ (SCSQ)CICS |
| CSQ4CVJ1 | kompilatory | Pobierz program źródłowy | SCSQKOB | SCSQ (SCSQ)CICS |

*Tabela 179. CICS Pobieranie i umieszczanie przykładów (kontynuacja)*

| Nazwa elementu klastra | Dla języka | Opis | Zbiór źródłowy podany w bibliotece | Plik wykonywalny podany w bibliotece |
|---|---|---|---|---|
| CSQ4CVK1 | kompilatory | Umieść program źródłowy | SCSQKOB | SCSQ (SCSQ)CICS |
| CSQ4S100 | Niezależne | Zestaw danych definicji systemu CICS | SCSQPROC | Brak |

*Tabela 180. CICS Przykład atrybutów kolejki*

| Nazwa elementu klastra | Dla języka | Opis | Zbiór źródłowy podany w bibliotece | Plik wykonywalny podany w bibliotece |
|---|---|---|---|---|
| CSQ4CVC1 | kompilatory | Program źródłowy | SCSQKOB | SCSQ (SCSQ)CICS |
| CSQ4VMSG | kompilatory | Definicja komunikatu | SCSQCOBC, | Brak |
| CSQ4VCMS | kompilatory | Definicja ekranu BMS | SCSQMAPY | SCSQCICS (o nazwie CSQ4ACM) |
| CSQ4CAC1 | Assembler | Program źródłowy | SCSQASMS, | SCSQ (SCSQ)CICS |
| CSQ4AMSG | Assembler | Definicja komunikatu | SCSQMACY | Brak |
| CSQ4ACMS | Assembler | Definicja ekranu BMS | SCSQMAPY | SCSQCICS (o nazwie CSQ4ACM) |
| CSQ4CCC1 | C | Program źródłowy | SCSQC37S | SCSQ (SCSQ)CICS |
| CSQ4CMSG | C | Definicja komunikatu | SCSQC370 | Brak |
| CSQ4CCMS | C | Definicja ekranu BMS | SCSQMAPY | SCSQCICS (o nazwie CSQ4ACM) |
| CSQ4S100 | Niezależne | Zestaw danych definicji systemu CICS | SCSQPROC | Brak |

*Tabela 181. CICS Mail Manager-przykład (tylko w języku COBOL)*

| Nazwa elementu klastra | Opis | Zbiór źródłowy podany w bibliotece |
|---|---|---|
| CSQ4CVD | Definicje obiektów IBM MQ for z/OS | SCSQPROC |
| CSQ4CVD1 | Źródło programu Menu | SCSQKOB |
| CSQ4CVD2 | Źródło programu Get Mail | SCSQKOB |
| CSQ4CVD3 | Źródło programu wyświetlania komunikatów | SCSQKOB |
| CSQ4CVD4 | Źródło programu Send Mail | SCSQKOB |
| CSQ4CVD5 | Źródło programu Nickname | SCSQKOB |

*Tabela 181. CICS Mail Manager-przykład (tylko w języku COBOL) (kontynuacja)*

| Nazwa elementu klastra | Opis | Zbiór źródłowy podany w bibliotece |
|---|---|---|
| CSQ4VDMS | Źródło definicji ekranu BMS | SCSQMAPY |
| CSQ4S100 | Zestaw danych definicji systemu CICS | SCSQPROC |
| CSQ4VD0 | Definicja danych | SCSQCOBC, |
| CSQ4VD3 | Definicja danych | SCSQCOBC, |
| CSQ4VD4 | Definicja danych | SCSQCOBC, |

*Tabela 182. CICS Przykład sprawdzania zdolności kredytowej*

| Nazwa elementu klastra | Dla języka | Opis | Zbiór źródłowy podany w bibliotece |
|---|---|---|---|
| CSQ4CVB | Niezależne | Definicje obiektów IBM MQ | SCSQPROC |
| CSQ4CCB | Niezależne | Definicje obiektów IBM MQ | SCSQPROC |
| CSQ4CVB1 | kompilatory | Źródło programu interfejsu użytkownika | SCSQKOB |
| CSQ4CVB2 | kompilatory | Źródło dla menedżera aplikacji kredytowych | SCSQKOB |
| CSQ4CVB3 | kompilatory | Źródło programu konta rozliczeniowego | SCSQKOB |
| CSQ4CVB4 | kompilatory | Źródło programu dystrybucyjnego | SCSQKOB |
| CSQ4CVB5 | kompilatory | Źródło dla agencji-program zapytania | SCSQKOB |
| CSQ4CCB1 | C | Źródło programu interfejsu użytkownika | SCSQC37S |
| CSQ4CCB2 | C | Źródło dla menedżera aplikacji kredytowych | SCSQC37S |
| CSQ4CCB3 | C | Źródło programu konta rozliczeniowego | SCSQC37S |
| CSQ4CCB4 | C | Źródło programu dystrybucyjnego | SCSQC37S |
| CSQ4CCB5 | C | Źródło dla agencji-program zapytania | SCSQC37S |
| CSQ4CB0 | C | Plik włączkowy | SCSQC370 |
| CSQ4CBMS | C | Źródło definicji ekranu BMS | SCSQMAPY |
| CSQ4VBMS | kompilatory | Źródło definicji ekranu BMS | SCSQMAPY |
| CSQ4VB0 | kompilatory | Definicja danych | SCSQCOBC, |
| CSQ4VB1 | kompilatory | Definicja danych | SCSQCOBC, |
| CSQ4VB2 | kompilatory | Definicja danych | SCSQCOBC, |
| CSQ4VB3 | kompilatory | Definicja danych | SCSQCOBC, |
| CSQ4VB4 | kompilatory | Definicja danych | SCSQCOBC, |
| CSQ4VB5 | kompilatory | Definicja danych | SCSQCOBC, |
| CSQ4VB6 | kompilatory | Definicja danych | SCSQCOBC, |
| CSQ4VB7 | kompilatory | Definicja danych | SCSQCOBC, |

| Tabela 182. CICS Przykład sprawdzania zdolności kredytowej (kontynuacja) | | | |
|---|---|---|---|
| **Nazwa elementu klastra** | **Dla języka** | **Opis** | **Zbiór źródłowy podany w bibliotece** |
| CSQ4VB8 | kompilatory | Definicja danych | SCSQCOBC, |
| CSQ4BAQ | Niezależne | Źródło zestawu danych VSAM | SCSQPROC |
| CSQ4FILE | Niezależne | JCL do budowania zestawu danych VSAM używanego przez CSQ4CVB3 | SCSQPROC |
| CSQ4S100 | Niezależne | Zestaw danych definicji systemu CICS | SCSQPROC |

| Tabela 183. CICS Przykłady wykorzystania asynchronicznego i publikowania/subskrypcji | | |
|---|---|---|
| **Nazwa elementu klastra** | **Opis** | **Zbiór źródłowy podany w bibliotece** |
| CSQ4CVCN | Źródło prostego programu do obsługi komunikatów | SCSQKOB |
| CSQ4CVCT | Źródło programu kontroli wykorzystania komunikatów | SCSQKOB |
| CSQ4CVEV | Źródło programu obsługi zdarzeń | SCSQKOB |
| CSQ4CVPT | Źródło programu klienta umieszczania komunikatów | SCSQKOB |
| CSQ4CVRG | Źródło programu klienta rejestracji | SCSQKOB |
| CSQ4S100 | Zestaw danych Definicja systemu CICS | SCSQPROC |

## z/OS *Przygotowywanie przykładowej aplikacji dla środowiska IMS w systemie z/OS*

Część przykładowej aplikacji Credit Check może działać w środowisku IMS .

Aby przygotować tę część aplikacji do działania z przykładem CICS , należy najpierw wykonać kroki opisane w sekcji “Przygotowywanie przykładowych aplikacji dla środowiska CICS w systemie z/OS” na stronie 1206.

Następnie należy wykonać następujące kroki:

1. Należy wykonać te same kroki, które były wykonywane podczas budowania dowolnej aplikacji produktu IMS IBM MQ for z/OS . Te kroki są wymienione w sekcji “Budowanie aplikacji IMS (BMP lub MPP)” na stronie 1058. Lista elementów biblioteki, które mają być używane, znajduje się w sekcji Tabela 184 na stronie 1211.
2. Zidentyfikuj aplikację i bazę danych dla IMS. W celu włączenia tej opcji przykłady są dostarczane z instrukcjami PSBGEN, DBDGEN, ACB, IMSGEN i IMSDALOC.
3. Załaduj bazę danych CSQ4CA , dostosowując i uruchamiając przykładowy kod JCL udostępniony w tym celu (CSQ4ILDB). Ten kod JCL ładuje bazę danych z pliku CSQ4BAQ. Zaktualizuj region sterujący IMS za pomocą instrukcji DD dla bazy danych CSQ4CA.
4. Uruchom program konta rozliczeniowego jako program przetwarzania komunikatów wsadowych (Batch Message Processing-BMP), dostosowując i uruchamiając przykładowy kod JCL udostępniony w tym celu. Ten kod JCL uruchamia program BMP zorientowany wsadowo. Aby uruchomić program jako program BMP zorientowany na komunikaty, usuń znaki komentarza z wiersza w JCL, który zawiera instrukcję IN=.

z/OS *Nazwy przykładowej aplikacji IMS w systemie z/OS*

Te informacje zawierają tabelę z listą źródeł i JCL dostarczonych dla przykładowej aplikacji IMS sprawdzania kredytu.

*Tabela 184. Źródło i JCL dla przykładu sprawdzania kredytu IMS (tylko C)*

| Nazwa elementu klastra | Opis | Dostarczane w bibliotece |
|---|---|---|
| CSQ4CVB | Definicje obiektów IBM MQ | SCSQPROC |
| CSQ4ICB3 | Źródło programu konta rozliczeniowego | SCSQC37S |
| CSQ4ICBL | Źródło ładowania bazy danych kont kontrolnych | SCSQC37S |
| CSQ4CBI | Definicja danych | SCSQC370 |
| CSQ4PSBL | PSBGEN JCL dla programu ładowania bazy danych | SCSQPROC |
| CSQ4PSB3 | PSBGEN JCL dla programu checking-account | SCSQPROC |
| CSQ4DBDS | DBDGEN JCL dla bazy danych CSQ4CA | SCSQPROC |
| CSQ4GIMS | IMSDefinicje makr GEN dla CSQ4IVB3 i CSQ4CA | SCSQPROC |
| CSQ4ACBG | Definicja bloku kontrolnego aplikacji (ACB) dla CSQ4IVB3 | SCSQPROC |
| CSQ4BAQ | Źródło bazy danych | SCSQPROC |
| CSQ4ILDB | Przykładowe uruchomienie JCL dla zadania ładowania bazy danych | SCSQPROC |
| CSQ4ICBR | Przykładowy kod JCL uruchamiania dla programu do obsługi kont kontrolnych | SCSQPROC |
| CSQ4DYNA | Definicje makr produktu IMSDALOC dla bazy danych | SCSQPROC |

## z/OS *Przykłady Put w systemie z/OS*

Przykładowe programy wstawiające komunikaty do kolejki za pomocą wywołania MQPUT.

Programy źródłowe są dostarczane w językach C i COBOL w środowiskach wsadowych i CICS (patrz Tabela 172 na stronie 1202 i Tabela 179 na stronie 1207 ).

### Projekt próbki odkładanej

Przepływ przez logikę programu jest następujący:

1. Nawiąż połączenie z menedżerem kolejek przy użyciu wywołania MQCONN. Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny oraz zatrzymaj przetwarzanie.

   **Uwaga:** Jeśli przykład jest uruchamiany w środowisku CICS , nie trzeba wywoływać wywołania MQCONN. W przeciwnym razie zwraca wartość DEF_HCONN. W przypadku kolejnych wywołań MQI można użyć uchwytu połączenia MQHC_DEF_HCONN.

2. Otwórz kolejkę za pomocą wywołania MQOPEN z opcją MQOO_OUTPUT. Na wejściu do tego wywołania program używa uchwytu połączenia zwróconego w kroku “1” na stronie 1214. W przypadku struktury deskryptora obiektu (MQOD) używane są wartości domyślne dla wszystkich pól z wyjątkiem pola nazwy kolejki, które jest przekazywane jako parametr do programu. Jeśli wywołanie MQOPEN nie powiedzie się, wydrukuj kody zakończenia i przyczyny oraz zatrzymaj przetwarzanie.
3. Utwórz pętlę w programie wysyłającym wywołania MQPUT do momentu umieszczenia w kolejce wymaganej liczby komunikatów. Jeśli wywołanie MQPUT nie powiedzie się, pętla zostanie wcześniej porzucona, nie będzie podejmowana żadna kolejna próba wywołania MQPUT i zostaną zwrócone kody zakończenia i przyczyny.
4. Zamknij kolejkę za pomocą wywołania MQCLOSE z uchwytem obiektu zwróconym w kroku “2” na stronie 1214. Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny.
5. Rozłącz się z menedżerem kolejek przy użyciu wywołania MQDISC z uchwytem połączenia zwróconym w kroku “1” na stronie 1214. Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny.

   **Uwaga:** Jeśli przykład jest uruchamiany w środowisku CICS , nie jest konieczne wywoływanie komendy MQDISC.

z/OS *Przykłady umieszczania dla środowiska wsadowego w systemie z/OS*

Ten temat należy wykorzystać przy rozważaniu użycia opcji Put samples for the batch environment.

Aby uruchomić przykłady, zmodyfikuj i uruchom przykładowy kod JCL zgodnie z opisem w sekcji “Przygotowywanie i uruchamianie przykładowych aplikacji dla środowiska wsadowego w systemie z/OS” na stronie 1201.

Programy przyjmują następujące parametry w komendzie EXEC PARM, oddzielone spacjami w języku C i przecinkami w języku COBOL:

1. Nazwa menedżera kolejek (4 znaki)
2. Nazwa kolejki docelowej (48 znaków)
3. Liczba wiadomości (maksymalnie 4 cyfry)
4. Znak dopełniający, który ma zostać zapisany w komunikacie (1 znak)
5. Liczba znaków do zapisania w komunikacie (maksymalnie 4 cyfry)
6. Trwałość komunikatu (1 znak: P dla trwałych lub N dla nietrwałych)

Jeśli któryś z tych parametrów zostanie wprowadzony niepoprawnie, zostaną wyświetlone odpowiednie komunikaty o błędach.

Wszystkie komunikaty z przykładów są zapisywane w zestawie danych SYSPRINT.

## Użycie notatek

- Aby zachować prostotę przykładów, istnieją niewielkie różnice funkcjonalne między wersjami językowymi. Różnice te są jednak zminimalizowane, jeśli używany jest układ parametrów przedstawionych w przykładowych uruchomach JCL, CSQ4BCJRi CSQ4BVJR. Żadna z tych różnic nie dotyczy interfejsu MQI.
- CSQ4BCK1 umożliwia wprowadzenie więcej niż czterech cyfr dla liczby wysłanych komunikatów i długości komunikatów.
- W przypadku dwóch pól liczbowych należy wprowadzić dowolną cyfrę z zakresu od 1 do 9999. Wprowadzona wartość powinna być liczbą dodatnią. Na przykład, aby umieścić pojedynczy komunikat, można wpisać 1, 01, 001 lub 0001 jako wartość. Wprowadzenie wartości nieliczbowych lub ujemnych może spowodować wystąpienie błędu. Jeśli na przykład zostanie wprowadzona wartość -1, program w języku COBOL wyśle komunikat 1-bajtowy, ale program w języku C otrzyma błąd.
- W przypadku obu programów, CSQ4BCK1 i CSQ4BVK1, należy wprowadzić P w parametrze trwałości, + + PER + +, jeśli komunikat ma być trwały. Jeśli wykonanie tej czynności nie powiedzie się, komunikat będzie nietrwały.

z/OS *Przykłady umieszczania dla środowiska CICS w systemie z/OS*

Ten temat należy wykorzystać przy rozważaniu użycia opcji Put samples dla środowiska CICS .

Transakcje przyjmują następujące parametry rozdzielone przecinkami:

1. Liczba wiadomości (maksymalnie 4 cyfry)
2. Znak dopełniający, który ma zostać zapisany w komunikacie (1 znak)
3. Liczba znaków do zapisania w komunikacie (maksymalnie 4 cyfry)
4. Trwałość komunikatu (1 znak: P dla trwałych lub N dla nietrwałych)
5. Nazwa kolejki docelowej (48 znaków)

Jeśli któryś z tych parametrów zostanie wprowadzony niepoprawnie, zostaną wyświetlone odpowiednie komunikaty o błędach.

W przypadku przykładu COBOL wywołaj przykład Put w środowisku CICS , wprowadzając komendę:

```
MVPT,9999,*,9999,P,QUEUE.NAME
```

W przypadku przykładu w języku C wywołaj przykład Put w środowisku CICS , wprowadzając komendę:

```
MCPT,9999,*,9999,P,QUEUE.NAME
```

Wszystkie komunikaty z przykładów są wyświetlane na ekranie.

## Użycie notatek

- Aby zachować prostotę przykładów, istnieją niewielkie różnice funkcjonalne między wersjami językowymi. Żadna z tych różnic nie dotyczy interfejsu MQI.
- Jeśli zostanie wprowadzona nazwa kolejki dłuższa niż 48 znaków, jej długość zostanie obcięta do maksymalnie 48 znaków, ale nie zostanie zwrócony żaden komunikat o błędzie.
- Przed wprowadzeniem transakcji naciśnij klawisz CLEAR.
- W przypadku dwóch pól liczbowych należy wprowadzić dowolną liczbę z zakresu od 1 do 9999. Wprowadzona wartość powinna być liczbą dodatnią. Na przykład, aby umieścić pojedynczy komunikat, można wprowadzić wartość 1, 01, 001 lub 0001. Wprowadzenie wartości nieliczbowych lub ujemnych może spowodować wystąpienie błędu. Jeśli na przykład zostanie wprowadzona wartość -1, program w języku COBOL wyśle komunikat 1-bajtowy, a program w języku C zakończy działanie z błędem funkcji malloc ().
- W przypadku obu programów, CSQ4CCK1 i CSQ4CVK1, należy wprowadzić P w parametrze trwałości, jeśli komunikat ma być trwały. W przypadku nietrwałych komunikatów wprowadź wartość N w parametrze trwałości. Jeśli zostanie wprowadzona dowolna inna wartość, zostanie wyświetlony komunikat o błędzie.
- Komunikaty są umieszczane w punkcie synchronizacji, ponieważ wartości domyślne są używane dla wszystkich parametrów z wyjątkiem tych, które zostały ustawione podczas wywoływania programu.

## z/OS *Przykłady Get w serwisie z/OS*

Przykładowe programy pobierające komunikaty z kolejki za pomocą wywołania MQGET.

Programy źródłowe są dostarczane w językach C i COBOL w środowiskach wsadowych i CICS (patrz Tabela 172 na stronie 1202 i Tabela 179 na stronie 1207 ).

z/OS *Projekt przykładu Get w serwisie z/OS*

Zapoznaj się z projektem przykładu Get i kilkoma uwagami dotyczącymi użycia, które należy wziąć pod uwagę.

Przepływ przez logikę programu jest następujący:

1. Nawiąż połączenie z menedżerem kolejek przy użyciu wywołania MQCONN. Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny oraz zatrzymaj przetwarzanie.

   **Uwaga:** Jeśli przykład jest uruchamiany w środowisku CICS , nie trzeba wywoływać wywołania MQCONN. W przeciwnym razie zwraca wartość DEF_HCONN. W przypadku kolejnych wywołań MQI można użyć uchwytu połączenia MQHC_DEF_HCONN.
2. Otwórz kolejkę za pomocą wywołania MQOPEN z opcjami MQOO_INPUT_SHARED i MQOO_BROWSE. Na wejściu do tego wywołania program używa uchwytu połączenia zwróconego w kroku “1” na stronie 1214. W przypadku struktury deskryptora obiektu (MQOD) używane są wartości domyślne dla wszystkich pól z wyjątkiem pola nazwy kolejki, które jest przekazywane jako parametr do programu. Jeśli wywołanie MQOPEN nie powiedzie się, wydrukuj kody zakończenia i przyczyny oraz zatrzymaj przetwarzanie.
3. Utwórz pętlę w programie wywołującym wywołania MQGET do momentu pobrania z kolejki wymaganej liczby komunikatów. Jeśli wywołanie MQGET nie powiedzie się, pętla zostanie wcześniej porzucona, nie będzie podejmowana żadna kolejna próba wywołania MQGET i zostaną zwrócone kody zakończenia i przyczyny. W wywołaniu MQGET określono następujące opcje:

   - MQGMO_NO_WAIT
   - Komunikat MQGMO_ACCEPT_TRUNCATED_MESSAGE
   - MQGMO_SYNCPOINT lub MQGMO_NO_SYNCPOINT
   - MQGMO_BROWSE_FIRST i MQGMO_BROWSE_NEXT

   Opis tych opcji zawiera sekcja MQGET. Dla każdego komunikatu drukowany jest numer komunikatu, po którym następuje długość komunikatu i dane komunikatu.
4. Zamknij kolejkę za pomocą wywołania MQCLOSE z uchwytem obiektu zwróconym w kroku “2” na stronie 1214. Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny.
5. Rozłącz się z menedżerem kolejek przy użyciu wywołania MQDISC z uchwytem połączenia zwróconym w kroku “1” na stronie 1214. Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny.

   **Uwaga:** Jeśli przykład jest uruchamiany w środowisku CICS , nie jest konieczne wywoływanie komendy MQDISC.

## Użycie notatek

- Aby zachować prostotę przykładów, istnieją niewielkie różnice funkcjonalne między wersjami językowymi. Jednak te różnice są minimalizowane, jeśli używany jest układ parametrów pokazany w przykładowym uruchomieniu JCL, CSQ4BCJRi CSQ4BVJR. Żadna z tych różnic nie dotyczy interfejsu MQI.
- CSQ4BCJ1 umożliwia wprowadzenie więcej niż czterech cyfr dla liczby pobieranych komunikatów.
- Komunikaty dłuższe niż 64 kB są obcinane.
- Komenda CSQ4BCJ1 może poprawnie wyświetlać tylko komunikaty znakowe, ponieważ jest ona wyświetlana tylko do momentu wyświetlenia pierwszego znaku NULL (\0).
- W polu liczbowym liczba komunikatów wpisz dowolną cyfrę z zakresu od 1 do 9999. Wprowadzona wartość powinna być liczbą dodatnią. Na przykład, aby otrzymać pojedynczy komunikat, można wpisać 1, 01, 001 lub 0001 jako wartość. Wprowadzenie wartości nieliczbowych lub ujemnych może spowodować wystąpienie błędu. Jeśli na przykład zostanie wprowadzona wartość -1, program w języku COBOL pobierze jeden komunikat, ale program w języku C nie pobierze żadnych komunikatów.
- Dla obu programów, CSQ4BCJ1 i CSQ4BVJ1, wpisz B w parametrze get, + + GET + +, jeśli chcesz przeglądać komunikaty.
- W przypadku obu programów, CSQ4BCJ1 i CSQ4BVJ1, wpisz S w parametrze syncpoint, + + SYNC + +, dla komunikatów, które mają zostać pobrane w punkcie synchronizacji.

z/OS *Przykłady pobierania dla środowiska wsadowego w systemie z/OS*

Aby uruchomić przykłady, zmodyfikuj i uruchom przykładowy kod JCL zgodnie z opisem w sekcji “Przygotowywanie i uruchamianie przykładowych aplikacji dla środowiska wsadowego w systemie z/OS” na stronie 1201.

Programy przyjmują następujące parametry w komendzie EXEC PARM, oddzielone spacjami w języku C i przecinkami w języku COBOL:

1. Nazwa menedżera kolejek (4 znaki)
2. Nazwa kolejki docelowej (48 znaków)
3. Liczba wiadomości do pobrania (maksymalnie 4 cyfry)
4. Opcja przeglądania/pobierania komunikatu (1 znak: B do przeglądania lub D do destrukcyjnego pobierania komunikatów)
5. Sterowanie punktem synchronizacji (1 znak: S dla punktu synchronizacji lub N dla punktu synchronizacji)

Jeśli któryś z tych parametrów zostanie wprowadzony niepoprawnie, zostaną wyświetlone odpowiednie komunikaty o błędach.

Dane wyjściowe z próbek są zapisywane w zestawie danych SYSPRINT:

```
==================================
PARAMETERS PASSED :
QMGR    - VC9
QNAME    - A.Q
NUMMSGS   - 000000002
GET     - D
SYNCPOINT  - N
==================================
MQCONN SUCCESSFUL
MQOPEN SUCCESSFUL
000000000 : 000000010 : **********
000000001 : 000000010 : **********
000000002 MESSAGES GOT FROM QUEUE
MQCLOSE SUCCESSFUL
MQDISC SUCCESSFUL
```

z/OS *Przykłady pobierania dla środowiska CICS w systemie z/OS*

Specjalne uwagi dotyczące pobierania przykładów dla środowiska CICS .

Transakcje przyjmują następujące parametry w EXEC PARM, oddzielone przecinkami:

1. Liczba wiadomości do pobrania (maksymalnie cztery cyfry)
2. Opcja przeglądania/pobierania komunikatów (jeden znak: B, aby przeglądać lub D, aby destrukcyjnie pobrać komunikaty)
3. Sterowanie punktem synchronizacji (jeden znak: S dla punktu synchronizacji lub N dla punktu synchronizacji)
4. Nazwa kolejki docelowej (48 znaków)

Jeśli któryś z tych parametrów zostanie wprowadzony niepoprawnie, zostaną wyświetlone odpowiednie komunikaty o błędach.

W przypadku przykładu COBOL wywołaj przykład Get w środowisku CICS , wpisując:

```
MVGT,9999,B,S,QUEUE.NAME
```

W przypadku przykładu w języku C wywołaj przykład Get w środowisku CICS , wpisując:

```
MCGT,9999,B,S,QUEUE.NAME
```

Po pobraniu komunikatów z kolejki są one umieszczane w kolejce pamięci tymczasowej systemu CICS o nazwie takiej samej jak nazwa transakcji CICS (na przykład MCGT dla przykładu C).

Poniżej przedstawiono przykładowe dane wyjściowe komendy Get samples:

```
*************************** TOP OF QUEUE ************************
000000000 : 000000010: **********
000000001 : 000000010 :**********
*************************** BOTTOM OF QUEUE *********************
```

## Użycie notatek

- Aby zachować prostotę przykładów, istnieją niewielkie różnice funkcjonalne między wersjami językowymi. Żadna z tych różnic nie dotyczy interfejsu MQI.
- Jeśli zostanie wprowadzona nazwa kolejki dłuższa niż 48 znaków, jej długość zostanie obcięta do maksymalnie 48 znaków, ale nie zostanie zwrócony żaden komunikat o błędzie.
- Przed wprowadzeniem transakcji naciśnij klawisz CLEAR.
- Komenda CSQ4CCJ1 może poprawnie wyświetlać tylko komunikaty znakowe, ponieważ jest ona wyświetlana tylko do momentu wyświetlenia pierwszego znaku NULL (\0).
- W polu liczbowym należy wprowadzić dowolną liczbę z zakresu od 1 do 9999. Wprowadzona wartość powinna być liczbą dodatnią. Na przykład, aby uzyskać pojedynczy komunikat, można wprowadzić wartość 1, 01, 001 lub 0001. Wprowadzenie wartości nieliczbowej lub ujemnej może spowodować wystąpienie błędu.
- Komunikaty dłuższe niż 24 526 bajtów w języku C i 9 950 bajtów w języku COBOL są obcinane. Wynika to ze sposobu używania kolejek pamięci tymczasowej systemu CICS .
- Dla obu programów, CSQ4CCK1 i CSQ4CVK1, wpisz B w parametrze get, jeśli chcesz przeglądać komunikaty, w przeciwnym razie wpisz D. Powoduje to wykonanie destrukcyjnych wywołań MQGET. Jeśli zostanie wprowadzona dowolna inna wartość, zostanie wyświetlony komunikat o błędzie.
- W przypadku obu programów, CSQ4CCJ1 i CSQ4CVJ1, w parametrze syncpoint należy wprowadzić wartość S, aby pobrać komunikaty w punkcie synchronizacji. Jeśli w parametrze punktu synchronizacji zostanie wprowadzona wartość N, wywołania MQGET będą wykonywane poza punktem synchronizacji. Jeśli zostanie wprowadzona dowolna inna wartość, zostanie wyświetlony komunikat o błędzie.

## z/OS *Przykład przeglądania w systemie z/OS*

Przykład przeglądania jest aplikacją wsadową, która demonstruje sposób przeglądania komunikatów w kolejce przy użyciu wywołania MQGET.

Aplikacja przechodzi przez wszystkie komunikaty w kolejce, drukując pierwsze 80 bajtów każdego z nich. Tej aplikacji można użyć do wyświetlania komunikatów w kolejce bez ich zmiany.

Programy źródłowe i przykładowy kod JCL uruchamiania są dostarczane w językach COBOL, assembler, PL/I oraz C (patrz sekcja Tabela 173 na stronie 1203 ).

Aby uruchomić aplikację, zmodyfikuj i uruchom przykładowy skrypt JCL, zgodnie z opisem w sekcji “Przygotowywanie i uruchamianie przykładowych aplikacji dla środowiska wsadowego w systemie z/OS” na stronie 1201. Istnieje możliwość wyświetlania komunikatów w jednej z własnych kolejek przez określenie nazwy kolejki w uruchomionym JCL.

Po uruchomieniu aplikacji (jeśli w kolejce znajdują się komunikaty), zestaw danych wyjściowych wygląda następująco:

```
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 **********
```

Jeśli w kolejce nie ma żadnych komunikatów, zestaw danych zawiera tylko nagłówki i komunikat `Koniec raportu` . Jeśli wystąpi błąd w którymkolwiek z wywołań MQI, kody zakończenia i przyczyny są dodawane do zestawu danych wyjściowych.

z/OS *Projekt przykładu przeglądania w systemie z/OS*

Przykładowa aplikacja Przeglądaj korzysta z pojedynczego modułu programu; jeden jest dostępny w każdym z obsługiwanych języków programowania.

Przepływ przez logikę programu jest następujący:

1. Otwórz zestaw danych wydruku i wydrukuj wiersz tytułowy raportu. Sprawdź, czy nazwy menedżera kolejek i kolejki zostały przekazane z uruchomionego zadania JCL. Jeśli obie nazwy zostały przekazane, wydrukuj wiersze raportu, które zawierają nazwy. Jeśli nie, wydrukuj komunikat o błędzie, zamknij zestaw danych drukowania i zatrzymaj przetwarzanie.

   Sposób, w jaki program testuje parametry przekazywane z JCL zależy od języka, w którym program został napisany; więcej informacji na ten temat zawiera sekcja “Uwagi dotyczące projektu zależnego od języka w systemie z/OS” na stronie 1218.
2. Nawiąż połączenie z menedżerem kolejek przy użyciu wywołania MQCONN. Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny, zamknij zestaw danych wydruku i zatrzymaj przetwarzanie.
3. Otwórz kolejkę za pomocą wywołania MQOPEN z opcją MQOO_BROWSE. Na wejściu do tego wywołania program używa uchwytu połączenia zwróconego w kroku “2” na stronie 1217. W przypadku struktury deskryptora obiektu (MQOD) używane są wartości domyślne dla wszystkich pól z wyjątkiem nazwy kolejki (która została przekazana w kroku “1” na stronie 1217 ). Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny, zamknij zestaw danych wydruku i zatrzymaj przetwarzanie.
4. Przejrzyj pierwszy komunikat w kolejce przy użyciu wywołania MQGET. Na wejściu do tego wywołania program określa:
   - Uchwyty połączenia i kolejki z kroków “2” na stronie 1217 i “3” na stronie 1217
   - Struktura MQMD ze wszystkimi polami ustawionymi na wartości początkowe
   - Dwie opcje:
     - MQGMO_BROWSE_FIRST
     - MQGMO_ACCEPT_TRUNCATED_MSG
   - Bufor o wielkości 80 bajtów do przechowywania danych skopiowanych z komunikatu

   Opcja MQGMO_ACCEPT_TRUNCATED_MSG umożliwia zakończenie wywołania, nawet jeśli komunikat jest dłuższy niż 80-bajtowy bufor określony w wywołaniu. Jeśli komunikat jest dłuższy niż bufor, komunikat jest obcinany w celu dopasowania do buforu, a kody zakończenia i przyczyny są ustawione w taki sposób, aby były wyświetlane. Przykład został zaprojektowany w taki sposób, aby komunikaty były obcinane do 80 znaków w celu ułatwienia odczytu raportu. Wielkość buforu jest ustawiana za pomocą instrukcji `DEFINE` , więc można ją łatwo zmienić.
5. Wykonaj następującą pętlę, dopóki wywołanie MQGET nie zakończy się niepowodzeniem:
   a. Wydrukuj wiersz raportu zawierający:
      - Numer kolejny komunikatu (jest to liczba operacji przeglądania).
      - Rzeczywista długość komunikatu (nie skrócona). Ta wartość jest zwracana w polu `DataLength` wywołania MQGET.
      - Pierwsze 80 bajtów danych komunikatu.

   b. Resetowanie pól `MsqId` i `CorrelId` struktury MQMD do wartości NULL

   c. Przejrzyj następny komunikat przy użyciu wywołania MQGET z następującymi dwiema opcjami:

- MQGMO_BROWSE_NEXT
- MQGMO_ACCEPT_TRUNCATED_MSG

6. Jeśli wywołanie MQGET nie powiedzie się, sprawdź kod przyczyny, aby sprawdzić, czy wywołanie nie zakończyło się niepowodzeniem, ponieważ kursor przeglądania znajduje się na końcu kolejki. W takim przypadku wydrukuj komunikat `Koniec raportu` i przejdź do kroku “7” na stronie 1218 ; W przeciwnym razie wydrukuj kody zakończenia i przyczyny, zamknij zestaw danych drukowania i zatrzymaj przetwarzanie.
7. Zamknij kolejkę za pomocą wywołania MQCLOSE z uchwytem obiektu zwróconym w kroku “3” na stronie 1217.
8. Rozłącz się z menedżerem kolejek przy użyciu wywołania MQDISC z uchwytem połączenia zwróconym w kroku “2” na stronie 1217.
9. Zamknij zestaw danych wydruku i zatrzymaj przetwarzanie.

z/OS *Uwagi dotyczące projektu zależnego od języka w systemie z/OS*

Moduły źródłowe są udostępniane dla przykładu Przeglądaj w czterech językach programowania.

Istnieją dwie główne różnice między modułami źródłowymi:

- Podczas testowania parametrów przekazanych z uruchomionego kodu JCL moduły języka COBOL, PL/I i języka asemblera wyszukują znak przecinka (,). Jeśli zadanie JCL przekaże wartość `PARM=(,LOCALQ1)`, aplikacja podejmie próbę otwarcia kolejki LOCALQ1 w domyślnym menedżerze kolejek. Jeśli po przecinku (lub przecinku) nie ma nazwy, aplikacja zwraca błąd. Moduł C nie wyszukuje znaku przecinka. Jeśli kod JCL przekazuje pojedynczy parametr (na przykład `PARM=('LOCALQ1')`), moduł C używa go jako nazwy kolejki w domyślnym menedżerze kolejek.
- Aby moduł języka asemblera był prosty, podczas tworzenia raportu wydruku używany jest format daty *yy/ddd* (na przykład 05/116). Inne moduły używają daty kalendarza w formacie *mm/dd/rr* .

## z/OS *Przykład Print Message w systemie z/OS*

Przykład Print Message to aplikacja wsadowa, która demonstruje sposób usuwania wszystkich komunikatów z kolejki za pomocą wywołania MQGET.

Przykład Print Message używa trzech parametrów:

1. Nazwa menedżera kolejek
2. Nazwa kolejki źródłowej
3. Parametr opcjonalny dla właściwości

Dla każdego komunikatu drukowane są także pola deskryptora komunikatu, po których następują dane komunikatu. Program drukuje dane zarówno w postaci szesnastkowej, jak i jako znaki (jeśli można je drukować). Jeśli znak nie jest drukowalny, program zastępuje go znakiem kropki (.). Programu można użyć do diagnozowania problemów z aplikacją umieszczającą komunikaty w kolejce.

Dopuszczalne wartości parametru właściwości to:

*Tabela 185. Dopuszczalne wartości parametru właściwości*

| **Wartość** | **zachowanie;** |
|---|---|
| 0 | Zachowanie domyślne. Właściwości dostarczane do aplikacji zależą od atrybutu kolejki **`PropertyControl`** , z którego jest pobierany komunikat. |

| *Tabela 185. Dopuszczalne wartości parametru właściwości (kontynuacja)* | |
|---|---|
| **Wartość** | **zachowanie;** |
| 1 | Uchwyt komunikatu jest tworzony i używany z usługą MQGET. Właściwości komunikatu, z wyjątkiem tych, które są zawarte w deskryptorze komunikatu (lub rozszerzeniu), są wyświetlane w sposób podobny do deskryptora komunikatu. Na przykład:<br>`****Message properties****`<br>`  property name: property value`<br>Lub jeśli nie są dostępne żadne właściwości:<br>`****Message properties****`<br>`  None`<br>Wartości liczbowe są wyświetlane przy użyciu `printf`, wartości łańcuchowe są ujęte w pojedynczy cudzysłów, a łańcuchy bajtów są ujęte w znaki X i pojedynczy cudzysłów, tak jak w przypadku deskryptora komunikatu. |
| 2 | Określono parametr MQGMO_NO_PROPERTIES, więc zostaną zwrócone tylko właściwości deskryptora komunikatu. |
| 3 | Określono właściwość MQGMO_PROPERTIES_FORCE_MQRFH2 , więc wszystkie właściwości są zwracane w danych komunikatu. |
| 4 | Określono parametr MQGMO_PROPERTIES_COMPATIBILITY, dzięki czemu wszystkie właściwości mogą być zwracane w zależności od tego, czy właściwość IBM MQ jest dołączona, w przeciwnym razie właściwości są usuwane. |

Aplikację można zmienić w taki sposób, aby przeglądane były komunikaty, zamiast usuwać je z kolejki. W tym celu należy przeprowadzić kompilację z opcją -DBROWSE, aby zdefiniować makro BROWSE, zgodnie z opisem w sekcji “Projekt przykładu Print Message w systemie z/OS” na stronie 1220. Kod wykonywalny jest udostępniany w bibliotece SCSQLOAD. Moduł CSQ4BCG0 jest budowany przy użyciu opcji -DBROWSE; moduł CSQ4BCG1 niszcząco odczytuje kolejkę.

Aplikacja ma pojedynczy program źródłowy, który jest napisany w języku C. Dostarczany jest również przykładowy kod uruchomienia JCL (patrz sekcja Tabela 174 na stronie 1203 ).

Aby uruchomić aplikację, zmodyfikuj i uruchom przykładowy skrypt JCL, zgodnie z opisem w sekcji “Przygotowywanie i uruchamianie przykładowych aplikacji dla środowiska wsadowego w systemie z/OS” na stronie 1201. Po uruchomieniu aplikacji (jeśli w kolejce znajdują się komunikaty) zestaw danych wyjściowych wygląda następująco: Rysunek 139 na stronie 1220.

```
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'C3E2D840D4D8F1C540404040404040C1EA537F03167D88'
CorrelId : X'C3E2D840D4D8F1C540404040404040C1EA537F0317A928'
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'000000000000000000000000000000000000000000000000'
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
```

*Rysunek 139. Przykład raportu z przykładowej aplikacji Drukowanie komunikatu*

z/OS *Projekt przykładu Print Message w systemie z/OS*

Przykładowa aplikacja Print Message korzysta z pojedynczego programu napisanego w języku C.

Przepływ przez logikę programu jest następujący:

1. Sprawdź, czy nazwy menedżera kolejek i kolejki zostały przekazane z uruchomionego zadania JCL. Jeśli nie, wydrukuj komunikat o błędzie i zatrzymaj przetwarzanie.
2. Nawiąż połączenie z menedżerem kolejek przy użyciu wywołania MQCONN. Jeśli wywołanie nie powiodło się, wydrukuj kody zakończenia i przyczyny oraz zatrzymaj przetwarzanie. W przeciwnym razie wydrukuj nazwę menedżera kolejek.
3. Otwórz kolejkę za pomocą wywołania MQOPEN z opcją MQOO_INPUT_SHARED.

   **Uwaga:** Jeśli aplikacja ma przeglądać komunikaty zamiast usuwać je z kolejki, skompiluj przykład za pomocą parametru -DBROWSE lub dodaj parametr `#define BROWSE` na początku kodu źródłowego. W takim przypadku preprocesor makr dodaje wiersz w programie, który wybiera opcję MQOO_BROWSE w kompilacji.

Na wejściu do tego wywołania program używa uchwytu połączenia zwróconego w kroku “2” na stronie 1220. W przypadku struktury deskryptora obiektu (MQOD) używane są wartości domyślne dla wszystkich pól z wyjątkiem nazwy kolejki (która została przekazana w kroku “1” na stronie 1220 ). Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny oraz zatrzymaj przetwarzanie; w przeciwnym razie wydrukuj nazwę kolejki.

4. Jeśli do uzyskania właściwości komunikatu używany jest uchwyt komunikatu, należy użyć komendy MQCRTMH w celu utworzenia takiego uchwytu do użycia w kolejnych wywołaniach MQGET. Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny oraz zatrzymaj przetwarzanie.
5. Ustaw opcje pobierania komunikatu, aby odzwierciedlić działanie żądania dla wszystkich właściwości komunikatu.
6. Wykonaj następującą pętlę, dopóki wywołanie MQGET nie zakończy się niepowodzeniem:
   a. Zainicjuj bufor jako pusty, aby dane komunikatu nie zostały uszkodzone przez żadne dane już znajdujące się w buforze.
   b. Ustaw wartości null w polach `MsgId` i `CorrelId` struktury MQMD, aby wywołanie MQGET wybierające pierwszy komunikat z kolejki.
   c. Pobierz komunikat z kolejki za pomocą wywołania MQGET. Na wejściu do tego wywołania program określa:
      - Połączenie i uchwyty obiektów z kroków “2” na stronie 1220 i “3” na stronie 1220.
      - Struktura MQMD ze wszystkimi polami ustawionymi na wartości początkowe. (Wartości`MsgId` i `CorrelId` są resetowane do wartości NULL dla każdego wywołania MQGET).
      - Opcja MQGMO_NO_WAIT.

        **Uwaga:** Jeśli aplikacja ma przeglądać komunikaty zamiast usuwać je z kolejki, skompiluj przykład za pomocą parametru -DBROWSE lub dodaj parametr `#define BROWSE` na początku kodu źródłowego. W takim przypadku preprocesor makr dodaje do kompilacji wiersz w programie, który wybiera opcję MQGMO_BROWSE_NEXT. Jeśli ta opcja jest używana w wywołaniu dla kolejki, dla której nie użyto wcześniej kursora przeglądania z bieżącym uchwytem obiektu, kursor przeglądania jest pozycjonowany logicznie przed pierwszym komunikatem.
      - Bufor o wielkości 64KB do przechowywania danych skopiowanych z komunikatu.
   d. Wywołaj podprocedurę printMD . Powoduje to wydrukowanie nazwy każdego pola w deskryptorze komunikatu, po którym następuje jego zawartość.
   e. Jeśli w kroku “4” na stronie 1221 został utworzony uchwyt komunikatu, wywołaj podprocedurę printProperties , aby wyświetlić właściwości komunikatu.
   f. Wydrukuj długość komunikatu, po której następują dane komunikatu. Każdy wiersz danych komunikatu ma następujący format:
      - Względna pozycja (szesnastkowa) tej części danych
      - 16 bajtów danych szesnastkowych
      - Te same 16 bajtów danych w formacie znakowym, jeśli są drukowalne (znaki niedrukowalne są zastępowane przez kropki)
7. Jeśli wywołanie MQGET nie powiedzie się, przetestuj kod przyczyny, aby sprawdzić, czy wywołanie nie zakończyło się niepowodzeniem, ponieważ w kolejce nie ma więcej komunikatów. W takim przypadku należy wydrukować komunikat: Brak komunikatów; w przeciwnym razie należy wydrukować kody zakończenia i przyczyny. W obu przypadkach przejdź do kroku “9” na stronie 1222.

   **Uwaga:** Wywołanie MQGET nie powiedzie się, jeśli znajdzie komunikat zawierający więcej niż 64KB danych. Aby zmienić program tak, aby obsługiwał większe komunikaty, można wykonać jedną z następujących czynności:

   - Dodaj opcję MQGMO_ACCEPT_TRUNCATED_MSG do wywołania MQGET, aby wywołanie pobrały pierwsze 64KB danych i odrzuciły pozostałą część.
   - Powoduje, że program pozostawia komunikat w kolejce, gdy znajdzie komunikat z taką ilością danych

- Zwiększ wielkość buforu

8. Jeśli w kroku “4” na stronie 1221 został utworzony uchwyt komunikatu, wywołaj komendę MQDLTMH, aby go usunąć.
9. Zamknij kolejkę za pomocą wywołania MQCLOSE z uchwytem obiektu zwróconym w kroku “3” na stronie 1220.
10. Rozłącz się z menedżerem kolejek przy użyciu wywołania MQDISC z uchwytem połączenia zwróconym w kroku “2” na stronie 1220.

### z/OS *Przykład atrybutów kolejki w systemie z/OS*

Przykład Atrybuty kolejki jest aplikacją CICS w trybie konwersacyjnym, która demonstruje użycie wywołań MQINQ i MQSET.

Przedstawia on sposób uzyskiwania informacji o wartościach atrybutów **InhibitPut** i **InhibitGet** kolejek oraz sposób ich zmiany, tak aby programy nie mogły umieszczać komunikatów w kolejce ani ich z niej pobrać. W ten sposób można *zablokować* kolejkę podczas testowania programu.

Aby zapobiec przypadkowej ingerencji we własne kolejki, przykład ten działa tylko w przypadku obiektu kolejki, który zawiera znaki CSQ4SAMP w pierwszych ośmiu bajtach nazwy. Jednak kod źródłowy zawiera komentarze, które pokazują, w jaki sposób usunąć to ograniczenie.

Programy źródłowe są dostarczane w językach COBOL, assembler i C (patrz Tabela 180 na stronie 1208 ).

Wersja w języku asembler przykładu używa kodu, który można ponownie wbudować. W tym celu należy zauważyć, że kod dla każdego wywołania MQI w tej wersji przykładu zawiera słowo kluczowe MF, na przykład:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

(Słowo kluczowe VL oznacza, że do debugowania programu można użyć transakcji dostarczonej przez narzędzie CICS Execution Diagnostic Facility (CEDF).) Więcej informacji na temat pisania programów z możliwością ponownego wpisywania znajduje się w sekcji Coding in System/390 assembler language(Kodowanie w języku asembler systemu System/390).

Aby uruchomić aplikację, należy uruchomić system CICS i użyć następujących transakcji CICS :

- W przypadku języka COBOL: MVC1
- Dla języka asemblera, MAC1
- Dla języka C: MCC1

Nazwę każdej z tych transakcji można zmienić, zmieniając zestaw danych CSD wymieniony w kroku 3.

## Wzór próbki

Po uruchomieniu przykładu zostanie wyświetlona mapa ekranowa, która zawiera pola dla następujących elementów:

- Nazwa kolejki.
- Żądanie użytkownika (poprawne działania to: inquire, allow lub inhibit)
- Bieżący status operacji umieszczania dla kolejki
- Bieżący status operacji pobierania dla kolejki

Pierwsze dwa pola służą do wprowadzania danych przez użytkownika. Dwa ostatnie pola są wypełniane przez aplikację: zawierają słowo `INHIBITED` lub słowo `ALLOWED`.

Aplikacja sprawdza poprawność wartości wprowadzonych w pierwszych dwóch polach. Sprawdza, czy nazwa kolejki rozpoczyna się od znaków CSQ4SAMP i czy w polu Działanie wprowadzono jedno z trzech poprawnych żądań. Aplikacja przekształca wszystkie dane wejściowe na wielkie litery, dlatego nie można używać kolejek z nazwami zawierającymi małe litery.

Jeśli w polu **Działanie** zostanie wprowadzona wartość `inquire` , przepływ przez logikę programu będzie następujący:

1. Otwórz kolejkę za pomocą wywołania MQOPEN z opcją MQOO_INQUIRE
2. Wywołaj komendę MQINQ przy użyciu selektorów MQIA_INHIBIT_GET i MQIA_INHIBIT_PUT
3. Zamknij kolejkę za pomocą wywołania MQCLOSE
4. Przeanalizuj atrybuty zwracane w parametrze **`IntAttrs`** wywołania MQINQ i przenieś słowa INHIBITED lub ALLOWED do odpowiednich pól ekranu.

Jeśli w polu **Działanie** zostanie wprowadzona wartość `inhibit` , przepływ przez logikę programu będzie następujący:

1. Otwórz kolejkę za pomocą wywołania MQOPEN z opcją MQOO_SET
2. Wywołaj komendę MQSET przy użyciu selektorów MQIA_INHIBIT_GET i MQIA_INHIBIT_PUT z wartościami MQQA_GET_INHIBITED i MQQA_PUT_INHIBITED w parametrze **`IntAttrs`** .
3. Zamknij kolejkę za pomocą wywołania MQCLOSE
4. Przenieś słowo INHIBITED do odpowiednich pól ekranu

Jeśli w polu **Czynność** zostanie wprowadzona wartość `allow` , aplikacja wykona przetwarzanie podobne do przetwarzania dla żądania blokady. Jedyne różnice to ustawienia atrybutów i słów wyświetlanych na ekranie.

Gdy aplikacja otwiera kolejkę, używa domyślnego uchwytu połączenia z menedżerem kolejek. (Program CICS nawiązuje połączenie z menedżerem kolejek podczas uruchamiania systemu CICS ). Na tym etapie aplikacja może przechwytywać następujące błędy:

- Aplikacja nie jest połączona z menedżerem kolejek
- Kolejka nie istnieje
- Użytkownik nie ma uprawnień dostępu do kolejki
- Aplikacja nie ma uprawnień do otwarcia kolejki

W przypadku innych błędów MQI aplikacja wyświetla kody zakończenia i przyczyny.

## z/OS *Przykład menedżera poczty w systemie z/OS*

Przykładowa aplikacja Mail Manager to pakiet programów demonstrujących wysyłanie i odbieranie komunikatów, zarówno w pojedynczym środowisku, jak i w różnych środowiskach. Aplikacja jest prostym elektronicznym systemem poczty elektronicznej, który umożliwia użytkownikom wymianę komunikatów, nawet jeśli używają oni różnych menedżerów kolejek.

Aplikacja demonstruje sposób tworzenia kolejek za pomocą wywołania MQOPEN i umieszczania komend IBM MQ for z/OS w kolejce wejściowej komend systemowych.

Dostępne są trzy wersje aplikacji:

- Aplikacja CICS napisana w języku COBOL
- Aplikacja TSO napisana w języku COBOL
- Aplikacja TSO napisana w języku C

### z/OS *Przygotowywanie przykładu programu Mail Manager w systemie z/OS*

Menedżer poczty jest dostępny w wersjach, które działają w dwóch środowiskach. Przygotowania, które należy wykonać przed uruchomieniem aplikacji, zależą od środowiska, które ma być używane.

Użytkownicy mogą uzyskiwać dostęp do kolejek poczty i kolejek pseudonimów zarówno z TSO, jak i z CICS , pod warunkiem, że ich identyfikatory są takie same w każdym systemie.

Przed wysłaniem komunikatów do innego menedżera kolejek należy skonfigurować kanał komunikatów do tego menedżera kolejek. W tym celu należy użyć funkcji sterowania kanałami systemu IBM MQopisanej w sekcji Funkcja sterowania kanałami.

## Przygotowywanie przykładu dla środowiska TSO

Wykonaj następujące kroki:

1. Przygotuj przykład zgodnie z opisem w sekcji “Przygotowywanie przykładowych aplikacji dla środowiska TSO w systemie z/OS” na stronie 1204.
2. Należy dostosować CLIST dostarczoną dla próbki w celu zdefiniowania:
   - Położenie paneli
   - Położenie pliku komunikatów
   - Położenie modułów ładowania
   - Nazwa menedżera kolejek, który ma być używany z aplikacją

   Dla każdej wersji językowej przykładu udostępniono osobną CLIST:
   - W przypadku wersji COBOL: CSQ4RVD1
   - Dla wersji C: CSQ4RCD1
3. Upewnij się, że kolejki używane przez aplikację są dostępne w menedżerze kolejek. (Kolejki są zdefiniowane w CSQ4CVD).

**Uwaga:** VS COBOL II nie obsługuje wielozadaniowości z interfejsem ISPF. Oznacza to, że nie można używać przykładowej aplikacji Mail Manager po obu stronach podzielonego ekranu. W przeciwnym razie wyniki będą nieprzewidywalne.

z/OS *Uruchamianie przykładu menedżera poczty elektronicznej w systemie z/OS*

Aby uruchomić przykład w środowisku produktu CICS Transaction Server for z/OS , uruchom transakcję MAIL. Jeśli użytkownik nie zalogował się jeszcze do programu CICS, w aplikacji zostanie wyświetlona prośba o wprowadzenie identyfikatora użytkownika, do którego będzie mógł wysyłać pocztę.

Po uruchomieniu aplikacji otwierana jest kolejka poczty. Jeśli ta kolejka nie istnieje, aplikacja utworzy ją dla użytkownika. Kolejki poczty elektronicznej mają nazwy w postaci CSQ4SAMP.MAILMGR. *userid*, gdzie *userid* zależy od środowiska:

**W TSO**
Identyfikator TSO użytkownika

**WCICS**
Logowanie użytkownika w systemie CICS lub identyfikator użytkownika wprowadzony przez użytkownika po wyświetleniu monitu o uruchomienie menedżera poczty elektronicznej

Wszystkie części nazw kolejek używane przez menedżera poczty muszą być zapisane wielkimi literami.

Aplikacja wyświetli następnie panel menu z opcjami dla:

- Odczytaj pocztę przychodzącą
- Wysyłanie wiadomości e-mail
- CREATE NICKNAME

Na panelu menu wyświetlana jest również liczba wiadomości oczekujących w kolejce poczty. Każda z opcji menu powoduje wyświetlenie kolejnego panelu:

**Odczytaj pocztę przychodzącą**
Menedżer poczty wyświetla listę wiadomości znajdujących się w kolejce poczty. (wyświetlanych jest tylko 99 pierwszych komunikatów w kolejce). Przykład tego panelu zawiera sekcja Rysunek 142 na stronie 1228. Po wybraniu komunikatu z listy zostanie wyświetlona jego treść (patrz sekcja Rysunek 143 na stronie 1229 ).

**Wysyłanie wiadomości e-mail**
Zostanie wyświetlony panel z prośbą o wprowadzenie następujących informacji:

- Nazwa użytkownika, do którego ma zostać wysłany komunikat
- Nazwa menedżera kolejek, do którego należy jego kolejka poczty

- Tekst wiadomości

W polu nazwy użytkownika można wprowadzić identyfikator użytkownika lub pseudonim utworzony za pomocą programu Mail Manager. Pole nazwy menedżera kolejek można pozostawić puste, jeśli właścicielem kolejki poczty elektronicznej użytkownika jest ten sam menedżer kolejek, który jest używany, i pozostawić to pole puste, jeśli w polu nazwy użytkownika wprowadzono pseudonim:

- Jeśli zostanie podana tylko nazwa użytkownika, program najpierw przyjmuje, że nazwa jest pseudonimem, i wysyła komunikat do obiektu zdefiniowanego przez tę nazwę. Jeśli taki pseudonim nie istnieje, program próbuje wysłać komunikat do kolejki lokalnej o tej nazwie.
- Jeśli zostanie podana zarówno nazwa użytkownika, jak i nazwa menedżera kolejek, program wysyła komunikat do kolejki poczty, która jest zdefiniowana przez te dwie nazwy.

Na przykład, aby wysłać komunikat do użytkownika JONESM w zdalnym menedżerze kolejek QM12, można wysłać komunikat na jeden z dwóch sposobów:

- Użyj obu pól, aby określić użytkownika JONESM w menedżerze kolejek QM12.
- Zdefiniuj pseudonim (na przykład MARY) dla tego użytkownika i wyślij mu komunikat, umieszczając MARY w polu nazwy użytkownika i nic w polu nazwy menedżera kolejek.

**CREATE NICKNAME**

Można zdefiniować łatwą do zapamiętania nazwę, której można użyć podczas wysyłania wiadomości do innego użytkownika, z którym często się kontaktujesz. Zostanie wyświetlona prośba o wprowadzenie identyfikatora innego użytkownika oraz nazwy menedżera kolejek, do którego należy jego kolejka poczty elektronicznej.

Pseudonimy to kolejki o nazwach w postaci CSQ4SAMP.MAILMGR. *userid.nickname*, gdzie *id_użytkownika* to własny identyfikator użytkownika, a *pseudonim* to pseudonim, którego chcesz użyć. Z nazwami ustrukturyzowanymi w ten sposób użytkownicy mogą mieć własny zestaw pseudonimów.

Typ kolejki, którą tworzy program, zależy od sposobu wypełnienia pól na panelu Tworzenie pseudonimu:

- Jeśli zostanie podana tylko nazwa użytkownika lub nazwa menedżera kolejek jest taka sama jak nazwa menedżera kolejek, z którym jest połączony menedżer poczty, program utworzy kolejkę aliasową.
- Jeśli zostanie podana zarówno nazwa użytkownika, jak i nazwa menedżera kolejek (a menedżer kolejek nie jest tym, z którym połączony jest menedżer poczty elektronicznej), program utworzy lokalną definicję kolejki zdalnej. Program nie sprawdza, czy istnieje kolejka, na którą rozstrzygana jest ta definicja, a nawet czy istnieje zdalny menedżer kolejek.

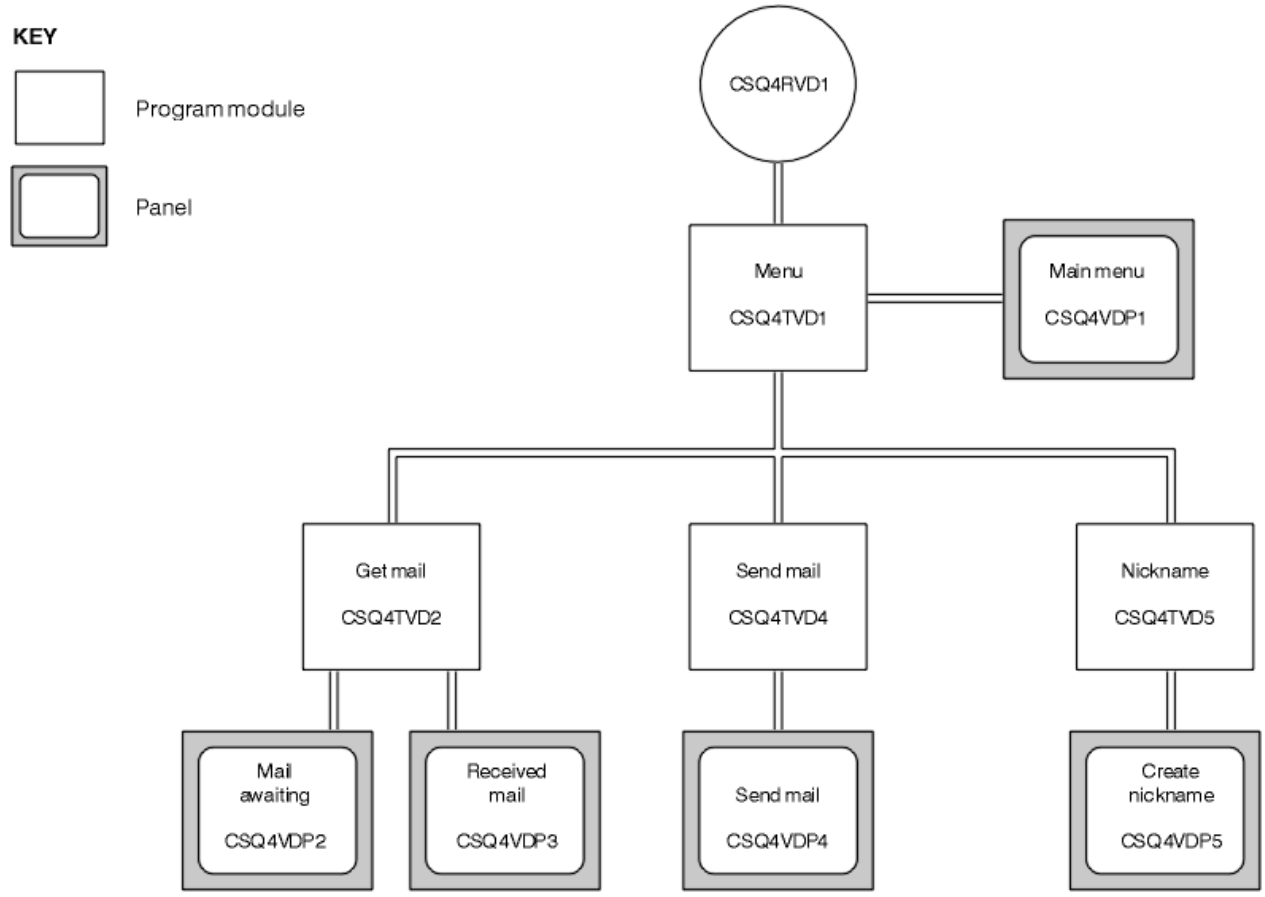
Na przykład, jeśli identyfikatorem użytkownika jest SMITHK, a użytkownik utworzy pseudonim MARY dla użytkownika JONESM (który używa zdalnego menedżera kolejek QM12), program pseudonimu utworzy lokalną definicję kolejki zdalnej o nazwie CSQ4SAMP.MAILMGR.SMITHK.MARY. Ta definicja jest tłumaczona na kolejkę poczty Mary, którą jest CSQ4SAMP.MAILMGR.JONESM w menedżerze kolejek QM12. Jeśli menedżer kolejek QM12 jest używany samodzielnie, program utworzy kolejkę aliasową o takiej samej nazwie (CSQ4SAMP.MAILMGR.SMITHK.MARY).

Wersja C aplikacji TSO korzysta z możliwości obsługi komunikatów narzędzia ISPFw większym stopniu niż wersja COBOL. Można zauważyć, że różne komunikaty o błędach są wyświetlane w wersjach C i COBOL.

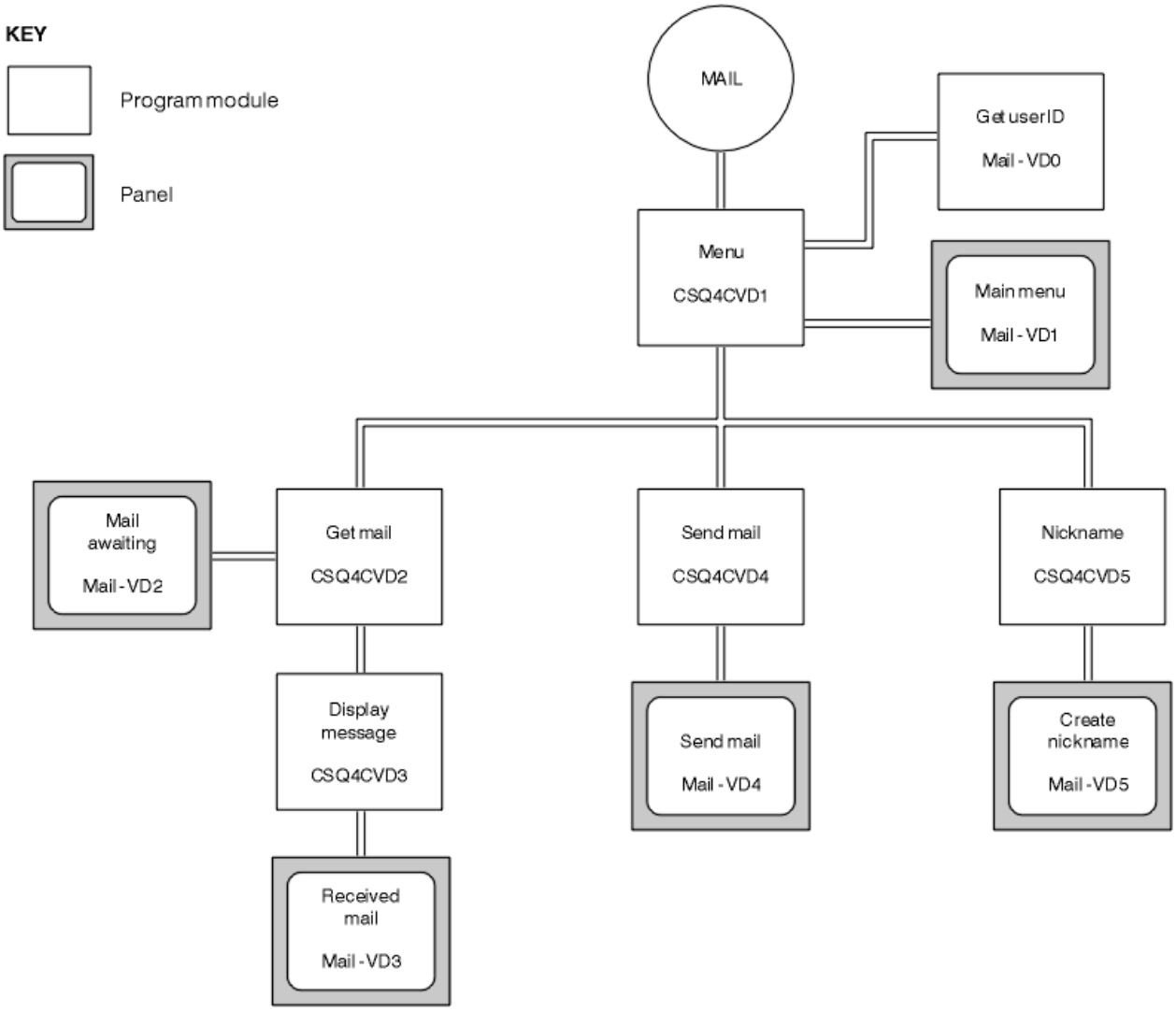
z/OS *Projekt przykładu programu Mail Manager w systemie z/OS*

W poniższych sekcjach opisano wszystkie programy, które tworzą przykładową aplikację Mail Manager.

Relacje między programami i panelami, z których korzysta aplikacja, są przedstawione w sekcji Rysunek 140 na stronie 1226 dla wersji TSO i w sekcji Rysunek 141 na stronie 1227 dla wersji produktu CICS Transaction Server for z/OS .




*Rysunek 140. Programy i panele dla wersji TSO programu Mail Manager*




*Rysunek 141. Programy i panele dla wersji CICS programu Mail Manager*

z/OS *Program menu w systemie z/OS*

W środowisku TSO program menu jest wywoływany przez CLIST. W środowisku CICS program jest wywoływany przez transakcję MAIL.

Program menu (CSQ4TVD1 dla TSO, CSQ4CVD1 dla CICS) jest programem początkowym w pakiecie. Wyświetla menu (CSQ4VDP1 dla TSO, VD1 dla CICS) i wywołuje inne programy po ich wybraniu z menu.

Program najpierw uzyskuje identyfikator użytkownika:

- W wersji programu dla systemu CICS , jeśli użytkownik wpisał się do systemu CICS, identyfikator użytkownika jest uzyskiwany za pomocą komendy CICS ASSIGN USERID. Jeśli użytkownik nie zalogował się, program wyświetli panel wpisywania się (CSQ4VD0), aby poprosić użytkownika o wprowadzenie identyfikatora użytkownika. W tym programie nie ma przetwarzania ochrony; użytkownik może podać dowolny ID użytkownika.
- W wersji TSO identyfikator użytkownika jest uzyskiwany z TSO w CLIST. Jest on przekazywany do programu menu jako zmienna w puli współużytkowanej ISPF .

Po uzyskaniu identyfikatora użytkownika program sprawdza, czy użytkownik ma kolejkę poczty (CSQ4SAMP.MAILMGR. *id_użytkownika* ). Jeśli kolejka poczty nie istnieje, program tworzy ją, umieszczając komunikat w kolejce wejściowej komend systemowych. Komunikat zawiera komendę IBM MQ for z/OS DEFINE QLOCAL. Definicja obiektu używana przez tę komendę ustawia maksymalną głębokość kolejki na 9999 komunikatów.

Program tworzy również tymczasową kolejkę dynamiczną do obsługi odpowiedzi z kolejki wejściowej komend systemowych. W tym celu program korzysta z wywołania MQOPEN z parametrem SYSTEM.DEFAULT.MODEL.QUEUE jako szablon dla kolejki dynamicznej. Menedżer kolejek tworzy tymczasową kolejkę dynamiczną o nazwie z przedrostkiem CSQ4SAMP; pozostała część nazwy jest generowana przez menedżer kolejek.

Następnie program otwiera kolejkę poczty użytkownika i wyszukuje liczbę komunikatów w kolejce, pytając o bieżące zapełnienie kolejki. W tym celu program używa wywołania MQINQ, określając selektor MQIA_CURRENT_Q_DEPTH.

Następnie program wykonuje pętlę, która wyświetla menu i przetwarza wybór dokonany przez użytkownika. Pętla jest zatrzymywana, gdy użytkownik naciśnie klawisz PF3 . Po dokonaniu poprawnego wyboru uruchamiany jest odpowiedni program; w przeciwnym razie wyświetlany jest komunikat o błędzie.

z/OS *Pobieranie poczty i wyświetlanie wiadomości w systemie z/OS*

W wersjach TSO aplikacji funkcje get-mail i display-message są wykonywane przez ten sam program (CSQ4TVD2). W wersji CICS aplikacji te funkcje są wykonywane przez osobne programy (CSQ4CVD2 i CSQ4CVD3).

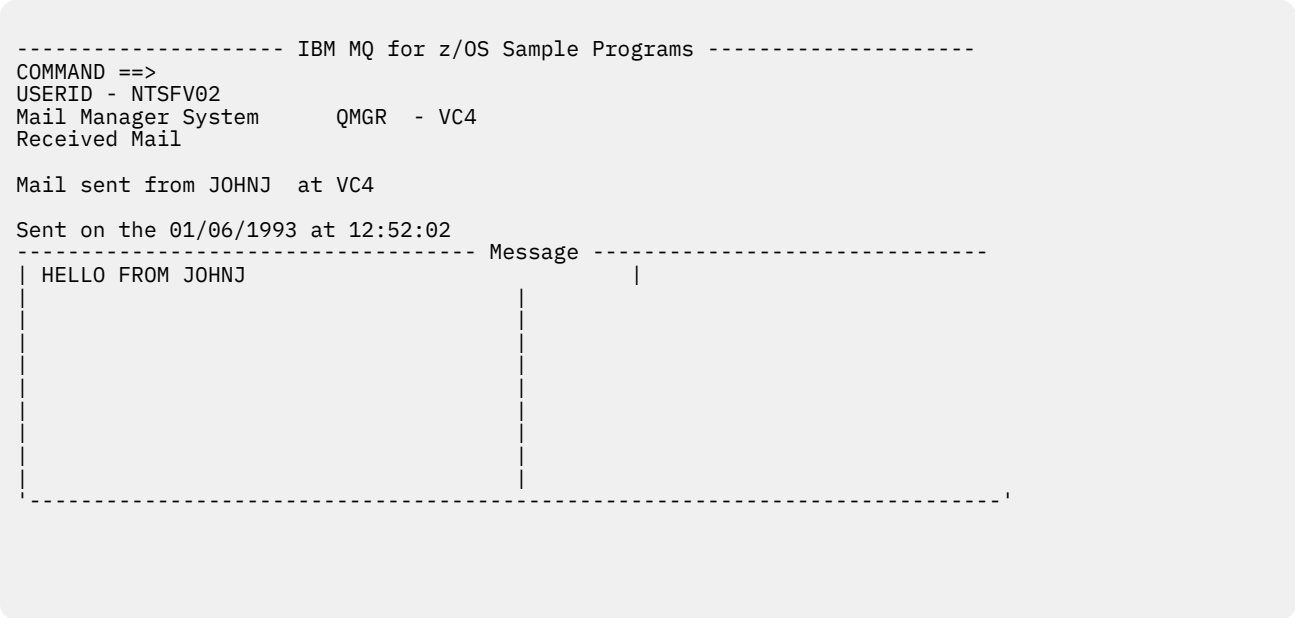
Panel Poczta oczekująca (CSQ4VDP2 dla TSO, VD2 dla CICS ; Rysunek 142 na stronie 1228 przedstawia przykład wszystkich wiadomości znajdujących się w kolejce poczty użytkownika. Aby utworzyć tę listę, program używa wywołania MQGET do przeglądania wszystkich komunikatów w kolejce, zapisując informacje o każdym z nich. Oprócz wyświetlanych informacji program zapisuje `MsgId` i `CorrelId` każdego komunikatu.

```
---------------------- 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   JOHNJ   01/06/1993 12:52:02
18   JOHNJ   01/06/1993 12:52:02
19   JOHNJ   01/06/1993 12:52:03
20   JOHNJ   01/06/1993 12:52:03
21   JOHNJ   01/06/1993 12:52:03
22   JOHNJ   01/06/1993 12:52:04
23   JOHNJ   01/06/1993 12:52:04
24   JOHNJ   01/06/1993 12:52:04
25   JOHNJ   01/06/1993 12:52:05
26   JOHNJ   01/06/1993 12:52:05
27   JOHNJ   01/06/1993 12:52:05
28   JOHNJ   01/06/1993 12:52:06
29   JOHNJ   01/06/1993 12:52:06
```

*Rysunek 142. Przykład panelu wyświetlającego listę oczekujących komunikatów*

Na panelu Poczta oczekująca użytkownik może wybrać jeden komunikat i wyświetlić jego treść (przykład można znaleźć w sekcji Rysunek 143 na stronie 1229 ). Program korzysta z wywołania MQGET w celu usunięcia tego komunikatu z kolejki przy użyciu komend `MsgId` i `CorrelId` , które program odnotował podczas przeglądania wszystkich komunikatów. To wywołanie MQGET jest wykonywane przy użyciu opcji MQGMO_SYNCPOINT. Program wyświetla treść komunikatu, a następnie deklaruje punkt synchronizacji: powoduje to zatwierdzenie wywołania MQGET, więc komunikat już nie istnieje.




*Rysunek 143. Przykład panelu przedstawiającego treść komunikatu*

Oczywistym rozszerzeniem funkcji udostępnianej przez menedżera poczty jest możliwość pozostawienia komunikatu w kolejce po wyświetleniu jego zawartości. W tym celu po wyświetleniu komunikatu należy wycofać wywołanie MQGET, które usuwa komunikat z kolejki.

z/OS *Program do wysyłania poczty elektronicznej w systemie z/OS*

Po zakończeniu panelu Wyślij pocztę (CSQ4VDP4 dla TSO, VD4 dla CICS ), program do wysyłania poczty (CSQ4TVD4 dla TSO, CSQ4CVD4 dla CICS ) umieszcza komunikat w kolejce poczty odbiorcy.

W tym celu program używa wywołania MQPUT1 . Miejsce docelowe komunikatu zależy od tego, w jaki sposób użytkownik wypełnił pola w panelu Wyślij wiadomość e-mail:

- Jeśli użytkownik określił tylko nazwę użytkownika, program najpierw przyjmuje, że nazwa jest pseudonimem, i wysyła komunikat do obiektu zdefiniowanego przez tę nazwę. Jeśli taki pseudonim nie istnieje, program próbuje wysłać komunikat do kolejki lokalnej o tej nazwie.
- Jeśli użytkownik określił zarówno nazwę użytkownika, jak i nazwę menedżera kolejek, program wysyła komunikat do kolejki poczty, która jest zdefiniowana przez te dwie nazwy.

Program nie akceptuje pustych komunikatów i usuwa odstępy wiodące z każdego wiersza tekstu komunikatu.

Jeśli wywołanie MQPUT1 zakończy się pomyślnie, program wyświetli komunikat z nazwą użytkownika i nazwą menedżera kolejek, w którym został umieszczony komunikat. Jeśli wywołanie nie powiedzie się, program sprawdza kody przyczyny wskazujące, że kolejka lub menedżer kolejek nie istnieje. Są to MQRC_UNKNOWN_OBJECT_NAME i MQRC_UNKNOWN_OBJECT_Q_MGR. Program wyświetla własny komunikat o błędzie dla każdego z tych błędów; w przypadku innych błędów program wyświetla kody zakończenia i przyczyny zwracane przez wywołanie.

z/OS *Program z pseudonimem w systemie z/OS*

Gdy użytkownik definiuje pseudonim, program pseudonimu (CSQ4TVD5 dla TSO, CSQ4CVD5 dla CICS ) tworzy kolejkę, której pseudonim jest częścią nazwy.

W tym celu program umieszcza komunikat w kolejce wejściowej komend systemowych. Komunikat zawiera komendę IBM MQ for z/OS DEFINE QALIAS lub DEFINE QREMOTE. Typ kolejki, którą tworzy program, zależy od tego, w jaki sposób użytkownik wypełnił pola panelu Tworzenie pseudonimu (CSQ4VDP5 dla TSO, VD5 dla CICS ):

- Jeśli użytkownik określił tylko nazwę użytkownika lub nazwa menedżera kolejek jest taka sama jak nazwa menedżera kolejek, z którym połączony jest menedżer poczty, program tworzy kolejkę aliasową.

- Jeśli użytkownik określił zarówno nazwę użytkownika, jak i nazwę menedżera kolejek (a menedżer kolejek nie jest tym, z którym połączony jest menedżer poczty elektronicznej), program tworzy lokalną definicję kolejki zdalnej. Program nie sprawdza, czy istnieje kolejka, na którą rozstrzygana jest ta definicja, a nawet czy istnieje zdalny menedżer kolejek.

Program tworzy również tymczasową kolejkę dynamiczną do obsługi odpowiedzi z kolejki wejściowej komend systemowych.

Jeśli menedżer kolejek nie może utworzyć kolejki pseudonimów z powodu oczekiwanego przez program (na przykład kolejka już istnieje), program wyświetla własny komunikat o błędzie. Jeśli menedżer kolejek nie może utworzyć kolejki z powodu, którego program nie oczekuje, program wyświetla maksymalnie dwa komunikaty o błędach, które są zwracane do programu przez serwer komend.

**Uwaga:** Dla każdego pseudonimu program tworzy tylko kolejkę aliasową lub lokalną definicję kolejki zdalnej. Kolejki lokalne, na które są tłumaczone te nazwy kolejek, są tworzone tylko wtedy, gdy do uruchomienia aplikacji Mail Manager używany jest identyfikator użytkownika, który jest zawarty w pseudonimie.

### z/OS *Przykład sprawdzania zdolności kredytowej w systemie z/OS*

Przykładowa aplikacja Credit Check jest pakietem programów demonstrujących sposób korzystania z wielu funkcji udostępnianych przez produkt IBM MQ for z/OS. Pokazuje on, w jaki sposób wiele programów składowych aplikacji może przekazywać komunikaty do siebie nawzajem przy użyciu technik kolejkowania komunikatów.

Przykład można uruchomić jako autonomiczną aplikację CICS . Jednak aby zademonstrować sposób projektowania aplikacji kolejkowania komunikatów korzystającej z narzędzi udostępnianych przez środowiska CICS i IMS , jeden moduł jest również dostarczany jako program przetwarzania komunikatów wsadowych systemu IMS . To rozszerzenie przykładu zostało opisane w sekcji “Rozszerzenie IMS do przykładu Sprawdzenie zdolności kredytowej w serwisie z/OS” na stronie 1241.

Przykład można również uruchomić w więcej niż jednym menedżerze kolejek i wysyłać komunikaty między poszczególnymi instancjami aplikacji. W tym celu należy zapoznać się z sekcją “Przykład sprawdzania kredytu z wieloma menedżerami kolejek w systemie z/OS” na stronie 1241.

Programy CICS są dostarczane w języku C i COBOL. Pojedynczy program IMS jest dostarczany tylko w języku C. Dostarczone zestawy danych są przedstawione w sekcji Tabela 182 na stronie 1209 i Tabela 184 na stronie 1211.

Wniosek przedstawia metodę oceny ryzyka, gdy klienci banku proszą o udzielenie pożyczki. W aplikacji przedstawiono, w jaki sposób bank może pracować na dwa sposoby w celu przetwarzania wniosków o pożyczkę:

- W przypadku bezpośredniego kontaktu z klientem pracownicy banku chcą mieć natychmiastowy dostęp do informacji o kontach i ryzyku kredytodawczym.
- Przy rozpatrywaniu pisemnych wniosków, pracownicy banku mogą złożyć serię wniosków o informacje dotyczące rachunku i ryzyka kredytowego, a następnie zająć się odpowiedziami w późniejszym terminie.

Szczegóły finansowe i dotyczące bezpieczeństwa w aplikacji zostały uproszczone, aby techniki kolejkowania komunikatów były jasne.

#### z/OS *Przygotowywanie i uruchamianie przykładu sprawdzania kredytu w systemie z/OS*

Aby przygotować i uruchomić przykład sprawdzania zdolności kredytowej, wykonaj następujące kroki:

1. Utwórz zestaw danych VSAM, który zawiera informacje o niektórych przykładowych kontach. W tym celu należy zmodyfikować i uruchomić zadanie JCL dostarczone w zestawie danych CSQ4FILE.
2. Wykonaj kroki opisane w temacie “Przygotowywanie przykładowych aplikacji dla środowiska CICS w systemie z/OS” na stronie 1206. (Dodatkowe kroki, które należy wykonać, aby użyć rozszerzenia IMS przykładu, zostały opisane w sekcji “Rozszerzenie IMS do przykładu Sprawdzenie zdolności kredytowej w serwisie z/OS” na stronie 1241).

3. Uruchom monitor wyzwalacza CKTI (dostarczany z programem IBM MQ for z/OS ) dla kolejki CSQ4SAMP.INITIATION.QUEUE, za pomocą transakcji CICS CKQC.
4. Aby uruchomić aplikację, należy uruchomić system CICS i użyć transakcji MVB1.
5. Na pierwszym panelu wybierz zapytanie **Natychmiastowe** lub **wsadowe** .

   Panele zapytań natychmiastowych i wsadowych są podobne; Rysunek 144 na stronie 1231 przedstawia panel zapytań natychmiastowych.

```
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
```

   *Rysunek 144. Panel natychmiastowego zapytania dla przykładowej aplikacji sprawdzania zdolności kredytowej*
6. W odpowiednich polach wprowadź numer konta i kwotę pożyczki. Sekcja “Wprowadzanie informacji na panelach zapytań” na stronie 1231 zawiera wskazówki dotyczące informacji, które należy wprowadzić w tych polach.

## Wprowadzanie informacji na panelach zapytań

Przykładowa aplikacja Credit Check sprawdza, czy dane wprowadzone w polu **Żądana kwota** na panelach zapytania mają postać liczb całkowitych.

Jeśli zostanie wprowadzony jeden z następujących numerów konta, aplikacja znajdzie odpowiednią nazwę konta, średnie saldo konta i indeks wiarygodności kredytowej w zestawie danych VSAM CSQ4BAQ: :

- 2222222222
- 3111234329
- 3256478962
- 3333333333
- 3501676212
- 3696879656
- 4444444444
- 5555555555
- 6666666666
- 7777777777

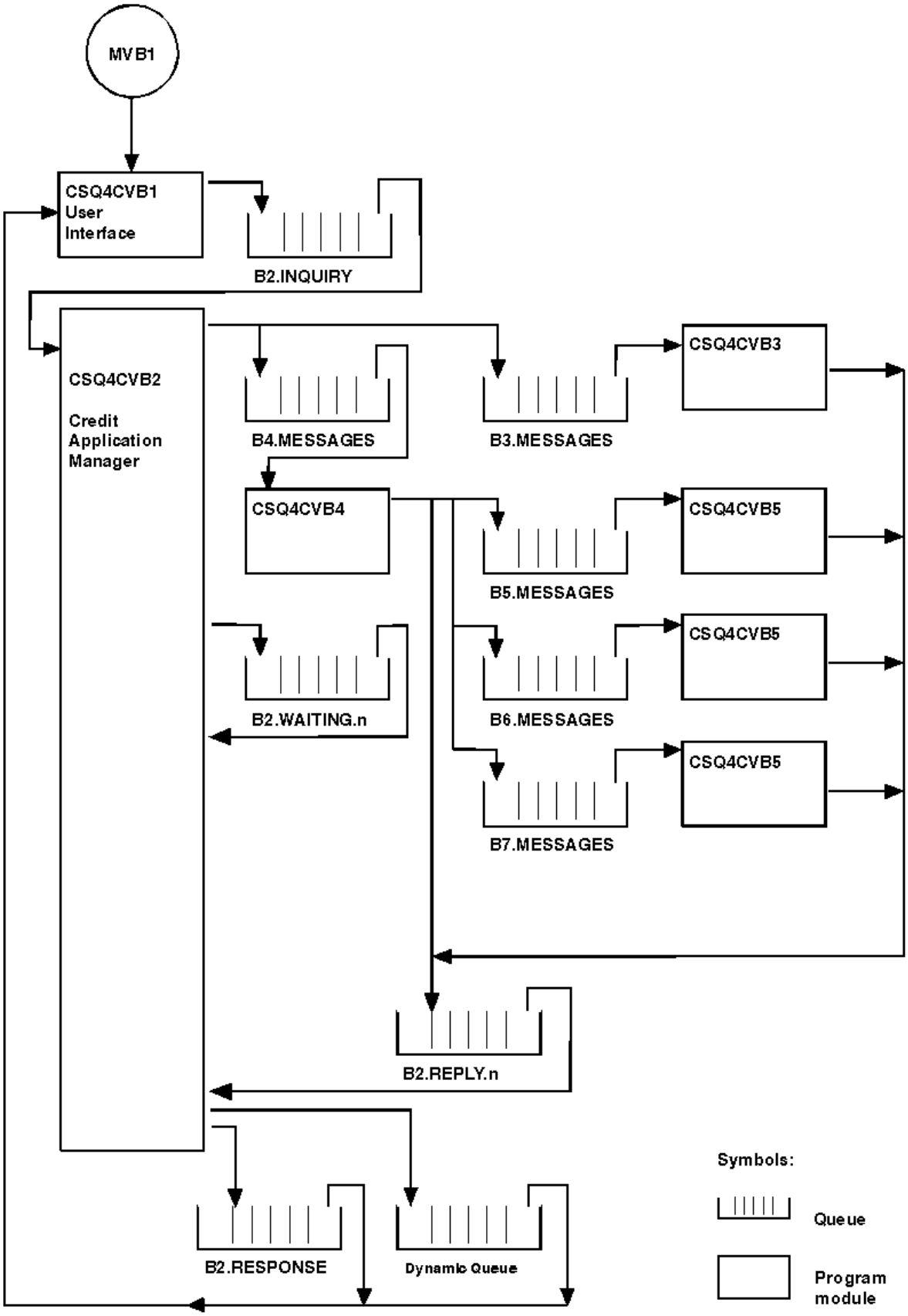
W innych polach można wprowadzać dowolne informacje lub nie wprowadzać żadnych informacji. Aplikacja zachowuje wszelkie wprowadzone informacje i zwraca te same informacje w generowanych przez nią raportach.

z/OS *Projekt przykładu Credit Check w serwisie z/OS*

W tej sekcji opisano projekt każdego z programów, które tworzą przykładową aplikację sprawdzania zdolności kredytowej.

Więcej informacji na temat niektórych technik, które były brane pod uwagę podczas projektowania aplikacji, zawiera sekcja “Uwagi dotyczące projektowania przykładu sprawdzania zdolności kredytowej w serwisie z/OS” na stronie 1238.

Rysunek 145 na stronie 1233 przedstawia programy, które tworzą aplikację, a także kolejki, które te programy obsługują. Na tym rysunku przedrostek CSQ4SAMP został pominięty we wszystkich nazwach kolejek, aby ułatwić zrozumienie tego rysunku.




*Rysunek 145. Programy i kolejki dla aplikacji przykładowej Credit Check (tylko programy w języku COBOL)*

z/OS *Program interfejsu użytkownika (CSQ4CVB1) w systemie z/OS*

Uruchomienie CICS transakcji MVB1w trybie konwersacyjnym powoduje uruchomienie programu interfejsu użytkownika dla aplikacji.

Program ten umieszcza komunikaty z zapytaniem w kolejce CSQ4SAMP.B2.INQUIRY i pobiera odpowiedzi na te zapytania z kolejki odpowiedzi, którą określa podczas wykonywania zapytania. Z poziomu interfejsu użytkownika można wprowadzać zapytania natychmiastowe lub wsadowe:

- W przypadku zapytań natychmiastowych program tworzy tymczasową kolejkę dynamiczną, która jest używana jako kolejka odpowiedzi. Oznacza to, że każde zapytanie ma własną kolejkę odpowiedzi.
- W przypadku zapytań wsadowych program interfejsu użytkownika otrzymuje odpowiedzi z kolejki CSQ4SAMP.B2.RESPONSE. Dla uproszczenia program otrzymuje odpowiedzi na wszystkie zapytania z tej jednej kolejki odpowiedzi. Łatwo jest zauważyć, że bank może chcieć użyć oddzielnej kolejki odpowiedzi dla każdego użytkownika MVB1, aby każdy z nich mógł zobaczyć odpowiedzi tylko na te zapytania, które zainicjował.

Ważne różnice między właściwościami komunikatów używanych w aplikacji w trybie wsadowym i natychmiastowym są następujące:

- W przypadku przetwarzania wsadowego komunikaty mają niski priorytet, dlatego są przetwarzane po każdym wniosku o pożyczkę, który został wprowadzony w trybie natychmiastowym. Ponadto komunikaty są trwałe, dlatego są odtwarzane, jeśli aplikacja lub menedżer kolejek musi zostać zrestartowany.
- W przypadku natychmiastowego działania komunikaty mają wysoki priorytet, dlatego są przetwarzane przed wszystkimi żądaniami pożyczki, które są wprowadzane w trybie wsadowym. Ponadto komunikaty nie są trwałe, dlatego są usuwane, jeśli aplikacja lub menedżer kolejek musi zostać zrestartowany.

Jednak we wszystkich przypadkach właściwości komunikatów żądania pożyczki są propagowane w całej aplikacji. Dlatego na przykład wszystkie komunikaty będące wynikiem żądania o wysokim priorytecie również będą miały wysoki priorytet.

z/OS *Menedżer aplikacji kredytowych (CSQ4CVB2) w systemie z/OS*

Program Credit Application Manager (CAM) wykonuje większość przetwarzania dla aplikacji Credit Check.

CAM jest uruchamiany przez monitor wyzwalacza CKTI (dostarczany z produktem IBM MQ for z/OS), gdy zdarzenie wyzwalające wystąpi w dowolnej kolejce CSQ4SAMP.B2.INQUIRY lub queue CSQ4SAMP.B2.REPLY. *n*, gdzie *n* jest liczbą całkowitą identyfikującą jedną z kolejek odpowiedzi. Komunikat wyzwalacza zawiera dane zawierające nazwę kolejki, w której wystąpiło zdarzenie wyzwalacza.

CAM używa kolejek o nazwach w postaci CSQ4SAMP.B2.WAITING.n -służy do przechowywania informacji o przetwarzanych zapytaniach. Nazwy kolejek są tak nazwane, aby każda z nich była połączona z kolejką odpowiedzi, na przykład kolejka CSQ4SAMP.B2.WAITING.3 zawiera dane wejściowe dla konkretnego zapytania oraz kolejkę CSQ4SAMP.B2.REPLY.3 zawiera zestaw komunikatów odpowiedzi (pochodzących z programów, które wysyłają zapytania do baz danych) odnoszących się do tego samego zapytania. Aby zrozumieć przyczyny takiego projektu, należy zapoznać się z sekcją "Oddzielne kolejki zapytań i odpowiedzi w CAM" na stronie 1239.

## Logika uruchamiania

Jeśli zdarzenie wyzwalające występuje w kolejce CSQ4SAMP.B2.INQUIRY, CAM otwiera kolejkę dla dostępu współużytkowanego. Następnie próbuje otworzyć każdą kolejkę odpowiedzi, aż do znalezienia kolejki wolnej. Jeśli nie może znaleźć wolnej kolejki odpowiedzi, CAM rejestruje fakt i kończy działanie normalnie.

Jeśli zdarzenie wyzwalające występuje w kolejce CSQ4SAMP.B2.REPLY.n, CAM otwiera kolejkę na wyłączny dostęp. Jeśli kod powrotu zgłasza, że obiekt jest już używany, CAM kończy działanie normalnie. Jeśli wystąpi inny błąd, CAM zarejestruje błąd i zakończy działanie. CAM otwiera odpowiednią kolejkę oczekiwania i kolejkę zapytania, a następnie rozpoczyna pobieranie i przetwarzanie komunikatów. Z kolejki oczekiwania CAM odzyskuje szczegóły częściowo zakończonych zapytań.

Dla uproszczenia w tym przykładzie nazwy używanych kolejek są przechowywane w programie. W środowisku biznesowym nazwy kolejek będą prawdopodobnie przechowywane w pliku, do którego program uzyskuje dostęp.

## Pobieranie komunikatu z kolejki zapytań

CAM najpierw próbuje pobrać komunikat z kolejki zapytań za pomocą wywołania MQGET z opcją MQGMO_SET_SIGNAL. Jeśli komunikat jest dostępny natychmiast, jest on przetwarzany; jeśli żaden komunikat nie jest dostępny, ustawiany jest sygnał.

Następnie CAM próbuje pobrać komunikat z kolejki odpowiedzi, ponownie używając wywołania MQGET z tą samą opcją. Jeśli komunikat jest dostępny natychmiast, jest on przetwarzany; w przeciwnym razie ustawiany jest sygnał.

Gdy oba sygnały są ustawione, program oczekuje na przesłanie jednego z sygnałów. Jeśli zostanie wysłany sygnał wskazujący, że komunikat jest dostępny, komunikat jest pobierany i przetwarzany. Jeśli sygnał utraci ważność lub menedżer kolejek zostanie zakończony, program zakończy działanie.

## Przetwarzanie komunikatu pobranego przez CAM

Komunikat pobrany przez CAM może być jednego z czterech typów:

- Komunikat z zapytaniem
- Komunikat odpowiedzi
- Komunikat propagacji
- Nieoczekiwany lub niepożądany komunikat

CAM przetwarza te komunikaty w sposób opisany w sekcji “Przetwarzanie komunikatu pobranego przez CAM w systemie z/OS” na stronie 1236.

## Wysyłanie odpowiedzi

Gdy CAM otrzyma wszystkie odpowiedzi, których oczekuje na zapytanie, przetwarza odpowiedzi i tworzy pojedynczy komunikat odpowiedzi. Konsoliduje on w jeden komunikat wszystkie dane ze wszystkich komunikatów odpowiedzi, które mają ten sam `CorrelId`. Ta odpowiedź jest umieszczana w kolejce odpowiedzi określonej w pierwotnym wniosku o pożyczkę. Komunikat odpowiedzi jest umieszczany w tej samej jednostce pracy, która zawiera pobranie końcowego komunikatu odpowiedzi. Ma to na celu uproszczenie odzyskiwania poprzez zapewnienie, że w kolejce CSQ4SAMP.B2.WAITING.n.

## Odzyskiwanie częściowo zakończonych dochodzeń

CAM kopiuje do kolejki CSQ4SAMP.B2.WAITING.n -wszystkie odebrane komunikaty. Ustawia pola deskryptora komunikatu w następujący sposób:

- Wartość *`Priority`* jest określana na podstawie typu komunikatu:
  - Dla komunikatów żądań, priorytet = 3
  - Dla datagramów, priorytet = 2
  - Dla komunikatów odpowiedzi, priorytet = 1
- Parametr *`CorrelId`* jest ustawiony na wartość *`MsgId`* komunikatu żądania pożyczki.
- Inne pola MQMD są kopiowane z pól odebranego komunikatu.

Po zakończeniu zapytania komunikaty dla konkretnego zapytania są usuwane z kolejki oczekiwania podczas przetwarzania odpowiedzi. Dlatego w dowolnym momencie kolejka oczekująca zawiera wszystkie komunikaty istotne dla trwających zapytań. Te komunikaty są używane do odzyskiwania szczegółów zapytań w toku, jeśli program musi zostać zrestartowany. Różne priorytety są ustawiane w taki sposób, aby komunikaty z zapytaniem były odtwarzane przed propagacją lub odpowiedzią.

z/OS *Przetwarzanie komunikatu pobranego przez CAM w systemie z/OS*

Komunikat pobrany przez menedżera aplikacji kredytowej (CAM) może być jednego z czterech typów. Sposób przetwarzania komunikatu przez CAM zależy od jego typu.

Komunikat pobrany przez CAM może być jednego z czterech typów:

- Komunikat z zapytaniem
- Komunikat odpowiedzi
- Komunikat propagacji
- Nieoczekiwany lub niepożądany komunikat

CAM przetwarza te komunikaty w następujący sposób:

**Komunikat z zapytaniem**

Komunikaty z zapytaniem pochodzą z programu interfejsu użytkownika. Dla każdego żądania pożyczki tworzony jest komunikat z zapytaniem.

W przypadku wszystkich wniosków o pożyczkę CAM żąda średniego salda konta rozliczeniowego klienta. Jest to realizowane przez umieszczenie komunikatu żądania w kolejce aliasowej CSQ4SAMP.B2.OUTPUT.ALIAS. Ta nazwa kolejki jest tłumaczona na kolejkę CSQ4SAMP.B3.MESSAGES, które są przetwarzane przez program konta rozliczeniowego CSQ4CVB3. Gdy CAM umieszcza komunikat w tej kolejce aliasowej, określa odpowiedni CSQ4SAMP.B2.REPLY.n dla kolejki odpowiedzi. W tym miejscu używana jest kolejka aliasowa, dzięki czemu program CSQ4CVB3 można łatwo zastąpić innym programem, który przetwarza kolejkę podstawową o innej nazwie. W tym celu należy ponownie zdefiniować kolejkę aliasową, tak aby jej nazwa była tłumaczona na nową kolejkę. Można również przypisać różne uprawnienia dostępu do kolejki aliasowej i do kolejki podstawowej.

Jeśli użytkownik żąda pożyczki większej niż 10000 jednostek, CAM inicjuje również sprawdzanie innych baz danych. Jest to realizowane przez umieszczenie komunikatu żądania w kolejce CSQ4SAMP.B4.MESSAGES, które są przetwarzane przez program dystrybucyjny CSQ4CVB4. Proces obsługujący tę kolejkę propaguje komunikat do kolejek obsługiwanych przez programy, które mają dostęp do innych rekordów, takich jak historia kart kredytowych, konta oszczędnościowe i płatności hipoteczne. Dane z tych programów są zwracane do kolejki odpowiedzi określonej w operacji umieszczania (put). Dodatkowo komunikat propagacji jest wysyłany przez ten program do kolejki odpowiedzi w celu określenia liczby wysłanych komunikatów propagacji.

W środowisku biznesowym program dystrybucyjny prawdopodobnie ponownie sformatowałby podane dane tak, aby były zgodne z formatem wymaganym przez każdy inny typ rachunku bankowego.

Dowolna z wymienionych kolejek może znajdować się w systemie zdalnym.

Dla każdego komunikatu z zapytaniem CAM inicjuje wpis w tabeli rekordów zapytania (IRT). Ten rekord zawiera:

- `MsgId` komunikatu z zapytaniem
- W polu ReplyExp : oczekiwana liczba odpowiedzi (równa liczbie wysłanych komunikatów).
- W polu ReplyRec : liczba odebranych odpowiedzi (zero na tym etapie).
- W polu PropsOut wskazuje, czy komunikat propagacji jest oczekiwany.

CAM kopiuje komunikat z zapytaniem do kolejki oczekiwania z następującymi danymi:

- `Priority` ustaw na 3
- Parametr `CorrelId` ustawiony na wartość `MsgId` komunikatu z zapytaniem.
- Pozostałe pola deskryptora komunikatu ustawione na pola komunikatu z zapytaniem

**Komunikat propagacji**

Komunikat propagacji zawiera liczbę kolejek, do których program dystrybucyjny przekazał zapytanie. Komunikat jest przetwarzany w następujący sposób:

1. Dodaj do pola ReplyExp odpowiedniego rekordu w produkcie IRT liczbę wysłanych komunikatów. Te informacje znajdują się w komunikacie.
2. Zwiększ o 1 pole ReplyRec rekordu w koszu IRT.
3. Zmniejsz o 1 pole PropsOut rekordu w koszu IRT.
4. Skopiuj komunikat do kolejki oczekujących. CAM ustawia `Priority` na 2, a pozostałe pola deskryptora komunikatu na pola komunikatu propagacji.

**Komunikat odpowiedzi**

Komunikat odpowiedzi zawiera odpowiedź na jedno z żądań skierowanych do programu obsługi kont kontrolnych lub do jednego z programów zapytań do agencji. Komunikaty odpowiedzi są przetwarzane w następujący sposób:

1. Zwiększ o 1 pole ReplyRec rekordu w koszu IRT.
2. Skopiuj komunikat do kolejki oczekującej z wartością `Priority` ustawioną na 1 i innymi polami deskryptora komunikatu ustawionymi na wartości z komunikatu odpowiedzi.
3. Jeśli ReplyRec = ReplyExpi PropsOut = 0, ustaw opcję MsgComplete .

**Mogą zostać wyświetlone inne komunikaty**

Aplikacja nie oczekuje innych komunikatów. Jednak aplikacja może odbierać komunikaty rozgłaszane przez system lub odpowiadać na komunikaty z nieznanym `CorrelIds`.

CAM umieszcza te komunikaty w kolejce CSQ4SAMP.DEAD.QUEUE, gdzie można je sprawdzić. Jeśli operacja umieszczania nie powiedzie się, komunikat zostanie utracony i program będzie kontynuował działanie. Więcej informacji na temat projektowania tej części programu zawiera sekcja "Sposób obsługi nieoczekiwanych komunikatów przez przykład" na stronie 1239.

z/OS *Program konta rozliczeniowego (CSQ4CVB3) w systemie z/OS*

Program konta rozliczeniowego jest uruchamiany przez zdarzenie wyzwalające w kolejce CSQ4SAMP.B3.MESSAGES. Po otwarciu kolejki program ten pobiera komunikat z kolejki za pomocą wywołania MQGET z opcją wait i z interwałem oczekiwania ustawionym na 30 sekund.

Program wyszukuje numer konta w komunikacie żądania pożyczki w zestawie danych VSAM CSQ4BAQ . Pobiera odpowiednią nazwę konta, średnie saldo i indeks wiarygodności kredytowej lub zauważa, że numer konta nie znajduje się w zestawie danych.

Następnie program umieszcza komunikat odpowiedzi (za pomocą wywołania MQPUT1 ) w kolejce odpowiedzi określonej w komunikacie żądania pożyczki. Dla tego komunikatu odpowiedzi program:

- Kopiuje `CorrelId` komunikatu żądania pożyczki
- Używa opcji MQPMO_PASS_IDENTITY_CONTEXT

Program kontynuuje pobieranie komunikatów z kolejki, dopóki nie upłynie odstęp czasu oczekiwania.

z/OS *Program dystrybucyjny (CSQ4CVB4) w systemie z/OS*

Program dystrybucyjny jest uruchamiany przez zdarzenie wyzwalane w kolejce CSQ4SAMP.B4.MESSAGES.

W celu symulowania dystrybucji żądania pożyczki do innych agencji, które mają dostęp do rekordów, takich jak historia kart kredytowych, konta oszczędnościowe i płatności hipoteczne, program umieszcza kopię tego samego komunikatu we wszystkich kolejkach na liście nazw CSQ4SAMP.B4.NAMELIST. Istnieją trzy z tych kolejek o nazwach w postaci CSQ4SAMP.B *n*.MESSAGES, gdzie *n* to 5, 6 lub 7. W aplikacji biznesowej agencje mogą znajdować się w różnych miejscach, dlatego te kolejki mogą być kolejkami zdalnymi. Aby zmodyfikować przykładową aplikację w celu wyświetlenia tej informacji, należy zapoznać się z sekcją "Przykład sprawdzania kredytu z wieloma menedżerami kolejek w systemie z/OS" na stronie 1241.

Program dystrybucyjny wykonuje następujące kroki:

1. Z listy nazw pobiera nazwy kolejek, które mają być używane przez program. Program wykonuje to za pomocą wywołania MQINQ w celu uzyskania informacji o atrybutach obiektu listy nazw.
2. Otwiera te kolejki, a także CSQ4SAMP.B4.MESSAGES.

3. Wykonuje następującą pętlę do momentu, gdy w kolejce CSQ4SAMP.B4.MESSAGES:
   a. Pobierz komunikat przy użyciu wywołania MQGET z opcją oczekiwania i z interwałem oczekiwania ustawionym na 30 sekund.
   b. Umieść komunikat w każdej kolejce wymienionej na liście nazw, podając nazwę odpowiedniego CSQ4SAMP.B2.REPLY.n dla kolejki odpowiedzi. Program kopiuje plik *CorrelId* komunikatu żądania pożyczki do tych komunikatów kopiowania i używa opcji MQPMO_PASS_IDENTITY_CONTEXT w wywołaniu MQPUT.
   c. Wyślij komunikat datagramu do kolejki CSQ4SAMP.B2.REPLY.n , aby wyświetlić liczbę pomyślnie umieszczonych komunikatów.
   d. Zadeklaruj punkt synchronizacji.

z/OS *Agency-query program (CSQ4CVB5/CSQ4CCB5) na platformie z/OS*

Program agency-query jest dostarczany zarówno jako program w języku COBOL, jak i jako program w języku C. Oba programy mają ten sam projekt. Oznacza to, że programy różnych typów mogą łatwo współistnieć w aplikacji IBM MQ i że moduły programów składające się na taką aplikację mogą być łatwo zastąpione.

Instancja programu jest uruchamiana przez zdarzenie wyzwalające w dowolnej z następujących kolejek:

- Dla programu w języku COBOL (CSQ4CVB5):
  - CSQ4SAMP.B5.MESSAGES
  - CSQ4SAMP.B6.MESSAGES
  - CSQ4SAMP.B7.MESSAGES
- Dla programu w języku C (CSQ4CCB5), kolejka CSQ4SAMP.B8.MESSAGES

**Uwaga:** Aby używać programu w języku C, należy zmienić definicję listy nazw CSQ4SAMP.B4.NAMELIST zastępuje kolejkę CSQ4SAMP.B7.MESSAGES z CSQ4SAMP.B8.MESSAGES. W tym celu można użyć jednej z następujących opcji:

- Operacje IBM MQ for z/OS i panele sterowania
- Komenda ALTER NAMELIST
- Program narzędziowy CSQUTIL

Po otwarciu odpowiedniej kolejki program pobiera komunikat z kolejki za pomocą wywołania MQGET z opcją oczekiwania i z interwałem oczekiwania ustawionym na 30 sekund.

Program symuluje wyszukiwanie w bazie danych agencji, wyszukując w zestawie danych VSAM CSQ4BAQ numer konta, który został przekazany w komunikacie żądania pożyczki. Następnie tworzy odpowiedź zawierającą nazwę kolejki, którą obsługuje, oraz indeks wiarygodności kredytowej. Aby uprościć przetwarzanie, indeks wiarygodności kredytowej jest wybierany losowo.

Podczas umieszczania komunikatu odpowiedzi program używa wywołania MQPUT1 i:

- Kopiuje `CorrelId` komunikatu żądania pożyczki
- Używa opcji MQPMO_PASS_IDENTITY_CONTEXT

Program wysyła komunikat odpowiedzi do kolejki odpowiedzi określonej w komunikacie żądania pożyczki. (Nazwa menedżera kolejek, który jest właścicielem kolejki odpowiedzi, jest również określona w komunikacie żądania pożyczki).

z/OS *Uwagi dotyczące projektowania przykładu sprawdzania zdolności kredytowej w serwisie z/OS*

Rozważania projektowe dotyczące przykładu sprawdzania zdolności kredytowej.

Ta sekcja zawiera informacje na temat:

- “Oddzielne kolejki zapytań i odpowiedzi w CAM” na stronie 1239
- “Sposób obsługi błędów przez przykład” na stronie 1239
- “Sposób obsługi nieoczekiwanych komunikatów przez przykład” na stronie 1239

- “Sposób użycia punktów synchronizacji w przykładzie” na stronie 1240
- “W jaki sposób przykład używa informacji o kontekście komunikatu” na stronie 1240
- “Stosowanie identyfikatorów komunikatów i korelacji w CAM” na stronie 1241

## Oddzielne kolejki zapytań i odpowiedzi w CAM

Aplikacja mogła używać pojedynczej kolejki zarówno dla zapytań, jak i dla odpowiedzi, ale została zaprojektowana do używania oddzielnych kolejek z następujących powodów:

- Jeśli program obsługuje maksymalną liczbę zapytań, dalsze zapytania mogą pozostać w kolejce. Jeśli używana jest pojedyncza kolejka, musi ona zostać pobrana z kolejki i zapisana w innym miejscu.
- Inne instancje CAM mogą być uruchamiane automatycznie w celu obsługi tej samej kolejki zapytań, jeśli ruch komunikatów był wystarczająco duży, aby to zagwarantować. Ale program musi śledzić w toku zapytania, a w tym celu musi odzyskać wszystkie odpowiedzi na zapytania, które zainicjował. Jeśli używana jest tylko jedna kolejka, program będzie musiał przejrzeć komunikaty, aby sprawdzić, czy są one przeznaczone dla tego lub innego programu. W ten sposób operacja byłaby znacznie mniej wydajna.

  Aplikacja może obsługiwać wiele serwerów CAMs i skutecznie odtwarzać zapytania w toku, korzystając z kolejek odpowiedzi i oczekujących.
- Program może oczekiwać w wielu kolejkach efektywnie za pomocą sygnalizacji.

## Sposób obsługi błędów przez przykład

Program interfejsu użytkownika obsługuje błędy, zgłaszając je bezpośrednio do użytkownika.

Inne programy nie mają interfejsów użytkownika, dlatego muszą obsługiwać błędy w inny sposób. Ponadto w wielu sytuacjach (na przykład w przypadku niepowodzenia wywołania MQGET) te inne programy nie znają tożsamości użytkownika aplikacji.

Inne programy umieszczają komunikaty o błędach w kolejce pamięci tymczasowej CICS o nazwie CSQ4SAMP. Kolejkę tę można przeglądać za pomocą transakcji CEBR dostarczonej przez CICS. Programy zapisują również komunikaty o błędach w dzienniku CSML CICS .

## Sposób obsługi nieoczekiwanych komunikatów przez przykład

Podczas projektowania aplikacji kolejkowania komunikatów należy zdecydować, w jaki sposób obsługiwać komunikaty, które nieoczekiwanie docierają do kolejki.

Dostępne są dwie podstawowe opcje:

- Aplikacja nie będzie już działać, dopóki nie przetworze nieoczekiwanego komunikatu. Prawdopodobnie oznacza to, że aplikacja powiadamia operatora, kończy działanie i zapewnia, że nie zostanie automatycznie zrestartowana (może to zrobić poprzez wyłączenie wyzwalania). Ten wybór oznacza, że całe przetwarzanie aplikacji może zostać zatrzymane przez jeden nieoczekiwany komunikat, a interwencja operatora jest wymagana do zrestartowania aplikacji.
- Aplikacja usuwa komunikat z kolejki, którą obsługuje, umieszcza komunikat w innym miejscu i kontynuuje przetwarzanie. Najlepszym miejscem na umieszczenie tego komunikatu jest systemowa kolejka niedostarczonych komunikatów.

W przypadku wybrania drugiej opcji:

- Operator lub inny program powinien sprawdzić komunikaty umieszczone w kolejce niedostarczonych komunikatów, aby dowiedzieć się, skąd pochodzą te komunikaty.
- Jeśli nie można umieścić nieoczekiwanego komunikatu w kolejce niedostarczonych komunikatów, zostanie on utracony.
- Długi nieoczekiwany komunikat jest obcinany, jeśli jest dłuższy niż limit dla komunikatów w kolejce niedostarczonych komunikatów lub dłuższy niż wielkość buforu w programie.

W celu zapewnienia, że aplikacja będzie bezproblemowo obsługiwać wszystkie zapytania z minimalnym wpływem działań zewnętrznych, w przykładowej aplikacji Credit Check używana jest druga opcja. Aby

umożliwić oddzielenie przykładu od innych aplikacji używających tego samego menedżera kolejek, w przykładzie sprawdzania zdolności kredytowej nie jest używana systemowa kolejka niedostarczonych komunikatów. Zamiast tego używana jest własna kolejka niedostarczonych komunikatów. Ta kolejka ma nazwę CSQ4SAMP.DEAD.QUEUE. Przykład obcina wszystkie komunikaty, które są dłuższe niż obszar buforu udostępniony dla programów przykładowych. Można użyć przykładowej aplikacji Przeglądaj, aby przeglądać komunikaty w tej kolejce, lub aplikacji przykładowej Drukuj komunikat, aby drukować komunikaty razem z ich deskryptorami komunikatów.

Jeśli jednak przykład zostanie rozszerzony w celu uruchomienia na więcej niż jednym menedżerze kolejek, menedżer kolejek może umieścić w systemowej kolejce niedostarczonych komunikatów nieoczekiwane komunikaty lub komunikaty, których nie można dostarczyć.

## Sposób użycia punktów synchronizacji w przykładzie

Programy w przykładowej aplikacji sprawdzania kredytu deklarują punkty synchronizacji, aby upewnić się, że:

- Tylko jeden komunikat odpowiedzi jest wysyłany w odpowiedzi na każdy oczekiwany komunikat
- Wiele kopii nieoczekiwanych komunikatów nigdy nie jest umieszczanych w kolejce niedostarczonych komunikatów przykładu.
- CAM może odzyskać stan wszystkich częściowo zakończonych zapytań, pobierając trwałe komunikaty z kolejki oczekiwania

Aby to osiągnąć, pojedyncza jednostka pracy jest używana do pokrycia pobierania komunikatu, przetwarzania tego komunikatu i wszystkich kolejnych operacji umieszczania.

## W jaki sposób przykład używa informacji o kontekście komunikatu

Gdy program interfejsu użytkownika (CSQ4CVB1) wysyła komunikaty, używa opcji MQPMO_DEFAULT_CONTEXT. Oznacza to, że menedżer kolejek generuje zarówno informacje o tożsamości, jak i o kontekście źródłowym. Menedżer kolejek pobiera te informacje z transakcji, która uruchomiła program (MVB1), oraz z identyfikatora użytkownika, który uruchomił transakcję.

Gdy CAM wysyła komunikaty z zapytaniem, używa opcji MQPMO_PASS_IDENTITY_CONTEXT. Oznacza to, że informacje o kontekście tożsamości umieszczanego komunikatu są kopiowane z kontekstu tożsamości oryginalnego komunikatu z zapytaniem. W przypadku użycia tej opcji informacje o kontekście źródłowym są generowane przez menedżer kolejek.

Gdy CAM wysyła komunikaty odpowiedzi, używa opcji MQPMO_ALTERNATE_USER_AUTHORITY. Powoduje to, że menedżer kolejek używa alternatywnego identyfikatora użytkownika na potrzeby sprawdzania zabezpieczeń, gdy CAM otwiera kolejkę odpowiedzi. CAM korzysta z identyfikatora użytkownika przesyłającego oryginalnego komunikatu z zapytaniem. Oznacza to, że użytkownicy mogą wyświetlać odpowiedzi tylko na te zapytania, z których pochodzą. Alternatywny identyfikator użytkownika jest uzyskiwany z informacji o kontekście tożsamości w deskryptorze oryginalnego komunikatu z zapytaniem.

Gdy programy zapytań (CSQ4CVB3/4/5) wysyłają komunikaty odpowiedzi, używają opcji MQPMO_PASS_IDENTITY_CONTEXT. Oznacza to, że informacje o kontekście tożsamości umieszczanego komunikatu są kopiowane z kontekstu tożsamości oryginalnego komunikatu z zapytaniem. W przypadku użycia tej opcji informacje o kontekście źródłowym są generowane przez menedżer kolejek.

**Uwaga:** Identyfikator użytkownika powiązany z transakcjami MVB3/4/5 wymaga dostępu do programu B2.REPLY.n kolejek. Te identyfikatory użytkowników mogą być różne od tych, które są powiązane z przetwarzanym żądaniem. Aby obejść ten problem, programy zapytań mogą używać opcji MQPMO_ALTERNATE_USER_AUTHORITY podczas umieszczania odpowiedzi. Oznacza to, że każdy użytkownik MVB1 musi mieć uprawnienia do otwierania programu B2.REPLY.n kolejek.

## Stosowanie identyfikatorów komunikatów i korelacji w CAM

Aplikacja musi monitorować postęp wszystkich bieżących zapytań, które przetwarza w dowolnym momencie. W tym celu używa unikalnego identyfikatora każdego komunikatu żądania pożyczki, aby powiązać wszystkie informacje dotyczące każdego zapytania.

CAM kopiuje `MsgId` komunikatu z zapytaniem do pliku `CorrelId` wszystkich komunikatów z żądaniem, które wysyła dla tego zapytania. Inne programy w przykładzie (CSQ4CVB3 -5) kopiują `CorrelId` każdego otrzymanego komunikatu do pliku `CorrelId` komunikatu odpowiedzi.

z/OS *Przykład sprawdzania kredytu z wieloma menedżerami kolejek w systemie z/OS*

Za pomocą przykładowej aplikacji Sprawdzanie kredytu można zademonstrować rozproszone kolejkowanie, instalując przykład w dwóch menedżerach kolejek i systemach CICS (z każdym menedżerem kolejek połączonym z innym systemem CICS ).

Po zainstalowaniu programu przykładowego i uruchomieniu monitora wyzwalacza (CKTI) w każdym systemie, należy wykonać następujące czynności:

1. Skonfiguruj łącze komunikacyjne między dwoma menedżerami kolejek. Więcej informacji na ten temat zawiera sekcja Konfigurowanie rozproszonego kolejkowania.
2. W jednym menedżerze kolejek utwórz lokalną definicję dla każdej kolejki zdalnej (w innym menedżerze kolejek), która ma być używana. Mogą to być dowolne z następujących kolejek: CSQ4SAMP.B *n*.MESSAGES, gdzie *n* to 3, 5, 6 lub 7. (Są to kolejki obsługiwane przez program checking-account i program agency-query). Informacje na ten temat można znaleźć w sekcji DEFINE QREMOTE oraz w sekcji DEFINE queues.
3. Zmień definicję listy nazw (CSQ4SAMP.B4.NAMELIST), aby zawierał nazwy kolejek zdalnych, które mają być używane. Informacje na ten temat zawiera sekcja DEFINE NAMELIST.

z/OS *Rozszerzenie IMS do przykładu Sprawdzenie zdolności kredytowej w serwisie z/OS*

Wersja programu do obsługi kont kontrolnych jest dostarczana jako program BMP ( IMS Batch Message Processing). Jest on napisany w języku C.

Program wykonuje tę samą funkcję, co wersja CICS , z tym wyjątkiem, że w celu uzyskania informacji o koncie program odczytuje bazę danych IMS zamiast pliku VSAM. Jeśli wersja CICS programu do obsługi kont kontrolnych zostanie zastąpiona wersją IMS , metoda korzystania z aplikacji nie będzie różniła się.

Aby przygotować i uruchomić wersję IMS , należy wykonać następujące czynności:

1. Wykonaj czynności opisane w sekcji “Przygotowywanie i uruchamianie przykładu sprawdzania kredytu w systemie z/OS” na stronie 1230.
2. Wykonaj czynności opisane w sekcji “Przygotowywanie przykładowej aplikacji dla środowiska IMS w systemie z/OS” na stronie 1210.
3. Zmień definicję kolejki aliasowej CSQ4SAMP.B2.OUTPUT.ALIAS , aby przetłumaczyć na kolejkę CSQ4SAMP.B3.IMS.MESSAGES (zamiast CSQ4SAMP.B3.MESSAGES). W tym celu można użyć jednej z następujących opcji:
   - Operacje IBM MQ for z/OS i panele sterowania
   - Komenda ALTER QALIAS .

Innym sposobem korzystania z programu IMS do obsługi kont kontrolnych jest obsługa jednej z kolejek, które odbierają komunikaty z programu dystrybucyjnego. W dostarczonej formie przykładowej aplikacji Credit Check istnieją trzy z tych kolejek (B5/6/7.MESSAGES), wszystkie obsługiwane przez program agency-query. Ten program przeszukuje zestaw danych VSAM. Aby porównać użycie zestawu danych VSAM i bazy danych IMS , można zamiast tego udostępnić jedną z tych kolejek przez program do obsługi kont kontrolnych IMS . W tym celu należy zmienić definicję listy nazw CSQ4SAMP.B4.NAMELIST zastępuje jeden z CSQ4SAMP.B *n*.Kolejki MESSAGES z CSQ4SAMP.B3.IMS.Kolejka komunikatów. Można użyć jednej z następujących opcji:

- Operacje IBM MQ for z/OS i panele sterowania
- Komenda ALTER NAMELIST .

Następnie można uruchomić przykład z poziomu CICS transakcji MVB1. Użytkownik nie widzi różnicy w działaniu lub odpowiedzi. Komponent BMP IMS zatrzymuje się po otrzymaniu komunikatu o zatrzymaniu lub po pięciu minutach braku aktywności.

## Projekt programu konta czekowego IMS (CSQ4ICB3)

Ten program działa jako BMP. Uruchom program przy użyciu jego kodu JCL przed wysłaniem do niego komunikatów IBM MQ .

Program wyszukuje w bazie danych IMS numer konta w komunikatach żądań pożyczki. Pobiera odpowiednią nazwę konta, średnie saldo i indeks wiarygodności kredytowej.

Program wysyła wyniki wyszukiwania w bazie danych do kolejki odpowiedzi o nazwie określonej w przetwarzanym komunikacie IBM MQ . Zwrócony komunikat dołącza typ konta i wyniki wyszukiwania do odebranego komunikatu, aby transakcja budujący odpowiedź mogła potwierdzić, że przetwarzane jest poprawne zapytanie. Komunikat ma postać trzech 79-znakowych grup:

```
'Response from CHECKING ACCOUNT for name : JONES J B'
'   Opened 870530, 3-month average balance = 000012.57'
'   Credit worthiness index - BBB'
```

Jeśli program działa jako BMP zorientowany na obsługę komunikatów, program odracza kolejkę komunikatów IMS , a następnie odczytuje komunikaty z kolejki IBM MQ for z/OS i przetwarza je. Z kolejki komunikatów IMS nie są odbierane żadne informacje. Program ponownie nawiązuje połączenie z menedżerem kolejek po każdym punkcie kontrolnym, ponieważ uchwyty zostały zamknięte.

Jeśli program działa w BMP zorientowanym na przetwarzanie wsadowe, program jest nadal połączony z menedżerem kolejek po każdym punkcie kontrolnym, ponieważ uchwyty nie są zamknięte.

## z/OS *Przykład procedury obsługi komunikatów w systemie z/OS*

Przykładowa aplikacja TSO procedury obsługi komunikatów umożliwia przeglądanie, przekazywanie i usuwanie komunikatów w kolejce. Przykład jest dostępny w językach C i COBOL.

## Przygotowywanie i uruchamianie przykładu

Wykonaj następujące kroki:

1. Przygotuj przykład zgodnie z opisem w sekcji “Przygotowywanie przykładowych aplikacji dla środowiska TSO w systemie z/OS” na stronie 1204.
2. Należy dostosować plik CLIST (CSQ4RCH1) udostępniony dla przykładu, aby zdefiniować położenie paneli, położenie pliku komunikatów i położenie modułów ładujących.

Za pomocą CLIST CSQ4RCH1 można uruchomić zarówno wersję C, jak i COBOL przykładu. Dostarczona wersja komendy CSQ4RCH1 uruchamia wersję w języku C i zawiera instrukcje dotyczące dostosowywania wersji w języku COBOL.

**Uwaga:**

1. Z tym przykładem nie są dostarczane żadne przykładowe definicje kolejek.
2. VS COBOL II nie obsługuje wielozadaniowości z interfejsem ISPF, dlatego nie należy używać przykładowej aplikacji Procedura obsługi komunikatów po obu stronach podzielonego ekranu. W przeciwnym razie wyniki będą nieprzewidywalne.

### z/OS *Korzystanie z przykładowej procedury obsługi komunikatów w systemie z/OS*

Po zainstalowaniu przykładu i wywołaniu go z dostosowanej listy CLIST CSQ4RCH1zostanie wyświetlony ekran, który przedstawia Rysunek 146 na stronie 1243 .

```
---------------------- IBM MQ for z/OS -- Samples ------------------------
COMMAND ===>
User Id : JOHNJ



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
```

*Rysunek 146. Ekran początkowy dla przykładu procedury obsługi komunikatów*

Wprowadź nazwę menedżera kolejek i nazwę kolejki do wyświetlenia (z rozróżnianiem wielkości liter) i zostanie wyświetlony ekran z listą komunikatów (patrz sekcja Rysunek 147 na stronie 1243 ).

```
---------------------- 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  JOHNJ    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 ********************************
```

*Rysunek 147. Ekran listy komunikatów dla przykładu Procedura obsługi komunikatów*

Na tym ekranie wyświetlane są pierwsze 99 komunikatów w kolejce, a dla każdego z nich wyświetlane są następujące pola:

**Nr komunikatu**
Numer komunikatu

**Data umieszczenia MM/DD/RRRR**
Data umieszczenia komunikatu w kolejce (GMT)

**Godzina umieszczenia GG:MM: SS**
Czas umieszczenia komunikatu w kolejce (GMT)

**Nazwa formatu**
MQMD.Format formatu

**Identyfikator użytkownika**
MQMD.UserIdentifier

**Typ aplikacji wstawiającej**
MQMD.PutApplType

**Nazwa aplikacji wstawiającej**
MQMD.PutApplName , pole

Wyświetlana jest również łączna liczba komunikatów w kolejce.

Z tego ekranu można wybrać komunikat, a nie numer pozycji kursora, a następnie go wyświetlić. Przykład zawiera sekcja Rysunek 148 na stronie 1244.

```
---------------------- IBM MQ for z/OS -- Samples ----- Row 1 to 35 of 35
COMMAND ==>

Queue Manager  : VM03                        :
Queue      : MQEI.IMS.BRIDGE.QUEUE              :
Forward to Q Mgr : VM03                       :
Forward to Queue : QL.TEST.ISCRES1                 :

Action : _ :  (D)elete (F)orward

Message Content :
--------------------------------------------------------------------------------
Message Descriptor
StrucId     : `MD `
Version     : 000000001
Report      : 000000000
MsgType     : 000000001
Expiry      : -00000001
Feedback     : 000000000
Encoding     : 000000785
CodedCharSetId  : 000000500
Format      : `MQIMS  `
Priority     : 000000000
Persistence   : 000000001
MsgId      : `C3E2D840E5D4F0F34040404040404040AF6B30F0A89B7605`X
CorrelId     : `00000000000000000000000000000000000000000000000000000000`X
BackoutCount   : 000000000
ReplyToQ     : `QL.TEST.ISCRES1                `
ReplyToQMgr   : `VM03                        `
UserIdentifier  : `NTSFV02   `
AccountingToken :
`06F2F5F5F3F0F1000000000000000000000000000000000000000000000000000`X
ApplIdentityData : `                        `
PutApplType   : 000000002
PutApplName   : `NTSFV02A          `
PutDate     : `19971016`
PutTime     : `13511903`
ApplOriginData  : `   `

Message Buffer :  108 byte(s)
00000000 : C9C9 C840 0000 0001 0000 0054 0000 0311 `IIH .....T......`
00000010 : 0000 0000 4040 4040 4040 4040 0000 0000 `....    ....`
00000020 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000030 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000040 : 0000 0000 0000 0000 0000 0000 0000 0000 `................`
00000050 : 40F1 C300 0018 0000 C9C1 D7D4 C4C9 F2F8 ` 1C.....IAPMDI28`
00000060 : 40C8 C5D3 D3D6 40E6 D6D9 D3C4      ` HELLO WORLD  `
****************************** Bottom of data ********************************
```

*Rysunek 148. Wyświetlany jest wybrany komunikat*

Po wyświetleniu komunikatu można go usunąć, pozostawić w kolejce lub przekazać do innej kolejki. Pola `Forward to Q Mgr` i `Forward to Queue` są inicjowane wartościami z deskryptora MQMD. Te wartości można zmienić przed przekazaniem komunikatu.

Przykładowy projekt umożliwia wybranie i wyświetlenie tylko komunikatów z unikalnymi kombinacjami `MsgId / CorrelId` , ponieważ komunikat jest pobierany przy użyciu klawiszy `MsgId` i `CorrelId` . Jeśli klucz nie jest unikalny, przykład nie może pobrać wybranego komunikatu z pewnością.

**Uwaga:** Jeśli do przeglądania komunikatów używany jest przykład SCSQCLST (CSQ4RCH1), każde wywołanie powoduje zwiększenie liczby wycofanych komunikatów. Aby zmienić zachowanie tego przykładu, należy skopiować przykład i zmodyfikować jego zawartość zgodnie z potrzebami. Należy pamiętać, że ta rosnąca liczba może mieć wpływ na inne aplikacje, które korzystają z tej liczby wycofań.

z/OS *Projekt przykładowej procedury obsługi komunikatów w systemie z/OS*

W tym temacie opisano projekt każdego z programów, które tworzą przykładową aplikację procedury obsługi komunikatów.

## Program sprawdzający poprawność obiektu

Spowoduje to żądanie poprawnej nazwy kolejki i menedżera kolejek.

Jeśli nazwa menedżera kolejek nie zostanie określona, zostanie użyty domyślny menedżer kolejek (jeśli jest dostępny). Można używać tylko kolejek lokalnych. Jeśli kolejka nie jest lokalna, wysyłany jest komunikat MQINQ w celu sprawdzenia, czy typ kolejki i błąd zostały zgłoszone. Jeśli kolejka nie zostanie pomyślnie otwarta lub wywołanie MQGET zostanie zablokowane, zostaną zwrócone komunikaty o błędach wskazujące kod powrotu CompCode i kod powrotu przyczyny.

## Program listy komunikatów

Spowoduje to wyświetlenie listy komunikatów w kolejce z informacjami o nich, takimi jak putdate, puttime i format komunikatu.

Maksymalna liczba komunikatów przechowywanych na liście wynosi 99. Jeśli w kolejce znajduje się więcej komunikatów, wyświetlana jest również bieżąca głębokość kolejki. Aby wybrać komunikat do wyświetlenia, wpisz numer komunikatu w polu wprowadzania (wartość domyślna to 01). Jeśli wpis nie jest poprawny, zostanie wyświetlony odpowiedni komunikat o błędzie.

## Program treści komunikatu

Spowoduje to wyświetlenie treści komunikatu.

Treść jest formatowana i dzielona na dwie części:

1. deskryptor komunikatu
2. Bufor komunikatów

Deskryptor komunikatu wyświetla zawartość każdego pola w osobnym wierszu.

Bufor komunikatów jest formatowany w zależności od jego zawartości. Jeśli bufor zawiera nagłówek niedostarczonego komunikatu (MQDLH) lub nagłówek kolejki transmisji (MQXQH), są one formatowane i wyświetlane przed samym buforem.

Przed sformatowaniem danych buforu wiersz tytułowy przedstawia długość buforu komunikatu w bajtach. Maksymalna wielkość buforu wynosi 32768 bajtów, a każdy komunikat dłuższy niż ten jest obcinany. Wyświetlana jest pełna wielkość buforu wraz z komunikatem wskazującym, że wyświetlane są tylko pierwsze 32768 bajtów komunikatu.

Dane buforu są formatowane na dwa sposoby:

1. Po wydrukowaniu przesunięcia w buforze dane buforu są wyświetlane w postaci szesnastkowej.
2. Dane buforu są następnie ponownie wyświetlane jako wartości EBCDIC. Jeśli nie można wydrukować żadnej wartości EBCDIC, zamiast tego drukowana jest kropka (.).

W polu czynności można wprowadzić D dla usunięcia lub F dla przekazania. Jeśli zostanie wybrana opcja przekazania komunikatu, wartości `forward-to queue` i `queue manager name` muszą być ustawione poprawnie. Wartości domyślne dla tych pól są odczytywane z pól deskryptora komunikatu ReplyToQ i ReplyToQMgr.

Jeśli komunikat jest przesyłany, każdy blok nagłówka zapisany w buforze jest pozbawiany. Jeśli komunikat zostanie przekazany pomyślnie, zostanie usunięty z oryginalnej kolejki. Jeśli zostaną wprowadzone niepoprawne działania, zostaną wyświetlone komunikaty o błędach.

Dostępny jest również przykładowy panel pomocy o nazwie CSQ4CHP9 .

### z/OS *Przykład asynchronicznego umieszczania w systemie z/OS*

Przykładowy program asynchronicznego umieszczania umieszcza komunikaty w kolejce przy użyciu asynchronicznego wywołania MQPUT. Przykład pobiera również informacje o statusie przy użyciu wywołania MQSTAT.

Aplikacje asynchronicznego umieszczania używają następujących wywołań MQI:

- ZMQCONN
- MQOPEN
- MQPUT
- MQSTAT
- MQCLOSE
- MQDISC

Programy przykładowe są dostarczane w języku programowania C.

Aplikacje asynchronicznego umieszczania działają w środowisku wsadowym. Informacje na temat aplikacji wsadowych zawiera sekcja Inne przykłady .

Ten temat zawiera również informacje na temat projektowania programu asynchronicznego konsumpcji i uruchamiania przykładu CSQ4BCS2 .


- "Uruchamianie przykładu CSQ4BCS2" na stronie 1246
- "Projekt przykładowego programu asynchronicznego umieszczania" na stronie 1246


## Uruchamianie przykładu CSQ4BCS2

Ten przykładowy program przyjmuje maksymalnie sześć parametrów:

1. Nazwa kolejki docelowej (wymagane).
2. Nazwa menedżera kolejek (opcjonalna).
3. Opcje otwierania (opcjonalnie).
4. Opcje zamykania (opcjonalnie).
5. Nazwa docelowego menedżera kolejek (opcjonalna).
6. Nazwa kolejki dynamicznej (opcjonalna).

Jeśli menedżer kolejek nie jest określony, komenda CSQ4BCS2 nawiązuje połączenie z domyślnym menedżerem kolejek. Treść komunikatu jest udostępniana przez wejście standardowe ( **SYSIN DD** ).

Istnieje przykładowy kod JCL służący do uruchamiania programu, który znajduje się w CSQ4BCSP.

## Projekt przykładowego programu asynchronicznego umieszczania

Program używa wywołania MQOPEN z podanymi opcjami wyjściowymi lub z opcjami MQOO_OUTPUT i MQOO_FAIL_IF_QUIESCING do otwierania kolejki docelowej w celu umieszczania komunikatów.

Jeśli program nie może otworzyć kolejki, program generuje komunikat o błędzie zawierający kod przyczyny zwrócony przez wywołanie MQOPEN. Aby program ten i kolejne wywołania MQI były proste, dla wielu opcji używane są wartości domyślne.

Dla każdego wiersza danych wejściowych program odczytuje tekst do buforu i używa wywołania MQPUT z odpowiedzią MQPMO_ASYNC_RESPONSE w celu utworzenia komunikatu datagramu zawierającego tekst tego wiersza i asynchronicznie umieszcza komunikat w kolejce docelowej. Działanie programu jest kontynuowane do momentu osiągnięcia końca danych wejściowych lub do momentu niepowodzenia wywołania MQPUT. Jeśli program osiągnie koniec danych wejściowych, zamyka kolejkę za pomocą wywołania MQCLOSE.

Następnie program wysyła wywołanie MQSTAT, które zwraca strukturę MQSTS i wyświetla komunikaty zawierające liczbę pomyślnie umieszczonych komunikatów, liczbę umieszczonych komunikatów z ostrzeżeniem oraz liczbę niepowodzeń.

**Uwaga:** Aby obserwować, co się dzieje, gdy błąd MQPUT zostanie wykryty przez wywołanie MQSTAT, należy ustawić parametr MAXDEPTH w kolejce docelowej na niską wartość.

## z/OS *Przykład wykorzystania asynchronicznego zadania wsadowego w systemie z/OS*

Program przykładowy CSQ4BCS1 jest dostarczany w języku C i demonstruje użycie MQCB i MQCTL do asynchronicznego odbierania komunikatów z wielu kolejek.

Przykłady wykorzystania asynchronicznego są uruchamiane w środowisku wsadowym. Informacje na temat aplikacji wsadowych zawiera sekcja Inne przykłady .

Dostępny jest również przykład w języku COBOL, który działa w środowisku CICS (patrz sekcja “Przykład CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS” na stronie 1248).

Aplikacje używają następujących wywołań MQI:

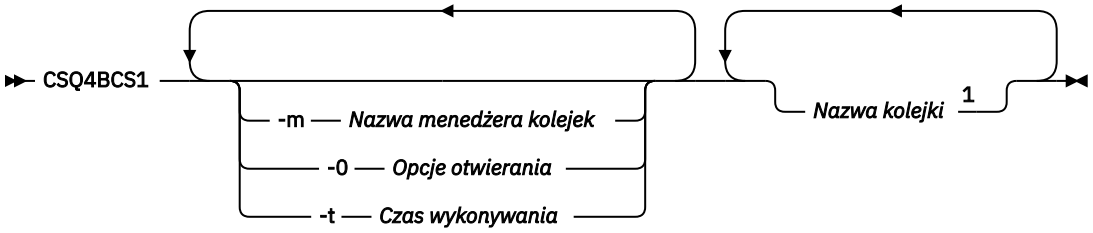
- ZMQCONN
- MQOPEN
- MQCLOSE
- MQDISC
- Baza MQCB
- Komenda MQCTL

Ten temat zawiera również informacje o następujących nagłówkach:

- “Uruchamianie przykładu CSQ4BCS1” na stronie 1247
- “Projekt przykładowego programu asynchronicznego przetwarzania wsadowego” na stronie 1247

### Uruchamianie przykładu CSQ4BCS1

Przykładowy program jest zgodny z następującą składnią:




Uwagi:

[1] W tym przykładzie obsługiwanych jest maksymalnie 10 kolejek. Aby odczytywać komunikaty z wielu kolejek, należy podać więcej niż jedną nazwę kolejki.

Do uruchomienia tego programu dostępny jest przykładowy kod JCL, który znajduje się w pliku CSQ4BCSC.

### Projekt przykładowego programu asynchronicznego przetwarzania wsadowego

Przykład przedstawia sposób odczytywania komunikatów z wielu kolejek w kolejności ich nadejścia. Wymaga to więcej kodu przy użyciu synchronicznego wywołania MQGET. W przypadku wykorzystania asynchronicznego nie jest wymagane żadne odpytywanie, a zarządzanie wątkami i pamięcią masową jest wykonywane przez produkt IBM MQ. W programie przykładowym błędy są zapisywane w konsoli.

Kod przykładowy składa się z następujących kroków:

1. Zdefiniuj funkcję zwrotną pojedynczego wykorzystania komunikatów.

```
void MessageConsumer(MQHCONN   hConn,
MQMD  * pMsgDesc,
MQGMO  * pGetMsgOpts,
MQBYTE * Buffer,
MQCBC  * pContext)
{ ... }
```

2. Nawiąż połączenie z menedżerem kolejek.

```
MQCONN(QMName,&Hcon,&CompCode,&CReason);
```

3. Otwórz kolejki wejściowe i powiąż każdą kolejkę z funkcją zwrotną `MessageConsumer` .

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   Parametr `cbd.CallbackFunction` nie musi być ustawiany dla każdej kolejki; jest to pole tylko wejściowe. Z każdą kolejką można powiązać inną funkcję zwrotną.

4. Rozpocznij przetwarzanie komunikatów.

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Poczekaj, aż użytkownik naciśnie klawisz Enter, a następnie zatrzymaj przetwarzanie komunikatów.

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. Na koniec rozłącz się z menedżerem kolejek.

```
MQDISC(&Hcon,&CompCode,&Reason);
```

### z/OS *Przykład CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS*

Programy przykładowe Wykorzystanie asynchroniczne oraz Publikowanie/subskrypcja demonstrują użycie wykorzystania asynchronicznego oraz funkcji publikowania i subskrypcji w produkcie CICS.

Program *Klient rejestracji* rejestruje trzy procedury obsługi wywołania zwrotnego (procedurę obsługi zdarzeń i dwa konsumenty komunikatów) i uruchamia asynchroniczne konsumowanie. Program *klienta przesyłania komunikatów* umieszcza komunikaty w kolejce lub publikuje odpowiednie komunikaty z konsoli CICS w celu ich wykorzystania przez dwa konsumenty komunikatów (CSQ4CVCN i CSQ4CVCT).

Aby zapewnić kontrolę środowiska wykonawczego nad zachowaniem przykładu, jeden z konsumentów komunikatów może zostać poinstruowany przy użyciu odebranych komunikatów, aby zawiesić, wznowić lub DEREGISTER dowolną z procedur obsługi wywołania zwrotnego. Można go również użyć do wywołania komendy MQCTL STOP w celu zakończenia używania asynchronicznego pod kontrolą. Drugi konsument komunikatów jest zarejestrowany do subskrybowania tematu.

Każdy program wydaje instrukcje DISPLAY języka COBOL w odpowiednich punktach, aby wyświetlić zachowanie przykładu.

Aplikacje używają następujących wywołań MQI:

- MQOPEN
- MQPUT
- MQSUB

- MQGET
- MQCLOSE
- Baza MQCB
- Komenda MQCTL

Programy są dostarczane w języku COBOL. Więcej informacji na ten temat zawiera sekcja CICS Przykłady asynchronicznego wykorzystania i publikowania/subskrypcji dla aplikacji CICS .

Ten temat zawiera również następujące informacje:


- "Konfiguracja" na stronie 1249
- "Klient rejestracji CSQ4CVRG" na stronie 1249
- "Procedura obsługi zdarzeń CSQ4CVEV" na stronie 1249
- "Prosty konsument komunikatów CSQ4CVCN" na stronie 1250
- "Konsument komunikatów sterujących CSQ4CVCT" na stronie 1250
- "Klient przesyłania komunikatów CSQ4CVPT" na stronie 1250


## Konfiguracja

Nazwy kolejki i tematu używane przez konsumentów komunikatów są zakodowane na stałe w programach klienta rejestracji i przesyłania komunikatów.

Kolejka, **SAMPLE.CONTROL.QUEUE**powinna być zdefiniowana w menedżerze kolejek powiązanym z regionem CICS przed uruchomieniem przykładu. W razie potrzeby można zdefiniować temat **Wiadomości/Media/Filmy**lub utworzyć go w środowisku wykonawczym przy użyciu domyślnego obiektu administracyjnego (jeśli nie istnieje).

Programy CICS i definicje transakcji można zainstalować, instalując grupę: CSQ4SAMP.

## Klient rejestracji CSQ4CVRG

Program Registration Client musi być uruchomiony w ramach transakcji MVRG systemu CICS . Nie przyjmuje żadnych danych wejściowych.

Po uruchomieniu klient rejestracji rejestruje następujące procedury obsługi wywołania zwrotnego przy użyciu obiektu MQCB:

- CSQ4CVEV jako procedura obsługi zdarzeń.
- CSQ4CVCN jako konsument komunikatów w temacie **Wiadomości/Media/Filmy**.
- CSQ4CVCT jako konsument komunikatów w kolejce, **SAMPLE.CONTROL.QUEUE**.

Klient rejestracji przekazuje strukturę danych zawierającą nazwy wszystkich trzech zarejestrowanych procedur obsługi wywołania zwrotnego do CSQ4CVCTwraz z uchwytami obiektów powiązanymi z dwoma konsumentami komunikatów.

Po zarejestrowaniu procedur obsługi wywołania zwrotnego klient rejestracji wysyła komendę MQCTL START_WAIT w celu uruchomienia asynchronicznego konsumowania i zawiesza działanie do momentu, gdy zostanie do niego zwrócona kontrola (na przykład przez jedną z procedur obsługi wywołania zwrotnego wywołujących komendę MQCTL STOP).

## Procedura obsługi zdarzeń CSQ4CVEV

Po sterowaniu procedura obsługi zdarzeń wyświetla komunikat wskazujący typ wywołania (na przykład START). W przypadku sterowania kodem przyczyny IBM MQ CONNECTION_QUIESCING procedura obsługi zdarzeń wysyła komendę MQCTL STOP w celu zakończenia asynchronicznego wykorzystania i zwrócenia kontroli do klienta rejestracji.

## Prosty konsument komunikatów CSQ4CVCN

Po sterowaniu ten konsument komunikatów wyświetla komunikat wskazujący typ wywołania (na przykład REGISTER). W przypadku wywołania typu MSG_REMOVED konsument komunikatów pobiera komunikat przychodzący i zapisuje go w dzienniku zadania CICS .

## Konsument komunikatów sterujących CSQ4CVCT

Po sterowaniu ten konsument komunikatów wyświetla komunikat wskazujący typ wywołania (na przykład START). W przypadku wywołania typu MSG_REMOVED konsument komunikatów pobiera komunikat przychodzący i strukturę danych przekazywane przez klienta rejestracji. W zależności od treści komunikatu wysyła on odpowiednie komendy MQCB lub MQCTL do jednego z następujących elementów:

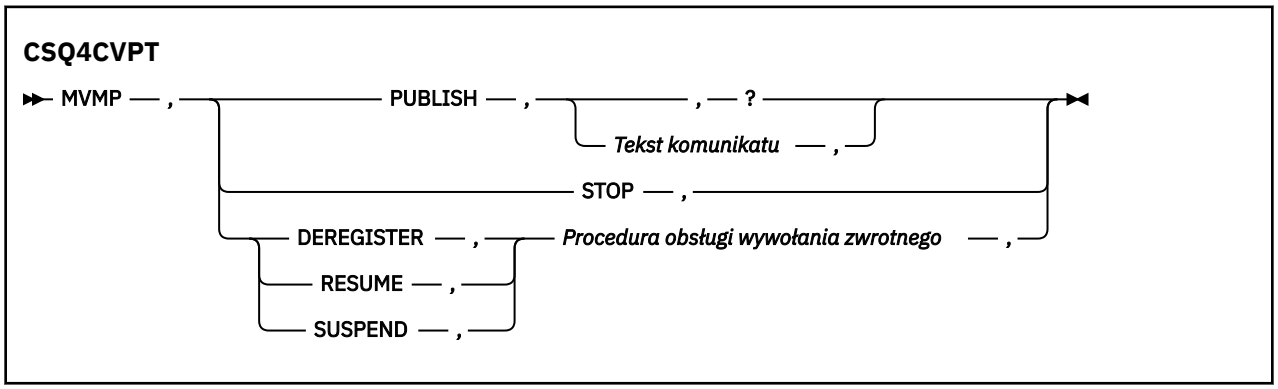
- STOP Asynchronous Konsumpcja (zwracanie kontroli do klienta rejestracji).
- SUSPEND, RESUME lub DEREGISTER nazwanego programu obsługi wywołania zwrotnego (w tym samego siebie).

## Klient przesyłania komunikatów CSQ4CVPT

Klient przesyłania komunikatów ma dwie funkcje:

- Publikuje komunikat w temacie do wykorzystania przez konsument komunikatów CSQ4CVCN.
- Komunikat sterujący jest umieszczany w kolejce w celu jego wykorzystania przez konsument komunikatów sterujących CSQ4CVCT, co może spowodować zmianę zachowania przykładu.

Program Messaging Client musi zostać uruchomiony z konsoli CICS w ramach transakcji CICS i przyjmuje on dane wejściowe wiersza komend z następującą składnią:




**PUBLISH**
Opublikuj tekst komunikatu (lub komunikat domyślny) jako zachowany komunikat do wykorzystania przez prosty konsument komunikatów.

**STOP**
Zatrzymaj wykorzystanie asynchroniczne.

**DEREGISTER (Deregator)**
Wyrejestruj nazwaną procedurę obsługi wywołania zwrotnego.

**RESUME**
Wznów nazwaną procedurę obsługi wywołania zwrotnego.

**SUSPEND**
Zawieś nazwaną procedurę obsługi wywołania zwrotnego.

Pola wejściowe są pozycyjne i rozdzielane przecinkami. W nazwach słów kluczowych i procedur obsługi wywołania zwrotnego nie jest rozróżniana wielkość liter.

Przykłady:

| *Tabela 186. Przykłady danych wejściowych* | |
|---|---|
| **Przykład** | **Opis** |
| MVMP, OPUBLIKUJ, | Opublikuj komunikat domyślny |
| MVMP,publish, `A short message`, | Opublikuj podany tekst |
| MVMP, ZATRZYMAJ, | Zatrzymaj wykorzystanie asynchroniczne |
| MVMP,DEREGISTER,CSQ4CVEV, | Wyrejestrowywania procedury obsługi zdarzeń |
| MVMP,resume,csq4cvcn, | Wznów prosty konsument komunikatów |
| MVMP,SUSPEND,CSQ4CVEV, | Zawieś procedurę obsługi zdarzeń |

Gdzie MVMP to transakcja CICS powiązana z programem klienta przesyłania komunikatów CSQ4CVPT.

**Uwaga:**

- Zawieszenie lub wyrejestrowanie wszystkich procedur obsługi wywołania zwrotnego powoduje zakończenie operacji START_WAIT wywołanej przez klienta rejestracji, zwrócenie do niego kontroli i zakończenie zadania.
- Zawieszenie lub wyrejestrowanie procedury obsługi wywołania zwrotnego sterowania nie zostało celowo uniemożliwione, ale eliminuje możliwość dalszego kontrolowania zachowania przykładu.

### z/OS *Przykład publikowania/subskrypcji w serwisie z/OS*

Przykładowe programy publikowania/subskrypcji demonstrują użycie funkcji publikowania i subskrypcji w produkcie IBM MQ.

Istnieją cztery przykładowe programy w języku programowania C i dwa w języku COBOL demonstrujące sposób programowania w interfejsie publikowania/subskrypcji produktu IBM MQ . Programy są dostarczane w językach C i COBOL. Aplikacje działają w środowisku przetwarzania wsadowego. Informacje na temat aplikacji wsadowych zawiera sekcja Przykłady publikowania/subskrypcji .

Istnieją również przykłady w języku COBOL, które działają w środowisku CICS ; patrz sekcja “Przykład CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS” na stronie 1248.

Ten temat zawiera również informacje dotyczące uruchamiania przykładowych programów publikowania/subskrypcji. Do tych przykładowych programów należą:


- “Uruchamianie przykładu CSQ4BCP1” na stronie 1251
- “Uruchamianie przykładu CSQ4BCP2” na stronie 1252
- “Uruchamianie przykładu CSQ4BCP3” na stronie 1252
- “Uruchamianie przykładu CSQ4BCP4” na stronie 1252
- “Uruchamianie przykładu CSQ4BVP1” na stronie 1253
- “Uruchamianie przykładu CSQ4BVP2” na stronie 1253


## Uruchamianie przykładu CSQ4BCP1

Ten program jest napisany w języku C; publikuje komunikaty w temacie. Przed uruchomieniem tego programu uruchom jeden z przykładowych subskrybentów.

Ten program przyjmuje do czterech parametrów:

1. Nazwa docelowego łańcucha tematu (wymagane).
2. Nazwa menedżera kolejek (opcjonalna).
3. Opcje otwierania (opcjonalnie).
4. Opcje zamykania (opcjonalnie).

Jeśli menedżer kolejek nie jest określony, komenda CSQ4BCP1 nawiązuje połączenie z domyślnym menedżerem kolejek. Istnieje przykładowy kod JCL służący do uruchamiania programu, który znajduje się w katalogu CSQ4BCPP.

Treść komunikatu jest udostępniana przez wejście standardowe (**SYSIN DD**).

## Uruchamianie przykładu CSQ4BCP2

Ten program jest napisany w języku C; subskrybuje temat i drukuje odebrane komunikaty.

Program ten może mieć do trzech parametrów:

1. Nazwa docelowego łańcucha tematu (wymagane).
2. Nazwa menedżera kolejek (opcjonalna).
3. Opcje subskrypcji MQSD (opcjonalne).

Jeśli menedżer kolejek nie jest określony, komenda CSQ4BCP2 nawiązuje połączenie z domyślnym menedżerem kolejek. Istnieje przykładowy kod JCL służący do uruchamiania programu, który znajduje się w katalogu CSQ4BCPS.

## Uruchamianie przykładu CSQ4BCP3

Ten program jest napisany w języku C; subskrybuje temat przy użyciu kolejki docelowej określonej przez użytkownika i drukuje odebrane komunikaty.

Ten program przyjmuje do czterech parametrów:

1. Nazwa docelowego łańcucha tematu (wymagane).
2. Nazwa miejsca docelowego (wymagane).
3. Nazwa menedżera kolejek (opcjonalna).
4. Opcje subskrypcji MQSD (opcjonalne).

Jeśli nie określono menedżera kolejek, komenda CSQ4BCP3 nawiązuje połączenie z domyślnym menedżerem kolejek. Istnieje przykładowy kod JCL służący do uruchamiania programu, który znajduje się w katalogu CSQ4BCPD.

## Uruchamianie przykładu CSQ4BCP4

Ten program jest napisany w języku C; subskrybuje i pobiera komunikaty z tematu, umożliwiając użycie opcji rozszerzonych w wywołaniu MQSUB, rozszerzając opcje dostępne w prostszym przykładzie MQSUB: CSQ4BCP2. Oprócz ładunku komunikatu, dla każdego komunikatu są odbierane i wyświetlane właściwości komunikatu.

Ten program przyjmuje zestaw zmiennych parametrów:

- **-t** *Topic string*.
- **-o** *Topic object name*.

  **Ważne:** Wymagany jest jeden z następujących elementów: **-t** , **-o**lub oba te elementy
- **-m** *Queue manager name* (opcjonalnie).
- **-b** *Connection binding type* (opcjonalnie), gdzie *typ* może mieć jedną z następujących wartości:
  - *standard* : MQCNO_STANDARD_BINDING, która jest wartością domyślną.
  - *shared*: MQCNO_SHARED_BINDING
  - *krótka ścieżka*: MQCNO_FASTPATH_BINDING
  - *izolowane*: MQCNO_IZOLOWANE_POWIĄZANIE
- **-q** *Destination queue name* (opcjonalnie).
- **-w** *Wait interval on MQGET in seconds* (opcjonalnie), gdzie *sekundy* mogą mieć jedną z następujących wartości:

  - `unlimited`: NIEOGRANICZONE MQWI_UNLIMITED
  - `none`: bez oczekiwania
  - *n*: odstęp czasu oczekiwania w sekundach
  - Nie określono wartości: jeśli nie określono żadnej wartości, wartością domyślną jest 30 sekund.
- **-d** *Subscription name* (opcjonalnie). Tworzy lub wznawia nazwaną trwałą subskrypcję.
- **-k** (opcjonalnie). Utrzymuje trwałą subskrypcję w MQCLOSE.

Jeśli menedżer kolejek nie jest określony, program CSQ4BCP4 nawiązuje połączenie z domyślnym menedżerem kolejek. Dostępny jest przykładowy kod JCL służący do uruchamiania programu, który znajduje się w katalogu CSQ4BCPE.

## Uruchamianie przykładu CSQ4BVP1

Ten program jest napisany w języku COBOL, publikuje komunikaty w temacie. Przed uruchomieniem tego programu uruchom jeden z przykładowych subskrybentów.

Ten program nie przyjmuje żadnych parametrów. Program **SYSIN DD** udostępnia nazwę tematu wejściowego, nazwę menedżera kolejek i treść komunikatu.

Jeśli nie określono menedżera kolejek, komenda CSQ4BVP1 nawiązuje połączenie z domyślnym menedżerem kolejek. Istnieje przykładowy kod JCL służący do uruchamiania programu, który znajduje się w katalogu CSQ4BVPP.

## Uruchamianie przykładu CSQ4BVP2

Ten program jest napisany w języku COBOL, subskrybuje temat i drukuje odebrane komunikaty.

Ten program nie przyjmuje żadnych parametrów. Program **SYSIN DD** udostępnia dane wejściowe dla nazwy tematu i nazwy menedżera kolejek.

Jeśli nie określono menedżera kolejek, komenda CSQ4BVP1 nawiązuje połączenie z domyślnym menedżerem kolejek. Istnieje przykładowy kod JCL służący do uruchamiania programu, który znajduje się w katalogu CSQ4BVPP.

## z/OS *Przykład właściwości Set and Inquire message property w systemie z/OS*

Przykładowe programy właściwości komunikatu demonstrują dodawanie zdefiniowanych przez użytkownika właściwości do uchwytu komunikatu oraz uzyskiwanie informacji o właściwościach powiązanych z tym komunikatem.

Aplikacje używają następujących wywołań MQI:

- ZMQCONN
- MQOPEN
- MQPUT
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP
- Komenda MQSETMP

Programy są dostarczane w języku C. Aplikacje działają w środowisku wsadowym. Informacje na temat aplikacji wsadowych zawiera sekcja Inne przykłady .

Program CSQ4BCM1 służy do uzyskiwania informacji o właściwościach uchwytu komunikatu z kolejki komunikatów i jest przykładem użycia wywołania API MQINQMP. Przykład pobiera jeden komunikat z kolejki, a następnie drukuje wszystkie właściwości uchwytu komunikatu.

Program CSQ4BCM2 służy do ustawiania właściwości uchwytu komunikatu w kolejce komunikatów i jest przykładem użycia wywołania API MQSETMP. Przykład tworzy uchwyt komunikatu i umieszcza go w polu `MsgHandle` struktury MQGMO. Następnie umieszcza komunikat w kolejce.

Inne przykłady sprawdzania i drukowania właściwości komunikatu znajdują się w programach przykładowych CSQ4BCG1 i CSQ4BCP4 .

Ten temat zawiera również informacje dotyczące uruchamiania przykładów właściwości Set and Inquire komunikatu pod następującymi nagłówkami:


- "Uruchamianie przykładu CSQ4BCM1" na stronie 1254
- "Uruchamianie przykładu CSQ4BCM2" na stronie 1254


### Uruchamianie przykładu CSQ4BCM1

Ten program przyjmuje do czterech parametrów:

1. Nazwa kolejki docelowej (wymagane).
2. Nazwa menedżera kolejek (opcjonalna).
3. Opcje otwierania (opcjonalnie).
4. Opcje zamykania (opcjonalnie).

### Uruchamianie przykładu CSQ4BCM2

Ten program przyjmuje do sześciu parametrów:

1. Nazwa kolejki docelowej (wymagane).
2. Nazwa menedżera kolejek (opcjonalna).
3. Opcje otwierania (opcjonalnie).
4. Opcje zamykania (opcjonalnie).
5. Nazwa docelowego menedżera kolejek (opcjonalna).
6. Nazwa kolejki dynamicznej (opcjonalna).

Nazwy właściwości, wartości i treść komunikatu są udostępniane za pośrednictwem wejścia standardowego ( **`SYSIN DD`** ). Dostępny jest przykładowy kod JCL umożliwiający uruchomienie programu, który znajduje się w katalogu CSQ4BCMP.

# Tworzenie aplikacji dla składnika Managed File Transfer

Określ programy, które mają być uruchamiane z systemem Managed File Transfer, użyj opcji Apache Ant z opcją Managed File Transfer, dopasuj opcję Managed File Transfer do procedur zewnętrznych i steruj opcją Managed File Transfer , umieszczając komunikaty w kolejce komend agenta.

## Określanie programów, które mają być uruchamiane z systemem MFT

Programy można uruchamiać w systemie, w którym działa Managed File Transfer Agent . W ramach żądania przesyłania plików można określić program, który ma zostać uruchomiony przed rozpoczęciem przesyłania lub po jego zakończeniu. Dodatkowo można uruchomić program, który nie jest częścią żądania przesyłania plików, wprowadzając żądanie wywołania zarządzanego.

### O tym zadaniu

Istnieje pięć scenariuszy, w których można określić program do uruchomienia:

- Jako część żądania przesyłania, w agencie źródłowym, przed rozpoczęciem przesyłania.

- Jako część żądania przesyłania, w agencie docelowym, przed rozpoczęciem przesyłania.
- W ramach żądania przesyłania, w agencie źródłowym, po zakończeniu przesyłania.
- W ramach żądania przesyłania, w agencie docelowym, po zakończeniu przesyłania.
- Nie jest częścią żądania przesyłania. Można wysłać do agenta żądanie uruchomienia programu. Ten scenariusz jest czasem nazywany wywołaniem zarządzanym.

Procedury zewnętrzne i wywołania programów są wywoływane w następującej kolejności:

```
- 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.
```

**Uwagi:**

1. Skrypt **DestinationTransferEndExits** jest uruchamiany tylko wtedy, gdy operacja przesyłania zakończy się pomyślnie lub częściowo pomyślnie.
2. Skrypt **postDestinationCall** jest uruchamiany tylko wtedy, gdy operacja przesyłania zakończy się pomyślnie lub częściowo pomyślnie.
3. Komenda **SourceTransferEndExits** jest uruchamiana w przypadku pomyślnych, częściowo pomyślnych lub nieudanych operacji przesyłania.
4. Funkcja **postSourceCall** jest wywoływana tylko wtedy, gdy:
   - Przesyłanie nie zostało anulowane.
   - Wynik jest pomyślny lub częściowo pomyślny.
   - Wszystkie programy przesyłania w miejscu docelowym zostały pomyślnie uruchomione.

## Procedura

- Określ program, który ma zostać uruchomiony, używając jednej z następujących opcji:

  **Użyj zadania Apache Ant**

  Użyj jednego z zadań `fte:filecopy`, `fte:filemove`i `fte:call` Ant , aby uruchomić program. Za pomocą zadania Ant można określić program w dowolnym z pięciu scenariuszy przy użyciu elementów zagnieżdżonych `fte:presrc`, `fte:predst`, `fte:postdst`, `fte:postsrci` `fte:command` . Więcej informacji na ten temat zawiera sekcja Elementy zagnieżdżone wywołania programu.

  **Edytuj komunikat żądania przesyłania plików**

  Można edytować kod XML, który jest generowany przez żądanie przesyłania. Za pomocą tej metody można uruchomić program w dowolnym z pięciu scenariuszy, dodając elementy **preSourceCall**, **postSourceCall**, **preDestinationCall**, **postDestinationCall**i **managedCall** do pliku XML. Następnie należy użyć tego zmodyfikowanego pliku XML jako definicji przesyłania dla nowego żądania przesyłania plików, na przykład z parametrem **fteCreateTransfer -td** . Więcej informacji na ten temat zawiera sekcja Przykłady komunikatów żądań wywołania agenta MFT.

  **Użycie komendy `fteCreateTransfer`**

  Do określenia programów do uruchomienia można użyć komendy **fteCreateTransfer** . Za pomocą tej komendy można określić programy, które mają być uruchamiane w pierwszych czterech scenariuszach, jako część żądania przesyłania, ale nie można uruchomić wywołania zarządzanego. Informacje na temat parametrów, które mają być używane, zawiera sekcja **fteCreateTransfer**: rozpoczynanie nowego przesyłania plików. Przykłady użycia tej komendy zawiera sekcja Przykłady używania komendy fteCreateTransfer do uruchamiania programów.

**Odsyłacze pokrewne**
Właściwość commandPath MFT

## Połączenia zarządzane

Agenty Managed File Transfer (MFT) są zwykle używane do przesyłania plików lub komunikatów. Są one nazywane *Zarządzanymi transferami*. Agenty mogą być również używane do uruchamiania komend, skryptów lub JCL bez konieczności przesyłania plików lub komunikatów. Ta możliwość jest nazywana *połączeniami zarządzanymi*.

Żądania połączeń zarządzanych można wysyłać do agenta na kilka sposobów:

- Przy użyciu zadania fte: call Ant.
- Konfigurowanie monitora zasobów z plikiem XML zadania, który uruchamia komendę lub skrypt. Więcej informacji na ten temat zawiera sekcja Konfigurowanie zadań monitorowania w celu uruchamiania komend i skryptów .
- Bezpośrednie umieszczanie komunikatu XML w kolejce komend agenta. Więcej informacji na temat schematu XML wywołania zarządzanego zawiera sekcja Format komunikatu żądania przesyłania plików .

W przypadku wywołań zarządzanych katalog zawierający uruchamianą komendę lub skrypt musi być określony we właściwości agenta **commandPath**.

Wywołania zarządzane nie mogą uruchamiać komend ani skryptów, które znajdują się w katalogach, które nie są określone w pliku **commandPath**agenta. Ma to na celu zapewnienie, że agent nie uruchomi żadnego złośliwego kodu.

**Ważne:** Aby upewnić się, że tak jest, domyślnie po podaniu wartości **commandPath**:

- Wszystkie istniejące środowiska testowe agentów są konfigurowane przez agenta podczas jego uruchamiania, dzięki czemu wszystkie katalogi **commandPath** są automatycznie dodawane do listy katalogów, które nie mają dostępu do operacji przesyłania.
- Wszystkie istniejące przestrzenie prywatne użytkownika są aktualizowane podczas uruchamiania agenta, dzięki czemu wszystkie katalogi **commandPath** (i ich podkatalogi) są dodawane jako elementy `<exclude>` do elementów `<read>` i `<write>` .
- Jeśli agent nie jest skonfigurowany do używania środowiska testowego agenta lub środowiska testowego użytkownika, podczas uruchamiania agenta tworzona jest nowa przestrzeń prywatna agenta, która zawiera katalogi **commandPath** określone jako katalogi odrzucone.

Ponadto można włączyć sprawdzanie uprawnień dla agenta, aby upewnić się, że tylko autoryzowani użytkownicy mogą wysyłać zarządzane żądania połączeń. Więcej informacji na ten temat zawiera sekcja Ograniczanie uprawnień użytkowników do działań agenta MFT.

Komenda, skrypt lub JCL wywołane jako część zarządzanego wywołania działa jako proces zewnętrzny, który jest monitorowany przez agenta. Po zakończeniu procesu wywołanie zarządzane zostaje zakończone, a kod powrotu z procesu jest udostępniany agentowi lub skryptowi Ant , który wywołał zadanie **fte:call** Ant .

Jeśli wywołanie zarządzane zostało uruchomione przez zadanie **fte:call** Ant , skrypt Ant może sprawdzić wartość kodu powrotu, aby określić, czy wywołanie zarządzane powiodło się.

Dla wszystkich innych typów wywołań zarządzanych można określić, które wartości kodów powrotu powinny być używane do wskazania, że wywołanie zarządzane zakończyło się pomyślnie. Agent porównuje kod powrotu z procesu z tymi kodami powrotu po zakończeniu procesu zewnętrznego.

**Uwaga:** Ponieważ wywołania zarządzane działają jako procesy zewnętrzne, nie można ich anulować po uruchomieniu.

### Zarządzane wywołania i źródłowe szczeliny przesyłania

Agent zawiera pewną liczbę źródłowych szczelin przesyłania, określonych przez właściwość agenta **maxSourceTransfers**, opisanych w sekcji Zaawansowane właściwości agenta: Limit przesyłania.

Za każdym razem, gdy uruchamiane jest połączenie zarządzane lub przesyłanie zarządzane, zajmują one źródłową szczelinę przesyłania. Gniazdo jest zwalniane po zakończeniu zarządzanego wywołania lub zarządzanego przesyłania.

Jeśli wszystkie źródłowe szczeliny przesyłania są używane, gdy agent odbiera nowe zarządzane wywołanie lub żądanie przesyłania zarządzanego, żądanie jest kolejkowane przez agenta, dopóki szczelina nie stanie się dostępna.

Jeśli wywołanie zarządzane uruchamia przesyłanie zarządzane (na przykład wywołanie zarządzane uruchamia skrypt Ant , a skrypt Ant używa zadania fte: filecopy lub fte: filemove w celu przesłania pliku), to wymagane są dwie źródłowe szczeliny przesyłania:

- Jeden dla zarządzanego przesyłania
- Jeden dla połączenia zarządzanego

W takiej sytuacji należy zauważyć, że jeśli przesyłanie zarządzane trwa długo lub jest wykonywane w celu odtworzenia, wówczas dwie źródłowe szczeliny przesyłania są zajęte do czasu zakończenia przesyłania zarządzanego, jego anulowania lub upłynięcia limitu czasu spowodowanego przez **transferRecoveryTimeout**. Szczegółowe informacje na temat **transferRecoveryTimeout**zawiera sekcja Pojęcia dotyczące limitu czasu odtwarzania przesyłania . Może to potencjalnie ograniczyć liczbę innych zarządzanych operacji przesyłania lub wywołań zarządzanych, które agent może przetworzyć.

Z tego powodu należy rozważyć zaprojektowanie zarządzanego wywołania, aby upewnić się, że nie zajmuje ono gniazd przesyłania źródłowego przez długi czas.

### Używanie programu REST API z połączeniami zarządzanymi

V 9.3.0 V 9.3.0

Komendy HTTP GET i HTTP POST są obsługiwane w przypadku włączania wywołań zarządzanych i działają tylko w wersji 3 serwera REST API.

Inne komendy, na przykład HTTP DELETE i HTTP UPDATE, nie są obsługiwane i zwracają kod błędu HTTP 405 w przypadku próby ich użycia.

**Ostrzeżenie:** Po wysłaniu połączenia zarządzanego nie można go anulować przy użyciu programu REST API.

## Używanie produktu Apache Ant z produktem MFT

Managed File Transfer udostępnia zadania, których można użyć do zintegrowania funkcji przesyłania plików z narzędziem Apache Ant .

Za pomocą komendy **fteAnt** można uruchomić zadania Ant w środowisku Managed File Transfer , które zostało już skonfigurowane. Za pomocą zadań przesyłania plików Ant ze skryptów Ant można koordynować złożone operacje przesyłania plików z poziomu interpretowanego języka skryptowego.

Więcej informacji na temat produktu Apache Antzawiera strona WWW projektu Apache Ant : https://ant.apache.org/

**Pojęcia pokrewne**

“Pierwsze kroki ze skryptami Ant w produkcie MFT” na stronie 1258
Użycie skryptów Ant z językiem Managed File Transfer umożliwia skoordynowanie złożonych operacji przesyłania plików z interpretowanego języka skryptowego.

**fteAnt**: uruchamianie zadań Ant w programie MFT

**Odsyłacze pokrewne**

“Przykładowe zadania programu Ant dla systemu MFT” na stronie 1259
Wraz z instalacją produktu Managed File Transferudostępniono kilka przykładowych skryptów Ant . Przykłady te znajdują się w katalogu *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Każdy przykładowy skrypt zawiera cel `init` , zmodyfikuj właściwości ustawione w celu `init` , aby uruchomić te skrypty z konfiguracją użytkownika.

## Pierwsze kroki ze skryptami Ant w produkcie MFT

Użycie skryptów Ant z językiem Managed File Transfer umożliwia skoordynowanie złożonych operacji przesyłania plików z interpretowanego języka skryptowego.

### Skrypty Ant

Skrypty Ant (lub pliki budowania) są dokumentami XML definiującymi co najmniej jeden element docelowy. Te cele zawierają elementy zadań do uruchomienia. Managed File Transfer udostępnia zadania, których można użyć do zintegrowania funkcji przesyłania plików z produktem Apache Ant. Aby uzyskać więcej informacji o skryptach Ant , należy zapoznać się ze stroną WWW projektu Apache Ant : https://ant.apache.org/

Przykłady skryptów Ant , które używają zadań Managed File Transfer , są dostarczane razem z instalacją produktu w katalogu *MQ_INSTALLATION_PATH*/mqft/samples/fteant .

W przypadku agentów mostu protokołu skrypty Ant są uruchamiane w systemie agenta mostu protokołu. Te skrypty Ant nie mają bezpośredniego dostępu do plików na serwerze FTP lub SFTP.

### Przestrzeń nazw

Przestrzeń nazw jest używana do odróżniania zadań Ant przesyłania plików od innych zadań Ant , które mogą mieć tę samą nazwę. Przestrzeń nazw jest definiowana w znaczniku projektu skryptu Ant .

```
<?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>
```

Atrybut `xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs"` informuje produkt Ant o konieczności wyszukania w bibliotece `com.ibm.wmqfte.ant.taskdefs`definicji zadań z przedrostkiem `fte` .

Nie trzeba używać przedrostka `fte` jako przedrostka przestrzeni nazw; można użyć dowolnej wartości. Przedrostek przestrzeni nazw `fte` jest używany we wszystkich przykładach i przykładowych skryptach Ant .

### Uruchamianie skryptów Ant

Aby uruchomić skrypty systemu Ant , które zawierają zadania przesyłania plików Ant , należy użyć komendy **fteAnt** . Na przykład:

```
fteAnt -file ant_script_location/ant_script_name
```

Więcej informacji na ten temat zawiera sekcja **fteAnt**: uruchamianie zadań Ant w produkcie MFT.

### Kody powrotu

Zadania przesyłania plików Ant zwracają te same kody powrotu, co komendy Managed File Transfer . Więcej informacji na ten temat zawiera sekcja Kody powrotu produktu MFT.

**Odsyłacze pokrewne**

**fteAnt**: uruchamianie zadań Ant w programie MFT

“Przykładowe zadania programu Ant dla systemu MFT” na stronie 1259

Wraz z instalacją produktu Managed File Transferudostępniono kilka przykładowych skryptów Ant . Przykłady te znajdują się w katalogu *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Każdy przykładowy skrypt zawiera cel `init` , zmodyfikuj właściwości ustawione w celu `init` , aby uruchomić te skrypty z konfiguracją użytkownika.

## Przykładowe zadania programu Ant dla systemu MFT

Wraz z instalacją produktu Managed File Transferudostępniono kilka przykładowych skryptów Ant . Przykłady te znajdują się w katalogu *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Każdy przykładowy skrypt zawiera cel `init` , zmodyfikuj właściwości ustawione w celu `init` , aby uruchomić te skrypty z konfiguracją użytkownika.

### Adres e-mail

Przykład email demonstruje sposób użycia zadań programu Ant do przesłania pliku i wysłania wiadomości e-mail na określony adres e-mail, jeśli przesyłanie nie powiedzie się. Skrypt sprawdza, czy agent źródłowy i docelowy są aktywne i mogą przetwarzać operacje przesyłania za pomocą zadania ping Managed File Transfer . Jeśli oba agenty są aktywne, skrypt używa zadania Managed File Transfer fte: filecopy do przesłania pliku między agentem źródłowym i docelowym bez usuwania oryginalnego pliku. Jeśli przesyłanie nie powiedzie się, skrypt wysyła wiadomość e-mail zawierającą informacje o niepowodzeniu przy użyciu standardowego zadania poczty elektronicznej produktu Ant .

### centrum

Przykład koncentratora składa się z dwóch skryptów: hubcopy.xml i hubprocess.xml. Skrypt hubcopy.xml pokazuje, w jaki sposób można używać skryptów Ant do budowania topologii w stylu gwiaździstej. W tym przykładzie dwa pliki są przesyłane z agentów działających na komputerach, na których działa serwer podległy, do agenta działającego na serwerze koncentrującym. Oba pliki są przesyłane w tym samym czasie, a po zakończeniu przesyłania skrypt hubprocess.xml Ant jest uruchamiany na komputerze koncentratora w celu przetworzenia plików. Jeśli oba pliki są przesyłane poprawnie, skrypt Ant konkatenuje zawartość plików. Jeśli pliki nie zostaną poprawnie przesłane, skrypt Ant usunie wszystkie przesłane dane plików. Aby ten przykład działał poprawnie, należy umieścić skrypt hubprocess.xml w ścieżce komend agenta koncentrującego monitorowanie. Więcej informacji na temat ustawiania ścieżki komend agenta zawiera sekcja commandPath MFT właściwość.

### Librarytransfer (tylko platformaIBM i )

IBM i

IBM i Przykład librarytransfer przedstawia sposób użycia zadań Ant do przesłania biblioteki IBM i w jednym systemie IBM i do drugiego systemu IBM i .

IBM i Przykład librarytransfer wykorzystuje obsługę rodzimego zbioru składowania w systemie IBM i z predefiniowanymi zadaniami Ant dostępnymi w systemie Managed File Transfer do przesyłania obiektów biblioteki rodzimej między dwoma systemami IBM i . W przykładzie użyto elementu zagnieżdżonego < presrc> w zadaniu Managed File Transfer filecopy do wywołania skryptu wykonywalnego `librarysave.sh` , który zapisuje żądaną bibliotekę w systemie agenta źródłowego w tymczasowym pliku zapisu. Zbiór składowania jest przenoszony przez zadanie Ant filecopy do docelowego systemu agenta, w którym zagnieżdżony element < postdst> jest używany do wywołania skryptu wykonywalnego `librayrestore.sh` w celu odtworzenia biblioteki zapisanej w zbiorze składowania w systemie docelowym.

IBM i Przed uruchomieniem tego przykładu należy przeprowadzić konfigurację zgodnie z opisem w pliku `librarytransfer.xml` . Wymagane jest również działające środowisko Managed File Transfer na dwóch komputerach z systemem IBM i . Konfiguracja musi składać się z agenta źródłowego działającego na pierwszym komputerze z systemem IBM i i agenta docelowego działającego na drugim komputerze z systemem IBM i . Dwa agenty muszą być w stanie komunikować się ze sobą.

IBM i Przykład librarytransfer składa się z następujących trzech plików:

- `librarytransfer.xml`
- `librarysave.sh` (< presrc> skrypt wykonywalny)
- `librayrestore.sh` (skrypt wykonywalny < postdst>)

Przykładowe pliki znajdują się w następującym katalogu: `/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer`

IBM i Aby uruchomić ten przykład, użytkownik musi wykonać następujące kroki:

1. Uruchom sesję Qshell. W oknie komend systemu IBM i wpisz: STRQSH
2. Przejdź do katalogu `bin` w następujący sposób:

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. Po zakończeniu wymaganej konfiguracji uruchom przykład za pomocą następującej komendy:

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer/librarytransfer.xml
```

## physicalfiletransfer (tylko platformaIBM i )

IBM i Przykład physicalfiletransfer demonstruje sposób użycia zadań Ant do przesłania źródłowego zbioru fizycznego lub zbioru bazy danych z biblioteki w jednym systemie IBM i do biblioteki w drugim systemie IBM i .

IBM i W przykładzie physicalfiletransfer używana jest obsługa rodzimych zbiorów składowania w systemie IBM i z predefiniowanymi zadaniami Ant dostępnymi w produkcie Managed File Transfer w celu przesłania kompletnych źródłowych zbiorów fizycznych i zbiorów bazy danych między dwoma systemami IBM i . W przykładzie użyto elementu zagnieżdżonego < presrc> w zadaniu Managed File Transfer filecopy do wywołania skryptu wykonywalnego `physicalfilesave.sh` w celu zapisania żądanego źródłowego pliku fizycznego lub pliku bazy danych z biblioteki w systemie agenta źródłowego do tymczasowego pliku zapisu. Plik zapisu jest przenoszony przez zadanie Ant filecopy do docelowego systemu agenta, w którym zagnieżdżony element < postdst> jest używany do wywołania skryptu wykonywalnego `physicalfilerestore.sh` , a następnie odtwarza obiekt pliku wewnątrz pliku zapisu do określonej biblioteki w systemie docelowym.

IBM i Przed uruchomieniem tego przykładu należy wykonać pewne czynności konfiguracyjne opisane w pliku `physicalfiletransfer.xml` . Wymagane jest również działające środowisko Managed File Transfer w dwóch systemach IBM i . Konfiguracja musi składać się z agenta źródłowego działającego w pierwszym systemie IBM i i agenta docelowego działającego w drugim systemie IBM i . Dwa agenty muszą być w stanie komunikować się ze sobą.

IBM i Przykład physicalfiletransfer składa się z trzech następujących plików:

- `physicalfiletransfer.xml`
- `physicalfilesave.sh` (< presrc> skrypt wykonywalny)
- `physicalfilerestore.sh` (skrypt wykonywalny < postdst>)

Przykładowe pliki znajdują się w następującym katalogu: `/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer`

IBM i Aby uruchomić ten przykład, użytkownik musi wykonać następujące kroki:

1. Uruchom sesję Qshell. W oknie komend systemu IBM i wpisz: STRQSH
2. Przejdź do katalogu `bin` w następujący sposób:

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. Po zakończeniu wymaganej konfiguracji uruchom przykład za pomocą następującej komendy:

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer/
physicalfiletransfer.xml
```

### limit czasu

Przykład timeout demonstruje sposób użycia zadań Ant do próby przesłania pliku i anulowania przesyłania, jeśli trwa ono dłużej niż określona wartość limitu czasu. Skrypt inicjuje przesyłanie plików przy użyciu zadania Managed File Transfer fte: filecopy . Wynik tego przesyłania jest odroczony. Skrypt używa zadania Managed File Transfer fte: awaitoutcome Ant , aby odczekać określoną liczbę sekund na zakończenie przesyłania. Jeśli przesyłanie nie zostanie zakończone w podanym czasie, do anulowania przesyłania plików używane jest zadanie Managed File Transfer fte: cancel Ant .

### vsamtransfer

z/OS

z/OS Przykład vsamtransfer demonstruje sposób użycia zadań Ant do przesłania z zestawu danych VSAM do innego zestawu danych VSAM za pomocą komendy Managed File Transfer. Produkt Managed File Transfer nie obsługuje obecnie przesyłania zestawów danych VSAM. Przykładowy skrypt usuwa z pamięci rekordy danych VSAM do sekwencyjnego zestawu danych za pomocą `presrc` elementów zagnieżdżonych wywołania programu w celu wywołania pliku wykonywalnego `datasetcopy.sh`. Skrypt używa zadania Managed File Transfer fte: filemove do przesłania sekwencyjnego zestawu danych z agenta źródłowego do agenta docelowego. Następnie skrypt używa `postdst` zagnieżdżonych elementów wywołania programu do wywołania skryptu `loadvsam.jcl` . Ten skrypt JCL ładuje przesłane rekordy zestawu danych do docelowego zestawu danych VSAM. W tym przykładzie użyto kodu JCL dla wywołania docelowego w celu zademonstrowania tej opcji języka. Ten sam wynik można również uzyskać za pomocą drugiego skryptu powłoki.

z/OS Ten przykład nie wymaga, aby źródłowy i docelowy zestaw danych był VSAM. Przykład działa dla dowolnego zestawu danych, jeśli źródłowy i docelowy zestaw danych są tego samego typu.

z/OS Aby ten przykład działał poprawnie, należy umieścić skrypt `datasetcopy.sh` w ścieżce komend agenta źródłowego i skrypt `loadvsam.jcl` w ścieżce komend agenta docelowego. Więcej informacji na temat ustawiania ścieżki komend agenta zawiera sekcja commandPath MFT właściwość.

### zip

Przykład zip składa się z dwóch skryptów: `zip.xml` i `zipfiles.xml`. W tym przykładzie przedstawiono sposób użycia `presrc` element zagnieżdżony w zadaniu Managed File Transfer fte: filemove w celu uruchomienia skryptu Ant przed wykonaniem operacji przenoszenia pliku. Skrypt `zipfiles.xml` wywołany przez zagnieżdżony element `presrc` w skrypcie `zip.xml` kompresuje zawartość katalogu. Skrypt `zip.xml` przesyła skompresowany plik. Ten przykład wymaga, aby skrypt `zipfiles.xml` Ant był obecny w ścieżce komend agenta źródłowego. Dzieje się tak, ponieważ skrypt `zipfiles.xml` Ant zawiera cel używany do kompresowania zawartości katalogu na agencie źródłowym. Więcej informacji na temat ustawiania ścieżki komend agenta zawiera sekcja commandPath MFT właściwość.

**Pojęcia pokrewne**

“Pierwsze kroki ze skryptami Ant w produkcie MFT” na stronie 1258
Użycie skryptów Ant z językiem Managed File Transfer umożliwia skoordynowanie złożonych operacji przesyłania plików z interpretowanego języka skryptowego.

**Odsyłacze pokrewne**

**`fteAnt`**: uruchamianie zadań Ant w programie MFT

## Dostosowywanie programu MFT za pomocą programów zewnętrznych

Funkcje programu Managed File Transfer można dostosować za pomocą własnych programów nazywanych procedurami zewnętrznymi.

**Ważne:** Kod w programie użytkownika nie jest obsługiwany przez produkt IBM, a wszelkie problemy z tym kodem muszą być wstępnie zbadane przez przedsiębiorstwo lub dostawcę, który udostępnił wyjście.

Managed File Transfer udostępnia punkty w kodzie, w których Managed File Transfer może przekazać sterowanie do napisanego programu (procedury zewnętrznej). Te punkty są nazywane punktami wyjścia użytkownika. System Managed File Transfer może wznowić sterowanie po zakończeniu pracy programu. Nie ma potrzeby korzystania z żadnych programów zewnętrznych, ale są one przydatne, gdy użytkownik chce rozszerzyć i dostosować funkcje systemu Managed File Transfer do konkretnych wymagań.

Istnieją dwa punkty podczas przetwarzania przesyłania plików, w których można wywołać program użytkownika w systemie źródłowym i dwa punkty podczas przetwarzania przesyłania plików, w których można wywołać program użytkownika w systemie docelowym. Poniższa tabela zawiera podsumowanie każdego z tych punktów wyjścia użytkownika i interfejsu Java , który należy zaimplementować w celu użycia punktów wyjścia.

*Tabela 187. Podsumowanie punktów wyjścia po stronie źródła i po stronie miejsca docelowego oraz interfejsów Java*

| **Punkt wyjścia** | **Interfejs Java do zaimplementowania** |
|---|---|
| **Punkty wyjścia po stronie źródła:** | |
| Przed rozpoczęciem przesyłania całego pliku | interfejsSourceTransferStartExit.java |
| Po zakończeniu przesyłania całego pliku | SourceTransferEndExit.java interfejs |
| **Punkty wyjścia po stronie miejsca docelowego:** | |
| Przed rozpoczęciem przesyłania całego pliku | DestinationTransferStartExit.java |
| Po zakończeniu przesyłania całego pliku | DestinationTransferEndExit.java |

Procedury zewnętrzne są wywoływane w następującej kolejności:

1. SourceTransferStartExit
2. DestinationTransferStartExit
3. DestinationTransferEndExit
4. SourceTransferEndExit

Zmiany wprowadzone przez wyjścia SourceTransferStartExit i DestinationTransferStartExit są propagowane jako dane wejściowe do kolejnych wyjść. Jeśli na przykład wyjście klasy SourceTransferStartExit modyfikuje metadane przesyłania, zmiany są odzwierciedlane w wejściowych metadanych przesyłania dla innych wyjść.

Procedury zewnętrzne i wywołania programów są wywoływane w następującej kolejności:

```
- 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.
```

**Uwagi:**

1. Skrypt **DestinationTransferEndExits** jest uruchamiany tylko wtedy, gdy operacja przesyłania zakończy się pomyślnie lub częściowo pomyślnie.
2. Skrypt **postDestinationCall** jest uruchamiany tylko wtedy, gdy operacja przesyłania zakończy się pomyślnie lub częściowo pomyślnie.
3. Komenda **SourceTransferEndExits** jest uruchamiana w przypadku pomyślnych, częściowo pomyślnych lub nieudanych operacji przesyłania.

4. Funkcja **postSourceCall** jest wywoływana tylko wtedy, gdy:
   - Przesyłanie nie zostało anulowane.
   - Wynik jest pomyślny lub częściowo pomyślny.
   - Wszystkie programy przesyłania w miejscu docelowym zostały pomyślnie uruchomione.

## Budowanie procedury zewnętrznej

Interfejsy służące do budowania procedury zewnętrznej są zawarte w pliku *MQ_INSTALL_DIRECTORY*/mqft/lib/com.ibm.wmqfte.exitroutines.api.jar. Podczas budowania wyjścia należy dołączyć ten plik .jar do ścieżki klasy. Aby uruchomić wyjście, wyodrębnij wyjście jako plik .jar i umieść ten plik .jar w katalogu zgodnie z opisem w poniższej sekcji.

## Położenia wyjścia użytkownika

Procedury użytkownika można przechowywać w dwóch możliwych miejscach:

- Katalog exits. W każdym katalogu agenta znajduje się katalog programów zewnętrznych. Na przykład: var\mqm\mqft\config\QM_JUPITER\agents\AGENT1\exits
- Aby określić alternatywne położenie, można ustawić właściwość ścieżki exitClass. Jeśli klasy wyjścia znajdują się zarówno w katalogu exits , jak i w ścieżce klasy ustawionej przez ścieżkę exitClass, pierwszeństwo mają klasy w katalogu exits , co oznacza, że jeśli w obu lokalizacjach istnieją klasy o takiej samej nazwie, pierwszeństwo mają klasy w katalogu exits .

## Konfigurowanie agenta do korzystania z programów zewnętrznych

Istnieją cztery właściwości agenta, które można ustawić w celu określenia procedur zewnętrznych wywołanych przez agenta. Te właściwości agenta to sourceTransferStartExitClasses, sourceTransferEndExitClasses, destinationTransferStartExitClassesi destinationTransferEndExitClasses. Informacje na temat korzystania z tych właściwości zawiera sekcja MFT Właściwości agenta dla procedur zewnętrznych.

## Uruchamianie programów zewnętrznych na agentach mostu protokołu

Gdy agent źródłowy wywołuje wyjście, przekazuje ono do wyjścia listę elementów źródłowych dla przesyłania. Dla zwykłych agentów jest to lista pełnych nazw plików. Ponieważ pliki powinny być lokalne (lub dostępne przez podłączenie), wyjście jest w stanie uzyskać do nich dostęp i zaszyfrować je.

Jednak w przypadku agenta mostu protokołu pozycje na liście mają następujący format:

```
"<file server identifier>:<fully-qualified file name of the file on the remote file server>"
```

Dla każdej pozycji na liście wyjście musi najpierw nawiązać połączenie z serwerem plików (przy użyciu protokołu FTP). Protokoły FTPS lub SFTP), pobierz plik, zaszyfruj go lokalnie, a następnie prześlij zaszyfrowany plik z powrotem na serwer plików.

## Uruchamianie programów zewnętrznych na agentach mostu Connect:Direct

Nie można uruchamiać programów zewnętrznych na agentach mostu Connect:Direct .

**Pojęcia pokrewne**

“Źródłowe i docelowe procedury zewnętrzne MFT” na stronie 1264

Metadane dla programów zewnętrznych MFT

Interfejsy Java dla programów zewnętrznych MFT

**Odsyłacze pokrewne**

“Włączanie zdalnego debugowania dla programów zewnętrznych MFT” na stronie 1268

Podczas tworzenia programów zewnętrznych można użyć debugera, aby ułatwić lokalizowanie problemów w kodzie.

“Przykładowa procedura zewnętrzna przesyłania kodu źródłowego w systemie MFT” na stronie 1269

“Przykładowa procedura zewnętrzna referencji mostu protokołu” na stronie 1270

Procedury zewnętrzne monitora zasobów MFT

MFT Właściwości agenta dla procedur zewnętrznych

# Źródłowe i docelowe procedury zewnętrzne MFT

## Separatory katalogów

Separatory katalogów w specyfikacjach plików źródłowych są zawsze reprezentowane za pomocą ukośników (/), niezależnie od tego, jak określono separatory katalogów w komendzie **fteCreateTransfer** lub w pliku IBM MQ Explorer. Należy to wziąć pod uwagę podczas pisania wyjścia. Na przykład, aby sprawdzić, czy następujący plik źródłowy istnieje: `c:\a\b.txt` i czy ten plik źródłowy został określony za pomocą komendy **fteCreateTransfer** lub komendy IBM MQ Explorer, należy zauważyć, że nazwa pliku jest rzeczywiście zapisana w postaci: `c:/a/b.txt` . Jeśli więc wyszukiwany jest oryginalny łańcuch `c:\a\b.txt`, nie zostanie znaleziony zgodny plik.

## Punkty wyjścia po stronie źródła

**Przed rozpoczęciem przesyłania całego pliku**

To wyjście jest wywoływane przez agenta źródłowego, gdy żądanie przesłania jest następne na liście oczekujących operacji przesyłania i operacja przesyłania ma się rozpocząć.

Przykładem użycia tego punktu wyjścia jest wysłanie plików w etapach do katalogu, do którego agent ma dostęp do odczytu i zapisu za pomocą komendy zewnętrznej lub zmiana nazwy plików w systemie docelowym.

Przekaż następujące argumenty do tego wyjścia:

- Nazwa agenta źródłowego
- Nazwa agenta docelowego
- Metadane środowiska
- Metadane przesyłania
- Specyfikacje plików (w tym metadane plików)

Dane zwracane przez to wyjście są następujące:

- Zaktualizowano metadane przesyłania. Pozycje można dodawać, modyfikować i usuwać.
- Zaktualizowana lista specyfikacji plików, która składa się z par nazwy pliku źródłowego i nazwy pliku docelowego. Pozycje można dodawać, modyfikować i usuwać
- Indykator określający, czy przesyłanie ma być kontynuowane
- Łańcuch, który ma zostać wstawiony do dziennika przesyłania.

Zaimplementuj interfejs SourceTransferStartExit.java w celu wywołania kodu wyjścia użytkownika w tym punkcie wyjścia.

**Po zakończeniu przesyłania całego pliku**

To wyjście jest wywoływane przez agenta źródłowego po zakończeniu przesyłania całego pliku.

Przykładem użycia tego punktu wyjścia jest wykonanie niektórych zadań zakończenia, takich jak wysłanie wiadomości e-mail lub wiadomości IBM MQ w celu oznaczenia, że przesyłanie zostało zakończone.

Przekaż następujące argumenty do tego wyjścia:

- Wynik wyjścia przesyłania
- Nazwa agenta źródłowego
- Nazwa agenta docelowego

- Metadane środowiska
- Metadane przesyłania
- Wyniki pliku

Dane zwracane przez to wyjście są następujące:

- Zaktualizowano łańcuch, który ma zostać wstawiony do dziennika przesyłania.

Zaimplementuj interfejs SourceTransferEndExit.java w celu wywołania kodu wyjścia użytkownika w tym punkcie wyjścia.

## Punkty wyjścia po stronie miejsca docelowego

**Przed rozpoczęciem przesyłania całego pliku**

Przykładem użycia tego punktu wyjścia jest sprawdzenie poprawności uprawnień w miejscu docelowym.

Przekaż następujące argumenty do tego wyjścia:

- Nazwa agenta źródłowego
- Nazwa agenta docelowego
- Metadane środowiska
- Metadane przesyłania
- Specyfikacja pliku

Dane zwracane przez to wyjście są następujące:

- Zaktualizowano zestaw nazw plików docelowych. Pozycje można modyfikować, ale nie można ich dodawać ani usuwać.
- Indykator określający, czy przesyłanie ma być kontynuowane
- Łańcuch, który ma zostać wstawiony do dziennika przesyłania.

Zaimplementuj interfejs DestinationTransferStartExit.java w celu wywołania kodu wyjścia użytkownika w tym punkcie wyjścia.

**Po zakończeniu przesyłania całego pliku**

Przykładem użycia tej procedury zewnętrznej jest uruchomienie procesu wsadowego korzystającego z przesłanych plików lub wysłanie wiadomości e-mail, jeśli przesyłanie nie powiodło się.

Przekaż następujące argumenty do tego wyjścia:

- Wynik wyjścia przesyłania
- Nazwa agenta źródłowego
- Nazwa agenta docelowego
- Metadane środowiska
- Metadane przesyłania
- Wyniki pliku

Dane zwracane przez to wyjście są następujące:

- Zaktualizowano łańcuch, który ma zostać wstawiony do dziennika przesyłania.

Zaimplementuj interfejs DestinationTransferEndExit.java w celu wywołania kodu wyjścia użytkownika w tym punkcie wyjścia.

**Pojęcia pokrewne**

Interfejsy Java dla programów zewnętrznych MFT

**Odsyłacze pokrewne**

“Włączanie zdalnego debugowania dla programów zewnętrznych MFT” na stronie 1268

Podczas tworzenia programów zewnętrznych można użyć debugera, aby ułatwić lokalizowanie problemów w kodzie.

“Przykładowa procedura zewnętrzna przesyłania kodu źródłowego w systemie MFT” na stronie 1269

Procedury zewnętrzne monitora zasobów MFT

## Korzystanie z procedur zewnętrznych we/wy przesyłania danych MFT

Procedury zewnętrzne we/wy przesyłania w systemie Managed File Transfer można użyć do skonfigurowania kodu niestandardowego w celu wykonania bazowej pracy we/wy systemu plików dla przesyłania w systemie Managed File Transfer .

Zwykle w przypadku przesyłania w systemie MFT agent wybiera jednego z wbudowanych dostawców we/wy do interakcji z odpowiednimi systemami plików dla przesyłania. Wbudowani dostawcy we/wy obsługują następujące typy systemów plików:

- Zwykłe systemy plików UNIX-type i Windows-type
- z/OS Sekwencyjne i partycjonowane zestawy danych z/OS (tylko w systemie z/OS )
- IBM i Rodzime zbiory składowania systemu IBM i (tylko w systemie IBM i )
- Kolejki produktu IBM MQ
- Zdalne serwery FTP i SFTP (tylko dla agentów mostu protokołu)
- Zdalne węzły Connect:Direct (tylko dla agentów mostu Connect:Direct )

W przypadku systemów plików, które nie są obsługiwane, lub w przypadku gdy wymagane jest niestandardowe zachowanie we/wy, można napisać procedurę zewnętrzną we/wy przesyłania.

Procedury zewnętrzne we/wy przesyłania używają istniejącej infrastruktury dla procedur zewnętrznych. Jednak te procedury zewnętrzne we/wy przesyłania różnią się od innych, ponieważ ich funkcje są wielokrotnie używane podczas przesyłania każdego pliku.

Użyj właściwości agenta IOExitClasses (w pliku `agent.properties` ), aby określić, które klasy wyjścia we/wy mają zostać załadowane. Każdą klasę wyjściową należy oddzielić przecinkiem, na przykład:

```
IOExitClasses=testExits.TestExit1,testExits.testExit2
```

Interfejsy Java dla procedur zewnętrznych we/wy przesyłania są następujące:

**Wyjście we/wy**
Główny punkt wejścia używany do określenia, czy wyjście we/wy jest używane. Ta instancja jest odpowiedzialna za tworzenie instancji IOExitPath .

Dla właściwości agenta IOExitClassesnależy podać tylko interfejs wyjścia we/wy IOExit.

**IOExitPath**
Reprezentuje interfejs abstrakcyjny, na przykład kontener danych lub znak wieloznaczny reprezentujący zestaw kontenerów danych. Nie można utworzyć instancji klasy, która implementuje ten interfejs. Interfejs umożliwia sprawdzenie ścieżki i wyświetlenie ścieżek pochodnych. Interfejsy IOExitResourcePath i IOExitWildcardPath rozszerzają interfejs IOExitPath.

**IOExitChannel**
Umożliwia odczyt lub zapis danych w zasobie IOExitPath .

**Kanał IOExitRecord**
Rozszerza interfejs IOExitChannel dla zorientowanych na rekordy zasobów IOExitPath , co umożliwia odczyt lub zapis danych do zasobu IOExitPath w wielokrotnościach rekordów.

**IOExitLock**
Reprezentuje blokadę zasobu IOExitPath dla dostępu współużytkowanego lub na wyłączność.

z/OS **IOExitRecordResourcePath**
Rozszerza interfejs ścieżki IOExitResourcew celu reprezentowania kontenera danych dla pliku zorientowanego na akta, na przykład zestawu danych z/OS . Za pomocą tego interfejsu można znaleźć dane i utworzyć instancje kanału IOExitRecorddla operacji odczytu lub zapisu.

**Ścieżka IOExitResource**
Rozszerza interfejs IOExitPath , aby reprezentował kontener danych, na przykład plik lub katalog. Za pomocą tego interfejsu można zlokalizować dane. Jeśli interfejs reprezentuje katalog, można użyć metody listPaths w celu zwrócenia listy ścieżek.

**IOExitWildcardŚcieżka**
Rozszerza interfejs IOExitPath o ścieżkę oznaczającą znak wieloznaczny. Tego interfejsu można użyć do dopasowania wielu ścieżek IOExitResource.

**IOExitProperties**
Określa właściwości określające sposób, w jaki program Managed File Transfer obsługuje ścieżkę IOExitPath dla niektórych aspektów operacji we/wy. Na przykład określa, czy mają być używane pliki pośrednie, czy zasób ma być ponownie odczytywany od początku, jeśli przesyłanie zostanie zrestartowane.

**Pojęcia pokrewne**
“Dostosowywanie programu MFT za pomocą programów zewnętrznych” na stronie 1261
Funkcje programu Managed File Transfer można dostosować za pomocą własnych programów nazywanych procedurami zewnętrznymi.

**Odsyłacze pokrewne**
Interfejs IOExit.java
Interfejs IOExitChannel.java
Interfejs IOExitLock.java
Interfejs IOExitPath.java
Interfejs IOExitProperties.java
Interfejs IOExitRecordChannel.java
z/OS Interfejs IOExitRecordResourcePath.java
Interfejs IOExitResourcePath.java
Interfejs IOExitWildcardPath.java
Plik MFT`agent.properties`

## IBM i Przykładowe procedury zewnętrzne MFT w systemie IBM i

Program Managed File Transfer udostępnia przykładowe procedury zewnętrzne specyficzne dla produktu IBM i wraz z instalacją. Przykłady znajdują się w katalogach *MQMFT_install_dir*`/samples/ioexit-IBMi` i *MQMFT_install_dir*`/samples/userexit-IBMi`.

**com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit**
Przykładowy program użytkownika com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit przesyła pliki w systemie plików QDLS w systemie IBM i. Po zainstalowaniu programu obsługi wyjścia wszystkie operacje przesyłania do plików, których nazwa rozpoczyna się od łańcucha `/QDLS` , automatycznie korzystają z tego programu.

Aby zainstalować to wyjście, wykonaj następujące kroki:

1. Skopiuj plik `com.ibm.wmqfte.samples.ibmi.ioexits.jar` z katalogu *WMQFTE_install_dir*`/samples/ioexit-IBMi` do katalogu `exits` agenta.
2. Dodaj właściwość com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit do właściwości IOExitClasses .
3. Zrestartuj agent.

**com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit**
Przykładowa procedura zewnętrzna com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit działa podobnie jak monitor plików MFT i automatycznie przesyła fizyczne elementy plików z biblioteki IBM i .

Aby uruchomić to wyjście, należy określić wartość w polu metadanych "library.qsys.monitor" (na przykład za pomocą parametru **-md** ). Ten parametr przyjmuje ścieżkę w stylu IFS do podzbioru zbioru i może zawierać znaki wieloznaczne zbioru i podzbioru. Na przykład /QSYS.LIB/FOO.LIB/BAR.FILE/*.MBR, /QSYS.LIB/FOO.LIB/*.FILE/BAR.MBR, /QSYS.LIB/FOO.LIB/*.ZBIÓR/*.MBR.

To przykładowe wyjście zawiera również opcjonalne pole metadanych "naming.scheme.qsys.monitor", którego można użyć do określenia schematu nazewnictwa używanego podczas przesyłania. Domyślnie to pole ma wartość "unix", co powoduje, że plik docelowy ma nazwę F00.MBR. Można również podać wartość "ibmi", aby użyć pliku IBM i FTP FILE.MEMBER , na przykład /QSYS.LIB/FOO.LIB/BAR.FILE/BAZ.MBR jest przesyłany jako plik BAR.BAZ.

Aby zainstalować to wyjście, wykonaj następujące kroki:

1. Skopiuj plik `com.ibm.wmqfte.samples.ibmi.userexits.jar` z katalogu *`WMQFTE_install_dir`*`/samples/userexit-IBMi` do katalogu `exits` agenta.
2. Dodaj właściwość com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit do klasyStartExitproduktu sourceTransferw pliku `agent.properties` .
3. Zrestartuj agent.

**com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit**
Przykładowy program użytkownika com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit usuwa pusty obiekt pliku, gdy element pliku źródłowego jest usuwany w ramach przesyłania. Ponieważ obiekty zbioru IBM i mogą zawierać wiele podzbiorów, obiekty zbioru są traktowane jak katalogi przez system MFT. Dlatego nie można wykonać operacji przenoszenia na obiekcie zbioru za pomocą MFT; operacje przenoszenia są obsługiwane tylko na poziomie podzbioru. W związku z tym podczas wykonywania operacji przenoszenia na podzbiorze pozostawiany jest pusty zbiór. Użyj tego przykładowego wyjścia, aby usunąć te puste pliki w ramach żądania przesyłania.

Jeśli dla metadanych "empty.file.delete" zostanie podana wartość "true" i zostanie przesłane ustawienie FTEFileMember, przykładowe wyjście usunie plik nadrzędny, jeśli plik jest pusty.

Aby zainstalować to wyjście, wykonaj następujące kroki:

1. Skopiuj plik com.ibm.wmqfte.samples.ibmi.userexits.jar z katalogu *`WMQFTE_install_dir`*`/samples/userexit-IBMi` do katalogu `exits` agenta.
2. Dodaj właściwość com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit do właściwościStartExitklasy sourceTransferw pliku `agent.properties` .
3. Zrestartuj agent.

**Odsyłacze pokrewne**
“Korzystanie z procedur zewnętrznych we/wy przesyłania danych MFT” na stronie 1266
Procedury zewnętrzne we/wy przesyłania w systemie Managed File Transfer można użyć do skonfigurowania kodu niestandardowego w celu wykonania bazowej pracy we/wy systemu plików dla przesyłania w systemie Managed File Transfer .

MFT Właściwości agenta dla procedur zewnętrznych

## Włączanie zdalnego debugowania dla programów zewnętrznych MFT

Podczas tworzenia programów zewnętrznych można użyć debugera, aby ułatwić lokalizowanie problemów w kodzie.

Ponieważ wyjścia są uruchamiane wewnątrz maszyny wirtualnej Java , na której działa agent, nie można używać obsługi debugowania bezpośredniego, która jest zwykle dołączona do zintegrowanego środowiska programistycznego. Można jednak włączyć zdalne debugowanie maszyny JVM, a następnie połączyć się z odpowiednim zdalnym debugerem.

Aby włączyć debugowanie zdalne, należy użyć standardowych parametrów maszyny JVM **-Xdebug** i **-Xrunjdwp**. Te właściwości są przekazywane do maszyny JVM, która uruchamia agenta, za pomocą zmiennej środowiskowej **BFG_JVM_PROPERTIES** . Na przykład w systemie AIX and Linux następujące komendy uruchamiają agenta i powodują nasłuchiwanie przez maszynę JVM połączeń debugera na porcie TCP 8765.

```
export BFG_JVM_PROPERTIES="-Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=8765"
fteStartAgent -F TEST_AGENT
```

Agent nie zostanie uruchomiony, dopóki debuger nie nawiąże połączenia. Zamiast komendy **export** należy użyć komendy **set** w systemie Windows .

Można również użyć innych metod komunikacji między debugerem i maszyną JVM. Na przykład maszyna JVM może otworzyć połączenie z debugerem zamiast odwrotnie lub użyć pamięci współużytkowanej zamiast protokołu TCP. Więcej informacji na ten temat zawiera dokumentacja architektury Java Platform Debugger Architecture .

Podczas uruchamiania agenta w zdalnym trybie debugowania należy użyć parametru **-F** (pierwszy plan).

### Korzystanie z debugera Eclipse

Poniższe kroki mają zastosowanie do możliwości debugowania zdalnego w środowisku programistycznym Eclipse . Można również użyć innych zdalnych debugerów, które są zgodne z JPDA.

1. Kliknij opcję **Uruchom** > **Otwórz okno dialogowe debugowania** (lub **Uruchom** > **Konfiguracje debugowania** lub **Uruchom** > **okno dialogowe debugowania** w zależności od wersji środowiska Eclipse).
2. Kliknij dwukrotnie opcję **Zdalna aplikacja Java** na liście typów konfiguracji, aby utworzyć konfigurację debugowania.
3. Wypełnij pola konfiguracyjne i zapisz konfigurację debugowania. Jeśli maszyna JVM agenta została już uruchomiona w trybie debugowania, można teraz nawiązać połączenie z maszyną JVM.

## Przykładowa procedura zewnętrzna przesyłania kodu źródłowego w systemie MFT

```
/*
 * 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";
    }
}
```

## Przykładowa procedura zewnętrzna referencji mostu protokołu

Informacje na temat korzystania z tej przykładowej procedury zewnętrznej zawiera sekcja Odwzorowywanie referencji dla serwera plików przy użyciu klas wyjścia.

```
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
    }
}
```

## Przykładowa procedura zewnętrzna właściwości mostu protokołu

Informacje na temat używania tej przykładowej procedury zewnętrznej zawiera sekcja ProtocolBridgePropertiesExit2: Wyszukiwanie właściwości serwera plików protokołu .

**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;
    }

}
```

## Sterowanie MFT przez umieszczanie komunikatów w kolejce komend agenta

Można napisać aplikację, która steruje systemem Managed File Transfer , umieszczając komunikaty w kolejkach komend agenta.

Można umieścić komunikat w kolejce komend agenta, aby zażądać, aby agent wykonał jedno z następujących działań:

- Tworzenie przesyłania plików
- Tworzenie zaplanowanego przesyłania plików
- Anuluj przesyłanie plików
- Anuluj zaplanowane przesyłanie plików
- Wywołaj komendę
- Tworzenie monitora
- Usuwanie monitora
- Zwróć komendę ping, aby wskazać, że agent jest aktywny

Aby zażądać, aby agent wykonał jedno z tych działań, komunikat musi być w formacie XML zgodnym z jednym z następujących schematów:

**FileTransfer.xsd**
Komunikaty w tym formacie mogą być używane do tworzenia przesyłania plików lub zaplanowanego przesyłania plików, do wywoływania komendy lub do anulowania przesyłania plików lub zaplanowanego przesyłania plików. Więcej informacji na ten temat zawiera sekcja Format komunikatu żądania przesyłania plików.

**Monitor.xsd**
Komunikaty w tym formacie mogą być używane do tworzenia lub usuwania monitora zasobów. Więcej informacji na ten temat zawiera sekcja Formaty komunikatów żądań monitora MFT.

**PingAgent.xsd**
Komunikaty w tym formacie mogą być używane do wysyłania pakietów ping do agenta w celu sprawdzenia, czy jest on aktywny. Więcej informacji na ten temat zawiera sekcja Format komunikatu żądania agenta Ping MFT.

Agent zwraca odpowiedź na komunikaty żądania. Komunikat odpowiedzi jest umieszczany w kolejce odpowiedzi zdefiniowanej w komunikacie żądania. Komunikat odpowiedzi jest w formacie XML zdefiniowanym przez następujący schemat:

**Reply.xsd**
Więcej informacji na ten temat zawiera sekcja Format komunikatu odpowiedzi agenta MFT.

# Tworzenie aplikacji dla składnika MQ Telemetry

Aplikacje telemetryczne integrują urządzenia zmysłu i sterowania z innymi źródłami informacji dostępnymi w Internecie i w przedsiębiorstwach.

Tworzenie aplikacji dla produktu MQ Telemetry przy użyciu wzorców projektowych, przykładów, przykładowych programów, pojęć związanych z programowaniem i informacji uzupełniających.

**Pojęcia pokrewne**

MQ Telemetry

Przypadki użycia danych telemetrycznych

**Zadania pokrewne**

Instalowanie produktu MQ Telemetry

administrowanieMQ Telemetry

Rozwiązywanie problemów z systemem MQ Telemetry

**Odsyłacze pokrewne**

Informacje uzupełniające dotyczące produktu MQ Telemetry

## IBM MQ Telemetry Transport programy przykładowe

Udostępniono przykładowe skrypty, które działają z przykładową aplikacją kliencką IBM MQ Telemetry Transport v3 (`mqttv3app.jar`). W systemie IBM MQ 8.0.0 i nowszych przykładowa aplikacja kliencka nie jest już dołączana do produktu MQ Telemetry. Był on częścią (nie jest już dostępny) IBM Messaging Telemetry Clients SupportPac. Podobne aplikacje przykładowe są nadal dostępne bezpłatnie w serwisie Eclipse Paho i MQTT.org.

Najbardziej aktualne informacje i pliki do pobrania można znaleźć w następujących zasobach:

- Projekt Eclipse Paho i produkt MQTT.org udostępniają bezpłatne pobieranie najnowszych klientów telemetrycznych oraz przykłady dla wielu języków programowania. Materiały dostępne w tych serwisach są przydatne przy rozbudowywaniu przykładowych programów do publikowania i subskrybowania przy użyciu protokołu IBM MQ Telemetry Transport, a także przy wprowadzaniu dodatkowych zabezpieczeń.
- Komponent IBM Messaging Telemetry Clients SupportPac nie jest już dostępny do pobrania. Zawartość ewentualnie wcześniej pobranej kopii jest następująca:
  - Wersja MA9B produktu IBM Messaging Telemetry Clients SupportPac obejmował skompilowaną przykładową aplikację (`mqttv3app.jar`) i powiązaną bibliotekę klienta (`mqttv3.jar`). Zostały one udostępnione w następujących katalogach:

    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`

- `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`

– W wersji MA9C tego pakietu SupportPac usunięto katalog i zawartość produktu `/SDK/`:

- Podano tylko źródło dla przykładowej aplikacji (`mqttv3app.jar`). Znajdowała się w następującym katalogu:

```
ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
```

- Nadal dostępna była skompilowana biblioteka kliencka. Znajdowała się w następującym katalogu:

```
ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
```

Jeśli nadal istnieje kopia pliku IBM Messaging Telemetry Clients SupportPac(już niedostępna), informacje na temat instalowania i uruchamiania przykładowej aplikacji zawiera sekcja Weryfikowanie instalacji produktu MQ Telemetry za pomocą wiersza komend.

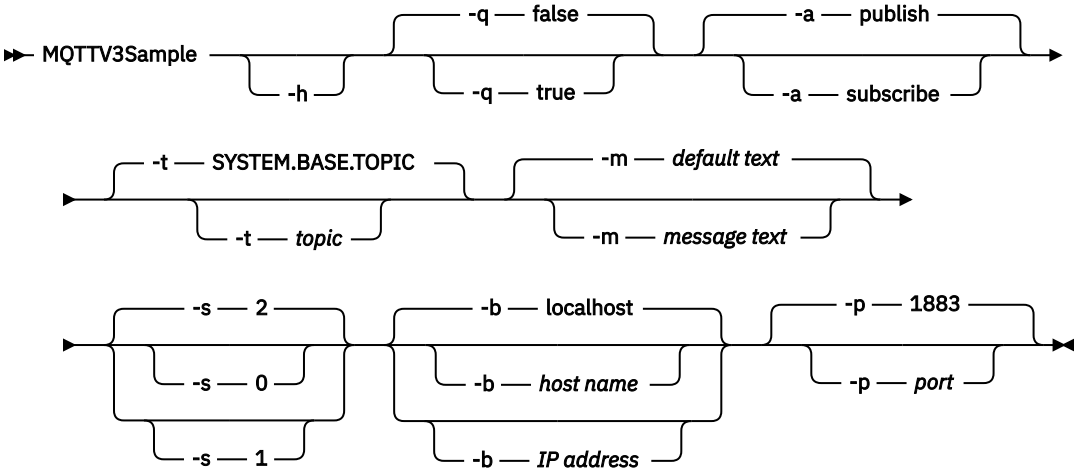
## Program MQTTV3Sample

Informacje uzupełniające o przykładowej składni i parametrach programu `MQTTV3Sample` .

### Przeznaczenie

Program `MQTTV3Sample` może być używany do publikowania komunikatu i subskrybowania tematu. Informacje na temat sposobu uzyskania tego programu przykładowego zawiera sekcja “IBM MQ Telemetry Transport programy przykładowe” na stronie 1276.

**MQTTV3Sample syntax**




### Parametry

**-h**
Wydrukuj ten tekst pomocy i wyjdź

**-q**
Ustaw tryb cichy, zamiast używać domyślnego trybu false.

**-a**
Ustaw publikowanie lub subskrybowanie zamiast domyślnego działania publikowania.

**-t**
Publikowanie lub subskrybowanie tematu zamiast publikowania lub subskrybowania tematu domyślnego

**-m**
Opublikuj tekst komunikatu zamiast wysyłać domyślny tekst publikacji "Hello z aplikacji MQTT v3 ".

**-s**
Zamiast używać domyślnej wartości QoS, 2, należy ustawić QoS .

**-b**
Nawiąż połączenie z tą nazwą hosta lub adresem IP zamiast z domyślną nazwą hosta (localhost).

**-p**
Użyj tego portu zamiast domyślnego (1883).

**Uruchom program MQTTV3Sample (V3Sample).**

Aby zasubskrybować temat w systemie Windows, należy użyć następującej komendy:

```
run MQTTV3Sample -a subscribe
```

Aby opublikować komunikat w systemie Windows, należy użyć komendy:

```
run MQTTV3Sample
```

## Pojęcia związane z programowaniem klienta MQTT

Pojęcia opisane w tej sekcji ułatwiają zrozumienie bibliotek klienta dla MQTT protocol. Pojęcia te uzupełniają dokumentację interfejsu API dołączoną do bibliotek klienta.

Najbardziej aktualne informacje i pliki do pobrania można znaleźć w następujących zasobach:

- Projekt Eclipse Paho i produkt MQTT.org udostępniają bezpłatne pobieranie najnowszych klientów telemetrycznych oraz przykłady dla wielu języków programowania. Materiały dostępne w tych serwisach są przydatne przy rozbudowywaniu przykładowych programów do publikowania i subskrybowania przy użyciu protokołu IBM MQ Telemetry Transport, a także przy wprowadzaniu dodatkowych zabezpieczeń.
- Komponent IBM Messaging Telemetry Clients SupportPac nie jest już dostępny do pobrania. Zawartość ewentualnie wcześniej pobranej kopii jest następująca:
  - Wersja MA9B produktu IBM Messaging Telemetry Clients SupportPac obejmował skompilowaną przykładową aplikację (`mqttv3app.jar`) i powiązaną bibliotekę klienta (`mqttv3.jar`). Zostały one udostępnione w następujących katalogach:
    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - W wersji MA9C tego pakietu SupportPac usunięto katalog i zawartość produktu `/SDK/`:
    - Podano tylko źródło dla przykładowej aplikacji (`mqttv3app.jar`). Znajdowała się w następującym katalogu:

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - Nadal dostępna była skompilowana biblioteka kliencka. Znajdowała się w następującym katalogu:

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

Aby utworzyć i uruchomić klienta MQTT , należy skopiować lub zainstalować te zasoby na urządzeniu klienckim. Nie ma potrzeby instalowania oddzielnego środowiska wykonawczego klienta.

Warunki licencji dla klientów są powiązane z serwerem, z którym są nawiązywane połączenia.

Biblioteki klienta MQTT są implementacjami odwołań do pliku MQTT protocol. Istnieje możliwość zaimplementowania własnych klientów w różnych językach odpowiednich dla różnych platform urządzeń. Patrz IBM MQ Telemetry Transport format i protokół (format and protocol).

Dokumentacja interfejsu API nie przyjmuje żadnych założeń co do tego, z którym serwerem MQTT jest połączony klient. Zachowanie klienta może się nieznacznie różnić w przypadku połączenia z różnymi serwerami. Poniższe opisy opisują zachowanie klienta po nawiązaniu połączenia z usługą telemetryczną IBM MQ .

## Wywołania zwrotne i synchronizacja w aplikacjach klienckich MQTT

Model programowania klienta MQTT wykorzystuje wątki w szerokim zakresie. Wątki oddzielają aplikację kliencką MQTT od opóźnień w przesyłaniu komunikatów do i z serwera. Publikacje, znaczniki dostarczania i zdarzenia utraty połączenia są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

### Wywołania zwrotne

**Uwaga:** Najnowsze zmiany w pliku `MqttCallback`można znaleźć w serwisie WWW Eclipse Paho . Na przykład interfejs `MqttCallback` jest zdefiniowany jako interfejs w wersji Paho klienta, a metody asynchroniczne są udostępniane przez klasę Paho `MqttAsyncClient` .

Interfejs `MqttCallback` ma trzy metody wywołania zwrotnego:

**`connectionLost(java.lang.Throwable cause)`**

Funkcja `connectionLost` jest wywoływana, gdy błąd komunikacji prowadzi do zerwania połączenia. Jest ona również wywoływana, jeśli serwer odrzuci połączenie w wyniku błędu na serwerze po nawiązaniu połączenia. Błędy serwera są rejestrowane w dzienniku błędów menedżera kolejek. Serwer porzuca połączenie z klientem, a klient wywołuje funkcję `MqttCallback.connectionLost`.

Jedynymi błędami zdalnymi zgłaszanymi jako wyjątki w tym samym wątku, co aplikacja kliencka, są wyjątki od `MqttClient.connect`. Błędy wykryte przez serwer po nawiązaniu połączenia są zgłaszane z powrotem do metody wywołania zwrotnego `MqttCallback.connectionLost` jako `throwables`.

Typowe błędy serwera, które powodują `connectionLost` , to błędy autoryzacji. Na przykład serwer pomiarowy próbuje publikować w temacie w imieniu klienta, który nie ma uprawnień do publikowania w temacie. Wszystko, co powoduje zwrócenie kodu stanu `MQCC_FAIL` do serwera pomiarowego, może spowodować usunięcie połączenia.

**`deliveryComplete(IMqttDeliveryToken token)`**

Funkcja `deliveryComplete` jest wywoływana przez klienta MQTT w celu przekazania tokenu dostarczania z powrotem do aplikacji klienckiej. Informacje na ten temat zawiera sekcja “Znaczniki dostarczania” na stronie 1285. Za pomocą znacznika dostarczania wywołanie zwrotne może uzyskać dostęp do opublikowanego komunikatu przy użyciu metody `token.getMessage`.

Gdy wywołanie zwrotne aplikacji zwraca sterowanie do klienta MQTT po wywołaniu przez metodę `deliveryComplete` , dostarczanie jest zakończone. Do momentu zakończenia dostarczania komunikaty z jakością usługi QoS 1 lub 2 są zachowywane przez klasę trwałości.

Wywołanie metody `deliveryComplete` jest punktem synchronizacji między aplikacją i klasą trwałości. Metoda `deliveryComplete` nigdy nie jest wywoływana dwukrotnie dla tego samego komunikatu.

Gdy wywołanie zwrotne aplikacji wraca z systemu `deliveryComplete` do klienta MQTT , klient wywołuje funkcję `MqttClientPersistence.remove` dla komunikatów z QoS 1 lub 2. Program `MqttClientPersistence.remove` usuwa lokalnie zapisaną kopię opublikowanego komunikatu.

Z perspektywy przetwarzania transakcji wywołanie metody `deliveryComplete` jest transakcją jednofazową, która zatwierdza dostarczanie. Jeśli przetwarzanie nie powiedzie się podczas wywołania zwrotnego, po zrestartowaniu klienta `MqttClientPersistence.remove` zostanie ponownie wywołana w celu usunięcia lokalnej kopii opublikowanego komunikatu. Wywołanie zwrotne nie jest ponownie wywoływane. Jeśli wywołanie zwrotne jest używane do przechowywania dziennika dostarczonych komunikatów, nie można zsynchronizować dziennika z klientem MQTT . Jeśli dziennik ma być przechowywany w sposób wiarygodny, należy zaktualizować dziennik w klasie `MqttClientPersistence` .

Znacznik dostarczania i komunikat są przywoływane przez główny wątek aplikacji i klient MQTT . Klient MQTT wyłuskuje obiekt `MqttMessage` po zakończeniu dostarczania i obiekt znacznika dostarczania po rozłączeniu się klienta. Obiekt `MqttMessage` może zostać wyczyszczony po zakończeniu dostarczania, jeśli aplikacja kliencka wyłuskuje ten obiekt. Znacznik dostarczania może zostać wyczyszczony po rozłączeniu sesji.

Atrybuty IMqttDeliveryToken i MqttMessage można uzyskać po opublikowaniu komunikatu. W przypadku próby ustawienia atrybutów MqttMessage po opublikowaniu komunikatu wynik jest niezdefiniowany.

Klient MQTT kontynuuje przetwarzanie potwierdzeń dostarczenia, jeśli klient ponownie nawiąże połączenie z poprzednią sesją z tym samym identyfikatorem ClientIdentifier ; zawiera sekcja “Wyczyść sesje” na stronie 1282. Aplikacja kliencka MQTT musi ustawić parametr MqttClient.CleanSession na wartość false dla poprzedniej sesji i ustawić go na wartość false w nowej sesji. Klient MQTT tworzy nowe znaczniki dostarczania i obiekty komunikatów w nowej sesji dla oczekujących dostaw. Odtwarza obiekty za pomocą klasy MqttClientPersistence . Jeśli klient aplikacji nadal ma odwołania do starych znaczników dostarczania i komunikatów, należy je wyłuskać. Wywołanie zwrotne aplikacji jest wywoływane w nowej sesji dla wszystkich dostaw zainicjowanych w poprzedniej sesji i zakończonych w tej sesji.

Wywołanie zwrotne aplikacji jest wywoływane po nawiązaniu połączenia przez klienta aplikacji po zakończeniu dostarczania oczekującego. Zanim klient aplikacji nawiąże połączenie, może pobrać oczekujące dostawy przy użyciu metody MqttClient.getPendingDeliveryTokens .

Należy zauważyć, że aplikacja kliencka pierwotnie utworzyła opublikowany obiekt komunikatu i jego tablicę bajtów ładunku. Klient MQTT odwołuje się do tych obiektów. Obiekt komunikatu zwrócony przez znacznik dostarczania w metodzie token.getMessage nie musi być tym samym obiektem komunikatu, który został utworzony przez klienta. Jeśli nowa instancja klienta MQTT ponownie tworzy znacznik dostarczania, klasa MqttClientPersistence ponownie tworzy obiekt MqttMessage . W celu zachowania spójności funkcja token.getMessage zwraca wartość null , jeśli parametr token.isCompleted ma wartość true, niezależnie od tego, czy obiekt komunikatu został utworzony przez klienta aplikacji, czy przez klasę MqttClientPersistence .

**messageArrived(String topic, MqttMessage message)**

Funkcja messageArrived jest wywoływana po nadejściu publikacji dla klienta, który jest zgodny z tematem subskrypcji. topic jest tematem publikacji, a nie filtrem subskrypcji. Jeśli filtr zawiera znaki wieloznaczne, te dwa znaki mogą być różne.

Jeśli temat jest zgodny z wieloma subskrypcjami utworzonymi przez klienta, klient otrzymuje wiele kopii publikacji. Jeśli klient publikuje w temacie, który również subskrybuje, otrzymuje kopię własnej publikacji.

Jeśli komunikat jest wysyłany z wartością QoS 1 lub 2, jest on przechowywany przez klasę MqttClientPersistence przed wywołaniem messageArrivedprzez klienta MQTT . Działanie komendy messageArrived jest podobne do deliveryComplete: jest ona wywoływana tylko raz dla publikacji, a lokalna kopia publikacji jest usuwana przez program MqttClientPersistence.remove po powrocie programu messageArrived do klienta MQTT . Klient MQTT usuwa swoje odwołania do tematu i komunikatu, gdy program messageArrived powraca do klienta MQTT . Obiekty tematu i komunikatu są usuwane, jeśli klient aplikacji nie zachował odwołania do tych obiektów.

## Wywołania zwrotne, wątki i synchronizacja aplikacji klienckiej

Klient MQTT wywołuje metodę wywołania zwrotnego w oddzielnym wątku do głównego wątku aplikacji. Aplikacja kliencka nie tworzy wątku dla wywołania zwrotnego, tylko jest tworzona przez klienta MQTT .

Klient MQTT synchronizuje metody wywołania zwrotnego. W danym momencie uruchamiana jest tylko jedna instancja metody wywołania zwrotnego. Synchronizacja ułatwia aktualizację obiektu, który określa, które publikacje zostały dostarczone. Jedna instancja serwera MqttCallback.deliveryComplete jest uruchamiana w danym momencie, dlatego można bezpiecznie zaktualizować wynik bez dalszej synchronizacji. W tym samym czasie pojawia się tylko jedna publikacja. Kod w metodzie messageArrived może aktualizować obiekt bez synchronizowania go. Jeśli odwołujesz się do obiektu lub obiektu, który jest aktualizowany, w innym wątku, zsynchronizuj obiekt lub obiekt.

Znacznik dostarczania udostępnia mechanizm synchronizacji między głównym wątkiem aplikacji i dostarczeniem publikacji. Metoda token.waitForCompletion oczekuje na zakończenie dostarczania konkretnej publikacji lub na zakończenie opcjonalnego limitu czasu. Do przetwarzania jednej publikacji w danym momencie można użyć programu token.waitForCompletion w następujący sposób.

Synchronizowanie z metodą `MqttCallback.deliveryComplete` . Tylko wtedy, gdy program `MqttCallback.deliveryComplete` powróci do klienta MQTT , `token.waitForCompletion` zostanie wznowiony. Za pomocą tego mechanizmu można zsynchronizować działający kod w produkcie `MqttCallback.deliveryComplete` przed uruchomieniem kodu w głównym wątku aplikacji.

Co zrobić, jeśli chcesz publikować bez czekania na dostarczenie każdej publikacji, ale chcesz mieć potwierdzenie, że wszystkie publikacje zostały dostarczone? W przypadku publikowania w pojedynczym wątku, ostatnia publikacja do wysłania jest również ostatnią dostarczoną publikacją.

## Synchronizacja żądań wysyłanych do serwera

Tabela 188 na stronie 1281 opisuje metody w kliencie MQTT Java , które wysyłają żądanie do serwera. Jeśli klient aplikacji nie ustawi nieokreślonego limitu czasu, klient nigdy nie czeka na serwer w nieskończoność. Jeśli klient zawiesi się, oznacza to problem z programowaniem aplikacji lub defekt w kliencie MQTT .

*Tabela 188. Zachowanie synchronizacji metod, które powodują wysyłanie żądań do serwera*

| Metoda | Synchronizacja | Limit czasu |
|---|---|---|
| `MqttClient.Connect` | Oczekuje na nawiązanie połączenia z serwerem. | Wartością domyślną jest 30 sekund lub wartość ustawiona przez parametr, a następnie zgłaszany jest wyjątek. |
| `MqttClient.Disconnect` | Czeka, aż klient MQTT zakończy pracę, którą musi wykonać, oraz na rozłączenie sesji TCP/IP. | |
| `MqttClient.Subscribe` | Oczekuje na zakończenie metody subskrypcji lub UnSubscribe . | |
| `MqttClient.UnSubscribe` | | |
| `MqttClient.Publish` | Powraca natychmiast do wątku aplikacji po przekazaniu żądania do klienta MQTT . | Brak. |
| `IMqttDeliveryToken.waitForCompletion` | Oczekuje na zwrócenie znacznika dostarczania. | Nieokreślony lub ustawiony jako parametr. |

**Pojęcia pokrewne**

Wyczyść sesje
Klient MQTT i usługa telemetryczna (MQXR) przechowują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania oraz "dokładnie raz" odbierania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z informacjami o stanie między sesjami lub bez nich. Przed nawiązaniem połączenia należy zmienić tryb czystej sesji, ustawiając opcję `MqttConnectOptions.cleanSession` .

Identyfikator klienta
Identyfikator klienta jest 23-bajtowym łańcuchem, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator może zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby mieć procedurę przydzielania identyfikatorów klienta oraz sposób konfigurowania klienta o wybranym identyfikatorze.

Znaczniki dostarczania

Ostatnia wola i publikacja testamentu
Jeśli połączenie klienta MQTT zostanie nieoczekiwanie zakończone, można skonfigurować program MQ Telemetry do wysyłania publikacji "last will and testament". Wstępnie zdefiniuj treść publikacji i temat, do którego ma ona zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Należy go utworzyć przed nawiązaniem połączenia z klientem.

Trwałość komunikatu w klientach MQTT
Komunikaty publikacji są trwałe, jeśli są wysyłane z jakością usługi "co najmniej raz"lub "dokładnie raz". Można zaimplementować własny mechanizm trwałości na kliencie lub użyć domyślnego mechanizmu trwałości, który jest dostarczany z klientem. Trwałość działa w obu kierunkach, dla publikacji wysyłanych do lub z klienta.

Publikacje
Publikacje są instancjami `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty MQTT mogą tworzyć publikacje wysyłane do IBM MQi subskrybować tematy w serwisie IBM MQ w celu odbierania publikacji.

Jakość usługi zapewniana przez klienta MQTT
Klient systemu MQTT udostępnia trzy elementy jakości usług na potrzeby dostarczania publikacji do produktu IBM MQ i klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie raz". Gdy klient MQTT wysyła żądanie do usługi IBM MQ w celu utworzenia subskrypcji, jest ono wysyłane z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną zachowaną publikację. Jeśli zostanie utworzona subskrypcja tematu, który ma zachowaną publikację, zostanie ona natychmiast przekazana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. W celu zarejestrowania zainteresowania wieloma tematami klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, w którym używane są znaki wieloznaczne. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest rozłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuchy tematów i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama jak w IBM MQ.

## Wyczyść sesje

Klient MQTT i usługa telemetryczna (MQXR) przechowują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania oraz "dokładnie raz" odbierania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z informacjami o stanie między sesjami lub bez nich. Przed nawiązaniem połączenia należy zmienić tryb czystej sesji, ustawiając opcję `MqttConnectOptions.cleanSession` .

Podczas nawiązywania połączenia z aplikacją kliencką MQTT przy użyciu metody `MqttClient.connect` klient identyfikuje połączenie przy użyciu identyfikatora klienta i adresu serwera. Serwer sprawdza, czy informacje o sesji zostały zapisane z poprzedniego połączenia z serwerem. Jeśli poprzednia sesja nadal istnieje, a `cleanSession=true`, to informacje o poprzedniej sesji na kliencie i serwerze są czyszczone. Jeśli ustawiona jest wartość `cleanSession=false` , poprzednia sesja jest wznawiana. Jeśli poprzednia sesja nie istnieje, uruchamiana jest nowa sesja.

**Uwaga:** Administrator IBM MQ może wymusić zamknięcie otwartej sesji i usunięcie wszystkich informacji o sesji. Jeśli klient ponownie otworzy sesję z `cleanSession=false`, zostanie uruchomiona nowa sesja.

## Publikacje

Jeśli przed nawiązaniem połączenia z klientem zostanie użyta domyślna wartość `MqttConnectOptions`lub dla parametru `MqttConnectOptions.cleanSession` zostanie ustawiona wartość `true` , wszystkie oczekujące operacje dostarczania publikacji dla klienta zostaną usunięte, gdy klient nawiąże połączenie.

Ustawienie czyszczenia sesji nie ma wpływu na publikacje wysyłane za pomocą programu `QoS=0`. W systemach `QoS=1` i `QoS=2`użycie parametru `cleanSession=true` może spowodować utratę publikacji.

## Subskrypcje

Jeśli przed nawiązaniem połączenia z klientem zostanie użyty domyślny serwer `MqttConnectOptions`lub parametr `MqttConnectOptions.cleanSession` zostanie ustawiony na wartość `true` , wszystkie stare subskrypcje dla klienta zostaną usunięte podczas nawiązywania połączenia przez klient. Wszystkie nowe subskrypcje dokonane przez klienta podczas sesji zostaną usunięte po rozłączeniu.

Jeśli parametr `MqttConnectOptions.cleanSession` zostanie ustawiony na wartość `false` przed nawiązaniem połączenia, wszystkie subskrypcje tworzone przez klienta zostaną dodane do wszystkich subskrypcji, które istniały dla klienta przed nawiązaniem połączenia. Wszystkie subskrypcje pozostaną aktywne po rozłączeniu połączenia z klientem.

Atrybut `cleanSession` można także rozpatrywać jako atrybut modalny, jeśli chodzi o jego wpływ na subskrypcje. W domyślnym trybie atrybutu `cleanSession=true` klient tworzy subskrypcje i odbiera publikacje tylko w zasięgu sesji. W alternatywnym trybie atrybutu `cleanSession=false` subskrypcje są trwałe. Można nawiązywać połączenia z klientem i je rozłączać, a jego subskrypcje pozostają aktywne. Po ponownym nawiązaniu połączenia z klientem odbiera on wszystkie niedostarczone publikacje. Po nawiązaniu połączenia klient może modyfikować zestaw subskrypcji aktywnych na jego potrzeby.

Przed nawiązaniem połączenia należy ustawić tryb `cleanSession` . Tryb ten będzie obowiązywać przez całą sesję. Aby zmienić ustawienie trybu, należy rozłączyć połączenie z klientem, a następnie ponownie je nawiązać. Jeśli tryby zostaną zmienione z użycia trybu `cleanSession=false` na tryb `cleanSession=true`, wszystkie wcześniejsze subskrypcje dla klienta i wszystkie publikacje, które nie zostały odebrane, zostaną odrzucone.

**Pojęcia pokrewne**

Wywołania zwrotne i synchronizacja w aplikacjach klienckich MQTT
Model programowania klienta MQTT wykorzystuje wątki w szerokim zakresie. Wątki oddzielają aplikację kliencką MQTT od opóźnień w przesyłaniu komunikatów do i z serwera. Publikacje, znaczniki dostarczania i zdarzenia utraty połączenia są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Identyfikator klienta
Identyfikator klienta jest 23-bajtowym łańcuchem, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator może zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby mieć procedurę przydzielania identyfikatorów klienta oraz sposób konfigurowania klienta o wybranym identyfikatorze.

Znaczniki dostarczania

Ostatnia wola i publikacja testamentu
Jeśli połączenie klienta MQTT zostanie nieoczekiwanie zakończone, można skonfigurować program MQ Telemetry do wysyłania publikacji "last will and testament". Wstępnie zdefiniuj treść publikacji i temat, do którego ma ona zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Należy go utworzyć przed nawiązaniem połączenia z klientem.

Trwałość komunikatu w klientach MQTT
Komunikaty publikacji są trwałe, jeśli są wysyłane z jakością usługi "co najmniej raz"lub "dokładnie raz". Można zaimplementować własny mechanizm trwałości na kliencie lub użyć domyślnego mechanizmu trwałości, który jest dostarczany z klientem. Trwałość działa w obu kierunkach, dla publikacji wysyłanych do lub z klienta.

Publikacje
Publikacje są instancjami `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty MQTT mogą tworzyć publikacje wysyłane do IBM MQi subskrybować tematy w serwisie IBM MQ w celu odbierania publikacji.

Jakość usługi zapewniana przez klienta MQTT
Klient systemu MQTT udostępnia trzy elementy jakości usług na potrzeby dostarczania publikacji do produktu IBM MQ i klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie raz". Gdy klient MQTT

wysyła żądanie do usługi IBM MQ w celu utworzenia subskrypcji, jest ono wysyłane z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną zachowaną publikację. Jeśli zostanie utworzona subskrypcja tematu, który ma zachowaną publikację, zostanie ona natychmiast przekazana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. W celu zarejestrowania zainteresowania wieloma tematami klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, w którym używane są znaki wieloznaczne. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest rozłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuchy tematów i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama jak w IBM MQ.

## Identyfikator klienta

Identyfikator klienta jest 23-bajtowym łańcuchem, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator może zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby mieć procedurę przydzielania identyfikatorów klienta oraz sposób konfigurowania klienta o wybranym identyfikatorze.

Identyfikator klienta jest używany podczas administrowania systemem MQTT . Z potencjalnie setki tysięcy klientów do administrowania, trzeba być w stanie szybko zidentyfikować konkretnego klienta. Załóżmy na przykład, że urządzenie działa nieprawidłowo i użytkownik jest powiadamiany, na przykład przez klienta dzwoniącego do stanowiska pomocy. Klient musi być w stanie zidentyfikować urządzenie, a użytkownik musi być w stanie skorelować tę identyfikację z serwerem, który jest zwykle połączony z klientem.

Podczas przeglądania połączeń klienckich systemu MQTT każde połączenie jest oznaczone identyfikatorem klienta. Aby zdecydować, w jaki sposób najlepiej odwzorować ten identyfikator na urządzenie i serwer, zadaj sobie następujące pytania:

- Czy wygodnie jest obsługiwać i używać bazy danych, która odwzorowuje każde urządzenie na identyfikator klienta i serwer?
- Czy nazwa urządzenia może identyfikować serwer, do którego jest ono podłączone?
- Czy potrzebujesz tabeli wyszukiwania, która odwzorowuje identyfikator klienta na urządzenie fizyczne?
- Czy identyfikator klienta identyfikuje konkretne urządzenie, użytkownika lub aplikację działającą na kliencie?
- Jeśli klient zastąpi uszkodzone urządzenie nowym, czy nowe urządzenie ma taki sam identyfikator jak stare urządzenie, czy też przydzielany jest nowy identyfikator? (W przypadku zmiany urządzenia fizycznego i zachowania tego samego identyfikatora, zaległe publikacje i aktywne subskrypcje są automatycznie przesyłane do nowego urządzenia).

Potrzebny jest również system, który zapewni unikalność identyfikatorów klienta oraz niezawodny proces ustawiania identyfikatora na kliencie. Jeśli urządzenie klienckie jest "czarną skrzynką", bez interfejsu użytkownika, można utworzyć urządzenie z identyfikatorem klienta lub może istnieć proces instalacji i konfiguracji oprogramowania, który skonfiguruje urządzenie przed jego aktywowaniem.

Aby identyfikator był krótki i unikalny, można utworzyć identyfikator klienta na podstawie 48-bitowego adresu MAC urządzenia. Jeśli wielkość transmisji nie jest krytyczna, można użyć pozostałych 17 bajtów, aby ułatwić administrowanie adresem.

**Pojęcia pokrewne**

Wywołania zwrotne i synchronizacja w aplikacjach klienckich MQTT
Model programowania klienta MQTT wykorzystuje wątki w szerokim zakresie. Wątki oddzielają aplikację kliencką MQTT od opóźnień w przesyłaniu komunikatów do i z serwera. Publikacje, znaczniki dostarczania

i zdarzenia utraty połączenia są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Wyczyść sesje
Klient MQTT i usługa telemetryczna (MQXR) przechowują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania oraz "dokładnie raz" odbierania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z informacjami o stanie między sesjami lub bez nich. Przed nawiązaniem połączenia należy zmienić tryb czystej sesji, ustawiając opcję `MqttConnectOptions.cleanSession` .

Znaczniki dostarczania

Ostatnia wola i publikacja testamentu
Jeśli połączenie klienta MQTT zostanie nieoczekiwanie zakończone, można skonfigurować program MQ Telemetry do wysyłania publikacji "last will and testament". Wstępnie zdefiniuj treść publikacji i temat, do którego ma ona zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Należy go utworzyć przed nawiązaniem połączenia z klientem.

Trwałość komunikatu w klientach MQTT
Komunikaty publikacji są trwałe, jeśli są wysyłane z jakością usługi "co najmniej raz"lub "dokładnie raz". Można zaimplementować własny mechanizm trwałości na kliencie lub użyć domyślnego mechanizmu trwałości, który jest dostarczany z klientem. Trwałość działa w obu kierunkach, dla publikacji wysyłanych do lub z klienta.

Publikacje
Publikacje są instancjami `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty MQTT mogą tworzyć publikacje wysyłane do IBM MQi subskrybować tematy w serwisie IBM MQ w celu odbierania publikacji.

Jakość usługi zapewniana przez klienta MQTT
Klient systemu MQTT udostępnia trzy elementy jakości usług na potrzeby dostarczania publikacji do produktu IBM MQ i klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie raz". Gdy klient MQTT wysyła żądanie do usługi IBM MQ w celu utworzenia subskrypcji, jest ono wysyłane z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną zachowaną publikację. Jeśli zostanie utworzona subskrypcja tematu, który ma zachowaną publikację, zostanie ona natychmiast przekazana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. W celu zarejestrowania zainteresowania wieloma tematami klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, w którym używane są znaki wieloznaczne. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest rozłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuchy tematów i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama jak w IBM MQ.

## Znaczniki dostarczania

Podczas publikowania przez klienta tematu tworzony jest nowy znacznik dostarczania. Znacznik dostarczania służy do monitorowania dostarczania publikacji lub do blokowania aplikacji klienckiej do momentu zakończenia dostarczania.

Token jest obiektem `MqttDeliveryToken` . Jest on tworzony przez wywołanie metody MqttTopic.publish () i zachowywany przez klienta MQTT do momentu rozłączenia sesji klienta i zakończenia dostarczania.

Normalne użycie tokenu polega na sprawdzeniu, czy dostarczanie zostało zakończone. Zablokuj aplikację kliencką do momentu zakończenia dostarczania, wywołując funkcję `token.waitForCompletion`przy użyciu zwróconego znacznika. Alternatywnie można podać procedurę obsługi `MqttCallBack` . Gdy klient MQTT otrzyma wszystkie potwierdzenia, których oczekuje w ramach dostarczania publikacji, wywołuje funkcję `MqttCallBack.deliveryComplete` przekazującą znacznik dostarczania jako parametr.

Do momentu zakończenia dostarczania można sprawdzić publikację przy użyciu zwróconego znacznika dostarczania, wywołując metodę `token.getMessage`.

## Zakończone dostawy

Zakończenie dostaw jest asynchroniczne i zależy od jakości usług związanych z publikacją.

**Co najwyżej raz**

`QoS=0`
Dostawa jest zakończona natychmiast po zwrocie z `MqttTopic.publish`. Funkcja `MqttCallback.deliveryComplete` jest wywoływana natychmiast.

**Co najmniej raz**

`QoS=1`
Dostarczanie jest zakończone po odebraniu potwierdzenia publikacji z menedżera kolejek. Funkcja `MqttCallback.deliveryComplete` jest wywoływana po odebraniu potwierdzenia. Komunikat może zostać dostarczony więcej niż raz przed wywołaniem `MqttCallback.deliveryComplete` , jeśli komunikacja jest powolna lub zawodna.

**Dokładnie jeden raz**

`QoS=2`
Dostarczanie jest zakończone, gdy klient otrzyma komunikat o zakończeniu publikowania publikacji dla subskrybentów. Funkcja `MqttCallback.deliveryComplete` jest wywoływana natychmiast po odebraniu komunikatu publikacji. Nie czeka na komunikat o zakończeniu.

W rzadkich przypadkach aplikacja kliencka może normalnie nie powrócić do klienta MQTT z systemu `MqttCallback.deliveryComplete` . Wiadomo, że dostarczanie zostało zakończone, ponieważ wywołano `MqttCallback.deliveryComplete` . Jeśli klient zrestartuje tę samą sesję, program `MqttCallback.deliveryComplete` nie zostanie ponownie wywołany.

## Niedokończone dostawy

Jeśli dostarczanie nie zostanie zakończone po rozłączeniu sesji klienta, można ponownie połączyć się z klientem i zakończyć dostarczanie. Dostarczenie komunikatu można zakończyć tylko wtedy, gdy komunikat został opublikowany w sesji z atrybutem `MqttConnectionOptions` ustawionym na wartość `false`.

Utwórz klienta przy użyciu tego samego identyfikatora klienta i adresu serwera, a następnie nawiąż połączenie, ponownie ustawiając atrybut `cleanSession` `MqttConnectionOptions` na wartość `false` . Jeśli właściwość `cleanSession` zostanie ustawiona na wartość `true`, oczekujące znaczniki dostarczania zostaną odrzucone.

Istnieje możliwość sprawdzenia, czy istnieją oczekujące dostawy, wywołując metodę `MqttClient.getPendingDeliveryTokens`. Przed nawiązaniem połączenia z klientem można wywołać funkcję `MqttClient.getPendingDeliveryTokens` .

**Pojęcia pokrewne**

Wywołania zwrotne i synchronizacja w aplikacjach klienckich MQTT
Model programowania klienta MQTT wykorzystuje wątki w szerokim zakresie. Wątki oddzielają aplikację kliencką MQTT od opóźnień w przesyłaniu komunikatów do i z serwera. Publikacje, znaczniki dostarczania i zdarzenia utraty połączenia są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Wyczyść sesje
Klient MQTT i usługa telemetryczna (MQXR) przechowują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania oraz "dokładnie raz" odbierania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z informacjami o stanie między sesjami lub bez nich. Przed nawiązaniem połączenia należy zmienić tryb czystej sesji, ustawiając opcję `MqttConnectOptions.cleanSession` .

Identyfikator klienta

Identyfikator klienta jest 23-bajtowym łańcuchem, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator może zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby mieć procedurę przydzielania identyfikatorów klienta oraz sposób konfigurowania klienta o wybranym identyfikatorze.

Ostatnia wola i publikacja testamentu
Jeśli połączenie klienta MQTT zostanie nieoczekiwanie zakończone, można skonfigurować program MQ Telemetry do wysyłania publikacji "last will and testament". Wstępnie zdefiniuj treść publikacji i temat, do którego ma ona zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Należy go utworzyć przed nawiązaniem połączenia z klientem.

Trwałość komunikatu w klientach MQTT
Komunikaty publikacji są trwałe, jeśli są wysyłane z jakością usługi "co najmniej raz"lub "dokładnie raz". Można zaimplementować własny mechanizm trwałości na kliencie lub użyć domyślnego mechanizmu trwałości, który jest dostarczany z klientem. Trwałość działa w obu kierunkach, dla publikacji wysyłanych do lub z klienta.

Publikacje
Publikacje są instancjami `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty MQTT mogą tworzyć publikacje wysyłane do IBM MQi subskrybować tematy w serwisie IBM MQ w celu odbierania publikacji.

Jakość usługi zapewniana przez klienta MQTT
Klient systemu MQTT udostępnia trzy elementy jakości usług na potrzeby dostarczania publikacji do produktu IBM MQ i klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie raz". Gdy klient MQTT wysyła żądanie do usługi IBM MQ w celu utworzenia subskrypcji, jest ono wysyłane z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną zachowaną publikację. Jeśli zostanie utworzona subskrypcja tematu, który ma zachowaną publikację, zostanie ona natychmiast przekazana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. W celu zarejestrowania zainteresowania wieloma tematami klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, w którym używane są znaki wieloznaczne. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest rozłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuchy tematów i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama jak w IBM MQ.

## Ostatnia wola i publikacja testamentu

Jeśli połączenie klienta MQTT zostanie nieoczekiwanie zakończone, można skonfigurować program MQ Telemetry do wysyłania publikacji "last will and testament". Wstępnie zdefiniuj treść publikacji i temat, do którego ma ona zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Należy go utworzyć przed nawiązaniem połączenia z klientem.

Utwórz temat dla ostatniej woli i testamentu. Można utworzyć temat, taki jak `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`.

Skonfiguruj "ostatnią wolę i testament" za pomocą metody `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)`.

Rozważ utworzenie znacznika czasu w komunikacie `lastWillPayload` . Dołącz inne informacje o kliencie, które pomagają zidentyfikować klienta i okoliczności połączenia. Przekaż obiekt `MqttConnectionOptions` do konstruktora `MqttClient` .

Ustaw parametr `lastWillQos` na wartość 1 lub 2, aby komunikat był trwały w produkcie IBM MQi zagwarantować jego dostarczenie. Aby zachować informacje o ostatnio utraconym połączeniu, należy ustawić parametr `lastWillRetained` na wartość `true`.

Publikacja "last will and testament" jest wysyłana do subskrybentów, jeśli połączenie zostanie nieoczekiwanie zakończone. Jest on wysyłany, jeśli połączenie zostanie zakończone bez wywoływania metody `MqttClient.disconnect` przez klienta.

Aby monitorować połączenia, należy uzupełnić publikację "ostatniej woli i testamentu" o inne publikacje w celu rejestrowania połączeń i programowanych rozłączeń.

**Pojęcia pokrewne**

Wywołania zwrotne i synchronizacja w aplikacjach klienckich MQTT
Model programowania klienta MQTT wykorzystuje wątki w szerokim zakresie. Wątki oddzielają aplikację kliencką MQTT od opóźnień w przesyłaniu komunikatów do i z serwera. Publikacje, znaczniki dostarczania i zdarzenia utraty połączenia są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Wyczyść sesje
Klient MQTT i usługa telemetryczna (MQXR) przechowują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania oraz "dokładnie raz" odbierania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z informacjami o stanie między sesjami lub bez nich. Przed nawiązaniem połączenia należy zmienić tryb czystej sesji, ustawiając opcję `MqttConnectOptions.cleanSession` .

Identyfikator klienta
Identyfikator klienta jest 23-bajtowym łańcuchem, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator może zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby mieć procedurę przydzielania identyfikatorów klienta oraz sposób konfigurowania klienta o wybranym identyfikatorze.

Znaczniki dostarczania

Trwałość komunikatu w klientach MQTT
Komunikaty publikacji są trwałe, jeśli są wysyłane z jakością usługi "co najmniej raz"lub "dokładnie raz". Można zaimplementować własny mechanizm trwałości na kliencie lub użyć domyślnego mechanizmu trwałości, który jest dostarczany z klientem. Trwałość działa w obu kierunkach, dla publikacji wysyłanych do lub z klienta.

Publikacje
Publikacje są instancjami `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty MQTT mogą tworzyć publikacje wysyłane do IBM MQi subskrybować tematy w serwisie IBM MQ w celu odbierania publikacji.

Jakość usługi zapewniana przez klienta MQTT
Klient systemu MQTT udostępnia trzy elementy jakości usług na potrzeby dostarczania publikacji do produktu IBM MQ i klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie raz". Gdy klient MQTT wysyła żądanie do usługi IBM MQ w celu utworzenia subskrypcji, jest ono wysyłane z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną zachowaną publikację. Jeśli zostanie utworzona subskrypcja tematu, który ma zachowaną publikację, zostanie ona natychmiast przekazana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. W celu zarejestrowania zainteresowania wieloma tematami klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, w którym używane są znaki wieloznaczne. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest rozłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT

Łańcuchy tematów i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama jak w IBM MQ.

## Trwałość komunikatu w klientach MQTT

Komunikaty publikacji są trwałe, jeśli są wysyłane z jakością usługi "co najmniej raz"lub "dokładnie raz". Można zaimplementować własny mechanizm trwałości na kliencie lub użyć domyślnego mechanizmu trwałości, który jest dostarczany z klientem. Trwałość działa w obu kierunkach, dla publikacji wysyłanych do lub z klienta.

W produkcie MQTTtrwałość komunikatu ma dwa aspekty: sposób przesyłania komunikatu oraz to, czy jest on umieszczany w kolejce w produkcie IBM MQ jako komunikat trwały.

1. Klient MQTT łączy trwałość komunikatu z jakością usługi. W zależności od wybranej jakości usługi komunikat jest zachowywany. Trwałość komunikatu jest niezbędna do zaimplementowania wymaganej jakości usługi.

   Jeśli zostanie podana wartość "co najwyżej raz"( QoS=0), klient odrzuci komunikat natychmiast po jego opublikowaniu. Jeśli przetwarzanie komunikatu nie powiedzie się, komunikat nie zostanie ponownie wysłany. Nawet jeśli klient pozostaje aktywny, komunikat nie jest ponownie wysyłany. Zachowanie komunikatów QoS=0 jest takie samo jak szybkich komunikatów nietrwałych IBM MQ .
   Jeśli komunikat jest publikowany przez klienta z wartością QoS 1 lub 2, jest on trwały. Komunikat jest zapisywany lokalnie i usuwany z klienta tylko wtedy, gdy nie jest już potrzebny do zagwarantowania "co najmniej raz", QoS=1lub "dokładnie raz", QoS=2dostarczenia.
2. Jeśli komunikat jest oznaczony jako QoS 1 lub 2, jest umieszczany w kolejce IBM MQ jako komunikat trwały. Jeśli zostanie oznaczony jako QoS=0, zostanie umieszczony w kolejce IBM MQ jako komunikat nietrwały. W przypadku IBM MQ nietrwałych komunikatów są przesyłane między menedżerami kolejek "dokładnie raz", chyba że kanał komunikatów ma atrybut NPMSPEED ustawiony na wartość FAST.

Trwała publikacja jest przechowywana na kliencie, dopóki nie zostanie odebrana przez aplikację kliencką. W przypadku produktu QoS=2publikacja jest usuwana z klienta, gdy wywołanie zwrotne aplikacji zwraca sterowanie. W przypadku systemu QoS=1 aplikacja może ponownie otrzymać publikację, jeśli wystąpi awaria. W przypadku QoS=0wywołanie zwrotne odbiera publikację nie więcej niż raz. Publikacja może nie zostać odebrany, jeśli wystąpi awaria lub klient zostanie rozłączony w momencie publikacji.

Po zasubskrybowaniu tematu można zmniejszyć poziom jakości usługi ( QoS ), z którym subskrybent odbiera komunikaty, aby były zgodne z jego możliwościami trwałości. Publikacje utworzone na wyższym poziomie QoS są wysyłane z najwyższą wartością QoS , której zażądał subskrybent.

## Zapisywanie komunikatów

Implementacja pamięci masowej danych na małych urządzeniach jest bardzo zróżnicowana. Model tymczasowego zapisywania trwałych komunikatów w pamięci masowej zarządzanej przez klienta MQTT może być zbyt powolny lub wymagać zbyt dużej ilości pamięci masowej. W przypadku urządzeń przenośnych system operacyjny urządzenia przenośnego może udostępniać usługę pamięci masowej, która jest idealna dla komunikatów MQTT .

Aby zapewnić elastyczność w spełnianiu ograniczeń małych urządzeń, klient MQTT ma dwa interfejsy trwałości. Interfejsy definiują operacje, które są zaangażowane w zapisywanie trwałych komunikatów. Interfejsy są opisane w dokumentacji interfejsu API dla MQTT client for Java. Odsyłacze do dokumentacji interfejsu API klienta dla bibliotek klienta MQTT zawiera sekcja Skorowidz programistyczny klienta MQTT. Interfejsy można zaimplementować w taki sposób, aby pasowały do urządzenia. Klient MQTT , który działa w systemie Java SE, ma domyślną implementację interfejsów przechowujących komunikaty trwałe w systemie plików. Używa pakietu `java.io` .

## Klasy trwałości

**`MqttClientPersistence`**
Przekaż instancję implementacji `MqttClientPersistence` do klienta MQTT jako parametr konstruktora `MqttClient` . Jeśli parametr `MqttClientPersistence` zostanie pominięty

w konstruktorze `MqttClient` , klient MQTT zapisuje trwałe komunikaty przy użyciu klasy `MqttDefaultFilePersistence`.

**`MqttPersistable`**
`MqttClientPersistence` pobiera i umieszcza obiekty `MqttPersistable` przy użyciu klucza pamięci masowej. Jeśli nie jest używany produkt `MqttDefaultFilePersistence`, należy udostępnić implementację języka `MqttPersistable` , a także implementację języka `MqttClientPersistence` .

**`MqttDefaultFilePersistence`**

Klient MQTT udostępnia klasę `MqttDefaultFilePersistence` . W przypadku tworzenia instancji klasy `MqttDefaultFilePersistence` w aplikacji klienckiej można udostępnić katalog do przechowywania trwałych komunikatów jako parametr konstruktora `MqttDefaultFilePersistence` .

Alternatywnie klient MQTT może utworzyć instancję pliku `MqttDefaultFilePersistence` i umieścić pliki w następującym katalogu domyślnym:

*`client identifier`* `-tcp` *`hostname portnumber`*

Następujące znaki są usuwane z łańcucha nazwy katalogu:

`"\", "\\", "/", ":"¡" "`

Ścieżka do katalogu jest wartością właściwości systemowej `rcp.data`; jeśli parametr `rcp.data` nie jest ustawiony, ścieżka jest wartością właściwości systemowej `usr.data`, gdzie

- `rcp.data` to właściwość powiązana z instalacją środowiska OSGi lub platformy Eclipse Rich Client Platform (RCP).
- `usr.data` jest katalogiem, w którym uruchomiono komendę Java uruchamiającą aplikację.

**Pojęcia pokrewne**

Wywołania zwrotne i synchronizacja w aplikacjach klienckich MQTT
Model programowania klienta MQTT wykorzystuje wątki w szerokim zakresie. Wątki oddzielają aplikację kliencką MQTT od opóźnień w przesyłaniu komunikatów do i z serwera. Publikacje, znaczniki dostarczania i zdarzenia utraty połączenia są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Wyczyść sesje
Klient MQTT i usługa telemetryczna (MQXR) przechowują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania oraz "dokładnie raz" odbierania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z informacjami o stanie między sesjami lub bez nich. Przed nawiązaniem połączenia należy zmienić tryb czystej sesji, ustawiając opcję `MqttConnectOptions.cleanSession` .

Identyfikator klienta
Identyfikator klienta jest 23-bajtowym łańcuchem, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator może zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby mieć procedurę przydzielania identyfikatorów klienta oraz sposób konfigurowania klienta o wybranym identyfikatorze.

Znaczniki dostarczania

Ostatnia wola i publikacja testamentu
Jeśli połączenie klienta MQTT zostanie nieoczekiwanie zakończone, można skonfigurować program MQ Telemetry do wysyłania publikacji "last will and testament". Wstępnie zdefiniuj treść publikacji i temat, do którego ma ona zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Należy go utworzyć przed nawiązaniem połączenia z klientem.

Publikacje
Publikacje są instancjami `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty MQTT mogą tworzyć publikacje wysyłane do IBM MQi subskrybować tematy w serwisie IBM MQ w celu odbierania publikacji.

Jakość usługi zapewniana przez klienta MQTT
Klient systemu MQTT udostępnia trzy elementy jakości usług na potrzeby dostarczania publikacji do produktu IBM MQ i klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie raz". Gdy klient MQTT wysyła żądanie do usługi IBM MQ w celu utworzenia subskrypcji, jest ono wysyłane z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną zachowaną publikację. Jeśli zostanie utworzona subskrypcja tematu, który ma zachowaną publikację, zostanie ona natychmiast przekazana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. W celu zarejestrowania zainteresowania wieloma tematami klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, w którym używane są znaki wieloznaczne. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest rozłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuchy tematów i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama jak w IBM MQ.

## Publikacje

Publikacje są instancjami `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty MQTT mogą tworzyć publikacje wysyłane do IBM MQi subskrybować tematy w serwisie IBM MQ w celu odbierania publikacji.

Ładunkiem `MqttMessage` jest tablica bajtów. Mają na celu zachowanie jak najmniejszych wiadomości. Maksymalna długość komunikatu dozwolona przez MQTT protocol wynosi 250 MB.

Zazwyczaj program kliencki systemu MQTT używa języka `java.lang.String` lub `java.lang.StringBuffer` do manipulowania treścią komunikatów. Dla wygody klasa `MqttMessage` ma metodę `toString` , która przekształca swój ładunek w łańcuch. Aby utworzyć ładunek tablicy bajtów na podstawie elementu `java.lang.String` lub `java.lang.StringBuffer`, należy użyć metody `getBytes` .

Metoda `getBytes` przekształca łańcuch w domyślny zestaw znaków dla platformy. Zwykle domyślnym zestawem znaków jest `UTF-8`. Publikacje MQTT , które zawierają tylko tekst, są zwykle kodowane w języku `UTF-8`. Aby nadpisać domyślny zestaw znaków, należy użyć metody `getBytes("UTF8")` .

W systemie IBM MQpublikacja MQTT jest odbierana jako komunikat `jms-bytes` . Komunikat zawiera folder `MQRFH2` zawierający `<mqtt>`i folder `<mqps>` . Folder `<mqtt>` zawiera `clientId`, `msgId`i `qos`, ale ta treść może ulec zmianie w przyszłości.

`MqttMessage` ma trzy dodatkowe atrybuty: jakość usługi, to, czy jest zachowana, oraz to, czy jest duplikatem. Flaga duplikatu jest ustawiana tylko wtedy, gdy jakość usługi ma wartość "co najmniej raz" lub "dokładnie raz". Jeśli komunikat został wysłany wcześniej i nie został wystarczająco szybko potwierdzony przez klienta MQTT , komunikat jest wysyłany ponownie, ze zduplikowanym atrybutem ustawionym na wartość `true`.

### Publikowanie

Aby utworzyć publikację w aplikacji klienckiej MQTT , należy utworzyć plik `MqttMessage`. Ustaw jego ładunek, jakość usługi i informację o tym, czy jest on zachowywany, i wywołaj metodę `MqttTopic.publish(MqttMessage message)` . Zwracana jest metoda `MqttDeliveryToken` , a zakończenie publikacji jest asynchroniczne.

Alternatywnie klient MQTT może utworzyć tymczasowy obiekt komunikatu na podstawie parametrów metody `MqttTopic.publish(byte [] payload, int qos, boolean retained)` podczas tworzenia publikacji.

Jeśli publikacja ma jakość usługi "co najmniej raz" lub "dokładnie raz", QoS=1 lub QoS=2, klient MQTT wywołuje interfejs MqttClientPersistence . Wywołuje funkcję MqttClientPersistence w celu zapisania komunikatu przed zwróceniem znacznika dostarczania do aplikacji.

Aplikacja może zablokować komunikat do momentu dostarczenia go do serwera przy użyciu metody MqttDeliveryToken.waitForCompletion . Alternatywnie aplikacja może kontynuować działanie bez blokowania. Aby sprawdzić, czy publikacje są dostarczane bez blokowania, należy zarejestrować instancję klasy wywołania zwrotnego, która implementuje interfejs MqttCallback na kliencie MQTT . Klient MQTT wywołuje metodę MqttCallback.deliveryComplete natychmiast po dostarczeniu publikacji. W zależności od jakości usługi, dostawa może być prawie natychmiastowa w przypadku produktu QoS=0lub może zająć trochę czasu w przypadku produktu QoS=2.

Użyj metody MqttDeliveryToken.isComplete , aby odpytać, czy dostarczanie zostało zakończone. Jeśli wartością parametru MqttDeliveryToken.isComplete jest false, można wywołać funkcję MqttDeliveryToken.getMessage , aby pobrać treść komunikatu. Jeśli wynikiem wywołania metody MqttDeliveryToken.isComplete jest true, komunikat został odrzucony i wywołanie metody MqttDeliveryToken.getMessage spowodowałoby zgłoszenie wyjątku wskaźnika pustego. Nie ma wbudowanej synchronizacji między produktami MqttDeliveryToken.getMessage i MqttDeliveryToken.isComplete.

Jeśli klient rozłączy się przed odebraniem wszystkich oczekujących znaczników dostarczania, nowa instancja klienta może wysłać zapytanie do oczekujących znaczników dostarczania przed nawiązaniem połączenia. Dopóki klient nie nawiąże połączenia, nie są wykonywane żadne nowe dostawy i można bezpiecznie wywołać program MqttDeliveryToken.getMessage. Metoda MqttDeliveryToken.getMessage służy do określania, które publikacje nie zostały dostarczone. Tokeny oczekujące na dostarczenie są odrzucane, jeśli zostanie nawiązane połączenie z bazą danych MqttConnectOptions.cleanSession ustawioną na wartość domyślną true.

## subskrypcja

Menedżer kolejek jest odpowiedzialny za tworzenie publikacji wysyłanych do subskrybenta produktu MQTT . Menedżer kolejek sprawdza, czy filtr tematów w subskrypcji utworzonej przez klient MQTT jest zgodny z łańcuchem tematu w publikacji. Dopasowanie może być dokładne lub może zawierać znaki wieloznaczne. Przed przekazaniem publikacji przez menedżer kolejek do subskrybenta menedżer kolejek sprawdza atrybuty tematu powiązane z publikacją. Jest to zgodne z procedurą wyszukiwania opisaną w sekcji Subskrybowanie przy użyciu łańcucha tematu zawierającego znaki wieloznaczne , która określa, czy obiekt tematu administracyjnego nadaje użytkownikowi uprawnienie do subskrybowania.

Gdy klient MQTT odbiera publikację z jakością usługi co najmniej raz, wywołuje metodę MqttCallback.messageArrived w celu przetworzenia tej publikacji. Jeśli jakość usługi publikacji jest "dokładnie raz", QoS=2, klient MQTT wywołuje interfejs MqttClientPersistence w celu zapisania komunikatu po jego odebraniu. Następnie wywołuje funkcję MqttCallback.messageArrived.

**Pojęcia pokrewne**

Wywołania zwrotne i synchronizacja w aplikacjach klienckich MQTT
Model programowania klienta MQTT wykorzystuje wątki w szerokim zakresie. Wątki oddzielają aplikację kliencką MQTT od opóźnień w przesyłaniu komunikatów do i z serwera. Publikacje, znaczniki dostarczania i zdarzenia utraty połączenia są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs MqttCallback.

Wyczyść sesje
Klient MQTT i usługa telemetryczna (MQXR) przechowują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania oraz "dokładnie raz" odbierania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z informacjami o stanie między sesjami lub bez nich. Przed nawiązaniem połączenia należy zmienić tryb czystej sesji, ustawiając opcję MqttConnectOptions.cleanSession .

Identyfikator klienta
Identyfikator klienta jest 23-bajtowym łańcuchem, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator może zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć

dowolnego łańcucha identyfikacyjnego. Ważne jest, aby mieć procedurę przydzielania identyfikatorów klienta oraz sposób konfigurowania klienta o wybranym identyfikatorze.

Znaczniki dostarczania

Ostatnia wola i publikacja testamentu
Jeśli połączenie klienta MQTT zostanie nieoczekiwanie zakończone, można skonfigurować program MQ Telemetry do wysyłania publikacji "last will and testament". Wstępnie zdefiniuj treść publikacji i temat, do którego ma ona zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Należy go utworzyć przed nawiązaniem połączenia z klientem.

Trwałość komunikatu w klientach MQTT
Komunikaty publikacji są trwałe, jeśli są wysyłane z jakością usługi "co najmniej raz"lub "dokładnie raz". Można zaimplementować własny mechanizm trwałości na kliencie lub użyć domyślnego mechanizmu trwałości, który jest dostarczany z klientem. Trwałość działa w obu kierunkach, dla publikacji wysyłanych do lub z klienta.

Jakość usługi zapewniana przez klienta MQTT
Klient systemu MQTT udostępnia trzy elementy jakości usług na potrzeby dostarczania publikacji do produktu IBM MQ i klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie raz". Gdy klient MQTT wysyła żądanie do usługi IBM MQ w celu utworzenia subskrypcji, jest ono wysyłane z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną zachowaną publikację. Jeśli zostanie utworzona subskrypcja tematu, który ma zachowaną publikację, zostanie ona natychmiast przekazana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. W celu zarejestrowania zainteresowania wieloma tematami klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, w którym używane są znaki wieloznaczne. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest rozłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuchy tematów i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama jak w IBM MQ.

## Jakość usługi zapewniana przez klienta MQTT

Klient systemu MQTT udostępnia trzy elementy jakości usług na potrzeby dostarczania publikacji do produktu IBM MQ i klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie raz". Gdy klient MQTT wysyła żądanie do usługi IBM MQ w celu utworzenia subskrypcji, jest ono wysyłane z jakością usługi co najmniej raz.

Jakość usługi publikacji jest atrybutem `MqttMessage`. Jest ona ustawiana za pomocą metody `MqttMessage.setQos`.

Metoda `MqttClient.subscribe` może obniżyć jakość usługi zastosowaną do publikacji wysyłanych do klienta w danym temacie. Jakość usługi publikacji przekazanej subskrybentowi może być inna niż jakość usługi publikacji. Niższa z tych dwóch wartości jest używana do przekazywania publikacji.

**Co najwyżej raz**
`QoS=0`

Komunikat jest dostarczany nie więcej niż jeden raz lub nie jest dostarczany w ogóle. Dostarczenie komunikatu za pośrednictwem sieci nie jest potwierdzane.

Komunikat nie jest przechowywany. Komunikat może zostać utracony, jeśli klient zostanie rozłączony lub nastąpi awaria serwera.

`QoS=0` jest najszybszym trybem przesyłania. Czasami nazywa się to "ogniem i zapominaniem".

MQTT protocol nie wymaga, aby serwery przekazywał publikacje w serwisie `QoS=0` do klienta. Jeśli klient zostanie rozłączony w momencie odbierania publikacji przez serwer, publikacja może zostać odrzucona, w zależności od serwera. Usługa telemetryczna (MQXR) nie usuwa komunikatów

wysłanych z produktem QoS=0. Są one przechowywane jako komunikaty nietrwałe i są usuwane tylko wtedy, gdy menedżer kolejek zostanie zatrzymany.

**Co najmniej raz**

QoS=1

QoS=1 jest domyślnym trybem przesyłania.

Komunikat jest zawsze dostarczany co najmniej raz. Jeśli nadawca nie otrzyma potwierdzenia, komunikat jest wysyłany ponownie z ustawioną flagą DUP do momentu odebrania potwierdzenia. W rezultacie odbiornik może być wielokrotnie wysyłany do tego samego komunikatu i może być przetwarzany wielokrotnie.

Komunikat musi być przechowywany lokalnie u nadawcy i odbiorcy do czasu jego przetworzenia.

Komunikat jest usuwany z odbiornika po przetworzeniu komunikatu. Jeśli odbiorcą jest broker, komunikat jest publikowany w subskrybentach. Jeśli odbiorcą jest klient, komunikat jest dostarczany do aplikacji subskrybenta. Po usunięciu komunikatu odbiorca wysyła potwierdzenie do nadawcy.

Komunikat jest usuwany z nadawcy po otrzymaniu potwierdzenia od odbiorcy.

**Dokładnie jeden raz**

QoS=2

Komunikat jest zawsze dostarczany dokładnie raz.

Komunikat musi być przechowywany lokalnie u nadawcy i odbiorcy do czasu jego przetworzenia.

QoS=2 jest najbezpieczniejszym, ale najwolniejszym trybem przesyłania. Zanim komunikat zostanie usunięty z nadawcy, pobiera on co najmniej dwie pary transmisji między nadawcą i odbiorcą. Komunikat może być przetwarzany w odbiorniku po pierwszej transmisji.

W pierwszej parze transmisji nadawca przesyła komunikat i otrzymuje potwierdzenie od odbiorcy, że zapisał komunikat. Jeśli nadawca nie otrzyma potwierdzenia, komunikat jest wysyłany ponownie z ustawioną flagą DUP do momentu odebrania potwierdzenia.

W drugiej parze transmisji nadawca informuje odbiorcę, że może zakończyć przetwarzanie komunikatu "PUBREL". Jeśli nadawca nie otrzyma potwierdzenia komunikatu "PUBREL" , komunikat "PUBREL" zostanie wysłany ponownie do momentu odebrania potwierdzenia. Nadawca usuwa zapisany komunikat po odebraniu potwierdzenia do komunikatu "PUBREL" .

Odbiorca może przetwarzać komunikat w pierwszej lub drugiej fazie, pod warunkiem, że nie przetwarza go ponownie. Jeśli odbiorca jest brokerem, publikuje komunikat do subskrybentów. Jeśli odbiorcą jest klient, komunikat jest dostarczany do aplikacji subskrybenta. Odbiorca wysyła komunikat o zakończeniu z powrotem do nadawcy, który zakończył przetwarzanie komunikatu.

**Pojęcia pokrewne**

Wywołania zwrotne i synchronizacja w aplikacjach klienckich MQTT

Model programowania klienta MQTT wykorzystuje wątki w szerokim zakresie. Wątki oddzielają aplikację kliencką MQTT od opóźnień w przesyłaniu komunikatów do i z serwera. Publikacje, znaczniki dostarczania i zdarzenia utraty połączenia są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Wyczyść sesje

Klient MQTT i usługa telemetryczna (MQXR) przechowują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania oraz "dokładnie raz" odbierania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z informacjami o stanie między sesjami lub bez nich. Przed nawiązaniem połączenia należy zmienić tryb czystej sesji, ustawiając opcję `MqttConnectOptions.cleanSession` .

Identyfikator klienta

Identyfikator klienta jest 23-bajtowym łańcuchem, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator może zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby mieć procedurę przydzielania identyfikatorów klienta oraz sposób konfigurowania klienta o wybranym identyfikatorze.

Znaczniki dostarczania

Ostatnia wola i publikacja testamentu
Jeśli połączenie klienta MQTT zostanie nieoczekiwanie zakończone, można skonfigurować program MQ Telemetry do wysyłania publikacji "last will and testament". Wstępnie zdefiniuj treść publikacji i temat, do którego ma ona zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Należy go utworzyć przed nawiązaniem połączenia z klientem.

Trwałość komunikatu w klientach MQTT
Komunikaty publikacji są trwałe, jeśli są wysyłane z jakością usługi "co najmniej raz"lub "dokładnie raz". Można zaimplementować własny mechanizm trwałości na kliencie lub użyć domyślnego mechanizmu trwałości, który jest dostarczany z klientem. Trwałość działa w obu kierunkach, dla publikacji wysyłanych do lub z klienta.

Publikacje
Publikacje są instancjami `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty MQTT mogą tworzyć publikacje wysyłane do IBM MQi subskrybować tematy w serwisie IBM MQ w celu odbierania publikacji.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną zachowaną publikację. Jeśli zostanie utworzona subskrypcja tematu, który ma zachowaną publikację, zostanie ona natychmiast przekazana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. W celu zarejestrowania zainteresowania wieloma tematami klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, w którym używane są znaki wieloznaczne. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest rozłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuchy tematów i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama jak w IBM MQ.

## Zachowane publikacje i klienty MQTT

Temat może mieć jedną i tylko jedną zachowaną publikację. Jeśli zostanie utworzona subskrypcja tematu, który ma zachowaną publikację, zostanie ona natychmiast przekazana do użytkownika.

Metoda `MqttMessage.setRetained` umożliwia określenie, czy publikacja dotycząca tematu ma być zachowywana.

Po utworzeniu lub zaktualizowaniu zachowanej publikacji należy wysłać ją z wartością `QoS` 1 lub 2. Jeśli zostanie ona wysłana z wartością `QoS` równą 0, IBM MQ utworzy nietrwałą zachowaną publikację. Publikacja nie jest zachowywana, jeśli menedżer kolejek zostanie zatrzymany.

Opublikowanie niezachowanej publikacji w temacie, który ma zachowaną publikację, nie ma wpływu na zachowaną publikację. Bieżący subskrybenci otrzymują nową publikację. Nowi subskrybenci otrzymują zachowaną publikację jako pierwsi, a następnie otrzymują nowe publikacje.

Zachowaną publikację można wykorzystać do zarejestrowania najnowszej wartości pomiaru. Nowi subskrybenci tematu natychmiast otrzymują najnowszą wartość pomiaru. Jeśli od ostatniej subskrypcji tematu publikacji subskrybent nie dokonał żadnych nowych pomiarów, a subskrybent ponownie subskrybuje ten temat, otrzyma on ponownie najnowszą zachowaną publikację w tym temacie.

Aby usunąć zachowaną publikację, dostępne są dwie opcje:

- Uruchom komendę **`CLEAR TOPICSTR`** MQSC.
- Utwórz zachowaną publikację o zerowej długości. Zgodnie ze specyfikacją MQTT 3.1.1 , jeśli zachowany komunikat o zerowej długości jest publikowany w temacie, każdy zachowany komunikat dla tego tematu jest czyszczony.

**Pojęcia pokrewne**
Wywołania zwrotne i synchronizacja w aplikacjach klienckich MQTT
Model programowania klienta MQTT wykorzystuje wątki w szerokim zakresie. Wątki oddzielają aplikację kliencką MQTT od opóźnień w przesyłaniu komunikatów do i z serwera. Publikacje, znaczniki dostarczania

i zdarzenia utraty połączenia są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Wyczyść sesje
Klient MQTT i usługa telemetryczna (MQXR) przechowują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania oraz "dokładnie raz" odbierania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z informacjami o stanie między sesjami lub bez nich. Przed nawiązaniem połączenia należy zmienić tryb czystej sesji, ustawiając opcję `MqttConnectOptions.cleanSession` .

Identyfikator klienta
Identyfikator klienta jest 23-bajtowym łańcuchem, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator może zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby mieć procedurę przydzielania identyfikatorów klienta oraz sposób konfigurowania klienta o wybranym identyfikatorze.

Znaczniki dostarczania

Ostatnia wola i publikacja testamentu
Jeśli połączenie klienta MQTT zostanie nieoczekiwanie zakończone, można skonfigurować program MQ Telemetry do wysyłania publikacji "last will and testament". Wstępnie zdefiniuj treść publikacji i temat, do którego ma ona zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Należy go utworzyć przed nawiązaniem połączenia z klientem.

Trwałość komunikatu w klientach MQTT
Komunikaty publikacji są trwałe, jeśli są wysyłane z jakością usługi "co najmniej raz"lub "dokładnie raz". Można zaimplementować własny mechanizm trwałości na kliencie lub użyć domyślnego mechanizmu trwałości, który jest dostarczany z klientem. Trwałość działa w obu kierunkach, dla publikacji wysyłanych do lub z klienta.

Publikacje
Publikacje są instancjami `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty MQTT mogą tworzyć publikacje wysyłane do IBM MQi subskrybować tematy w serwisie IBM MQ w celu odbierania publikacji.

Jakość usługi zapewniana przez klienta MQTT
Klient systemu MQTT udostępnia trzy elementy jakości usług na potrzeby dostarczania publikacji do produktu IBM MQ i klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie raz". Gdy klient MQTT wysyła żądanie do usługi IBM MQ w celu utworzenia subskrypcji, jest ono wysyłane z jakością usługi co najmniej raz.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. W celu zarejestrowania zainteresowania wieloma tematami klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, w którym używane są znaki wieloznaczne. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest rozłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuchy tematów i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama jak w IBM MQ.

## Subskrypcje

Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. W celu zarejestrowania zainteresowania wieloma tematami klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, w którym używane są znaki wieloznaczne. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest rozłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Subskrypcje można tworzyć przy użyciu metod `MqttClient.subscribe` , przekazując jeden lub więcej filtrów tematów i parametrów jakości usługi. Parametr jakości usługi ustawia maksymalną jakość usługi, którą subskrybent jest przygotowany do użycia w celu odebrania komunikatu. Komunikaty wysyłane do

tego klienta nie mogą być dostarczane z wyższą jakością usługi. Jakość usługi jest ustawiana na niższą z wartości oryginalnych w momencie publikowania komunikatu i na poziomie określonym dla subskrypcji. Domyślna jakość usługi dla odbierania komunikatów to `QoS=1`, co najmniej raz.

Samo żądanie subskrypcji jest wysyłane z produktem `QoS=1`.

Publikacje są odbierane przez subskrybenta, gdy klient MQTT wywołuje metodę `MqttCallback.messageArrived` . Metoda `messageArrived` przekazuje również łańcuch tematu, za pomocą którego komunikat został opublikowany w subskrybencie.

Korzystając z metod `MqttClient.unsubscribe` , można usunąć subskrypcję, zestaw lub subskrypcje.

Komenda IBM MQ może usunąć subskrypcję. Wyświetlanie listy subskrypcji przy użyciu programu IBM MQ Explorerlub komend **runmqsc** lub PCF. Zostaną nazwane wszystkie subskrypcje klienta MQTT. Otrzymują one nazwę w postaci: *`ClientIdentifier:Topic name`*

Jeśli przed nawiązaniem połączenia z klientem zostanie użyty domyślny serwer `MqttConnectOptions`lub parametr `MqttConnectOptions.cleanSession` zostanie ustawiony na wartość `true` , wszystkie stare subskrypcje dla klienta zostaną usunięte podczas nawiązywania połączenia przez klient. Wszystkie nowe subskrypcje dokonane przez klienta podczas sesji zostaną usunięte po rozłączeniu.

Jeśli parametr `MqttConnectOptions.cleanSession` zostanie ustawiony na wartość `false` przed nawiązaniem połączenia, wszystkie subskrypcje tworzone przez klienta zostaną dodane do wszystkich subskrypcji, które istniały dla klienta przed nawiązaniem połączenia. Wszystkie subskrypcje pozostaną aktywne po rozłączeniu połączenia z klientem.

Atrybut `cleanSession` można także rozpatrywać jako atrybut modalny, jeśli chodzi o jego wpływ na subskrypcje. W domyślnym trybie atrybutu `cleanSession=true` klient tworzy subskrypcje i odbiera publikacje tylko w zasięgu sesji. W alternatywnym trybie atrybutu `cleanSession=false` subskrypcje są trwałe. Można nawiązywać połączenia z klientem i je rozłączać, a jego subskrypcje pozostają aktywne. Po ponownym nawiązaniu połączenia z klientem odbiera on wszystkie niedostarczone publikacje. Po nawiązaniu połączenia klient może modyfikować zestaw subskrypcji aktywnych na jego potrzeby.

Przed nawiązaniem połączenia należy ustawić tryb `cleanSession` . Tryb ten będzie obowiązywać przez całą sesję. Aby zmienić ustawienie trybu, należy rozłączyć połączenie z klientem, a następnie ponownie je nawiązać. Jeśli tryby zostaną zmienione z użycia trybu `cleanSession=false` na tryb `cleanSession=true`, wszystkie wcześniejsze subskrypcje dla klienta i wszystkie publikacje, które nie zostały odebrane, zostaną odrzucone.

Publikacje zgodne z aktywnymi subskrypcjami są wysyłane do klienta natychmiast po ich opublikowaniu. Jeśli klient zostanie rozłączony, zostanie wysłany do klienta, jeśli ponownie nawiąże połączenie z tym samym serwerem z tym samym identyfikatorem klienta i wartością `MqttConnectOptions.cleanSession` ustawioną na `false`.

Subskrypcje dla konkretnego klienta są identyfikowane przez identyfikator klienta. Można ponownie połączyć klienta z innego urządzenia klienckiego z tym samym serwerem, kontynuować z tymi samymi subskrypcjami i odbierać niedostarczone publikacje.

**Pojęcia pokrewne**

Wywołania zwrotne i synchronizacja w aplikacjach klienckich MQTT
Model programowania klienta MQTT wykorzystuje wątki w szerokim zakresie. Wątki oddzielają aplikację kliencką MQTT od opóźnień w przesyłaniu komunikatów do i z serwera. Publikacje, znaczniki dostarczania i zdarzenia utraty połączenia są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Wyczyść sesje
Klient MQTT i usługa telemetryczna (MQXR) przechowują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania oraz "dokładnie raz" odbierania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z informacjami o stanie między sesjami lub bez nich. Przed nawiązaniem połączenia należy zmienić tryb czystej sesji, ustawiając opcję `MqttConnectOptions.cleanSession` .

Identyfikator klienta

Identyfikator klienta jest 23-bajtowym łańcuchem, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator może zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby mieć procedurę przydzielania identyfikatorów klienta oraz sposób konfigurowania klienta o wybranym identyfikatorze.

Znaczniki dostarczania

Ostatnia wola i publikacja testamentu
Jeśli połączenie klienta MQTT zostanie nieoczekiwanie zakończone, można skonfigurować program MQ Telemetry do wysyłania publikacji "last will and testament". Wstępnie zdefiniuj treść publikacji i temat, do którego ma ona zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Należy go utworzyć przed nawiązaniem połączenia z klientem.

Trwałość komunikatu w klientach MQTT
Komunikaty publikacji są trwałe, jeśli są wysyłane z jakością usługi "co najmniej raz"lub "dokładnie raz". Można zaimplementować własny mechanizm trwałości na kliencie lub użyć domyślnego mechanizmu trwałości, który jest dostarczany z klientem. Trwałość działa w obu kierunkach, dla publikacji wysyłanych do lub z klienta.

Publikacje
Publikacje są instancjami `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty MQTT mogą tworzyć publikacje wysyłane do IBM MQi subskrybować tematy w serwisie IBM MQ w celu odbierania publikacji.

Jakość usługi zapewniana przez klienta MQTT
Klient systemu MQTT udostępnia trzy elementy jakości usług na potrzeby dostarczania publikacji do produktu IBM MQ i klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie raz". Gdy klient MQTT wysyła żądanie do usługi IBM MQ w celu utworzenia subskrypcji, jest ono wysyłane z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną zachowaną publikację. Jeśli zostanie utworzona subskrypcja tematu, który ma zachowaną publikację, zostanie ona natychmiast przekazana do użytkownika.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuchy tematów i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama jak w IBM MQ.

## Łańcuchy tematów i filtry tematów w klientach MQTT

Łańcuchy tematów i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama jak w IBM MQ.

Łańcuchy tematów są używane do wysyłania publikacji do subskrybentów. Utwórz łańcuch tematu przy użyciu metody `MqttClient.getTopic(java.lang.String topicString)`.

Filtry tematów służą do subskrybowania tematów i odbierania publikacji. Filtry tematów mogą zawierać znaki wieloznaczne. Znaki wieloznaczne umożliwiają subskrybowanie wielu tematów. Utwórz filtr tematów przy użyciu metody subskrypcji, na przykład `MqttClient.subscribe(java.lang.String topicFilter)`.

### Łańcuchy tematów

Składnia łańcucha tematu IBM MQ jest opisana w sekcji Łańcuchy tematu. Składnia łańcuchów tematu MQTT jest opisana w klasie `MqttClient` w dokumentacji interfejsu API dla MQTT client for Java. Odsyłacze do dokumentacji interfejsu API klienta dla bibliotek klienta MQTT zawiera sekcja Skorowidz programistyczny klienta MQTT.

Składnia każdego typu łańcucha tematu jest prawie identyczna. Istnieją cztery niewielkie różnice:

1. Łańcuchy tematów wysyłane do produktu IBM MQ przez klienty MQTT muszą być zgodne z konwencją nazw menedżerów kolejek.

2. Maksymalne długości różnią się. IBM MQ łańcuchy tematów są ograniczone do 10 240 znaków. Klient MQTT może tworzyć łańcuchy tematów o długości do 65535 bajtów.
3. Łańcuch tematu utworzony przez klienta MQTT nie może zawierać znaku o kodzie zero.
4. W produkcie IBM Integration Buspoziom tematu o wartości NULL ( `'...//...'` ) jest niepoprawny. Poziomy tematów o wartości NULL są obsługiwane przez produkt IBM MQ.

W przeciwieństwie do protokołu IBM MQ publish/subscribe, protokół `mqttv3` nie ma pojęcia obiektu tematu administracyjnego. Nie można utworzyć łańcucha tematu na podstawie obiektu tematu i łańcucha tematu. Jednak łańcuch tematu jest odwzorowywany na temat administracyjny w sekcji IBM MQ. Kontrola dostępu powiązana z tematem administracyjnym określa, czy publikacja jest publikowana w temacie, czy odrzucana. Atrybuty tematu administracyjnego mają wpływ na atrybuty, które są stosowane do publikacji podczas jej przekazywania do subskrybentów.

## Filtry tematów

Składnia filtru tematu IBM MQ jest opisana w sekcji Schemat znaków wieloznacznych oparty na temacie. Składnia filtrów tematów, które można utworzyć za pomocą klienta MQTT , jest opisana w klasie `MqttClient` w dokumentacji interfejsu API dla MQTT client for Java. Odsyłacze do dokumentacji interfejsu API klienta dla bibliotek klienta MQTT zawiera sekcja Skorowidz programistyczny klienta MQTT.

**Pojęcia pokrewne**

Wywołania zwrotne i synchronizacja w aplikacjach klienckich MQTT
Model programowania klienta MQTT wykorzystuje wątki w szerokim zakresie. Wątki oddzielają aplikację kliencką MQTT od opóźnień w przesyłaniu komunikatów do i z serwera. Publikacje, znaczniki dostarczania i zdarzenia utraty połączenia są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Wyczyść sesje
Klient MQTT i usługa telemetryczna (MQXR) przechowują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania oraz "dokładnie raz" odbierania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z informacjami o stanie między sesjami lub bez nich. Przed nawiązaniem połączenia należy zmienić tryb czystej sesji, ustawiając opcję `MqttConnectOptions.cleanSession` .

Identyfikator klienta
Identyfikator klienta jest 23-bajtowym łańcuchem, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator może zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby mieć procedurę przydzielania identyfikatorów klienta oraz sposób konfigurowania klienta o wybranym identyfikatorze.

Znaczniki dostarczania

Ostatnia wola i publikacja testamentu
Jeśli połączenie klienta MQTT zostanie nieoczekiwanie zakończone, można skonfigurować program MQ Telemetry do wysyłania publikacji "last will and testament". Wstępnie zdefiniuj treść publikacji i temat, do którego ma ona zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Należy go utworzyć przed nawiązaniem połączenia z klientem.

Trwałość komunikatu w klientach MQTT
Komunikaty publikacji są trwałe, jeśli są wysyłane z jakością usługi "co najmniej raz"lub "dokładnie raz". Można zaimplementować własny mechanizm trwałości na kliencie lub użyć domyślnego mechanizmu trwałości, który jest dostarczany z klientem. Trwałość działa w obu kierunkach, dla publikacji wysyłanych do lub z klienta.

Publikacje
Publikacje są instancjami `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty MQTT mogą tworzyć publikacje wysyłane do IBM MQi subskrybować tematy w serwisie IBM MQ w celu odbierania publikacji.

Jakość usługi zapewniana przez klienta MQTT

Klient systemu MQTT udostępnia trzy elementy jakości usług na potrzeby dostarczania publikacji do produktu IBM MQ i klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie raz". Gdy klient MQTT wysyła żądanie do usługi IBM MQ w celu utworzenia subskrypcji, jest ono wysyłane z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną zachowaną publikację. Jeśli zostanie utworzona subskrypcja tematu, który ma zachowaną publikację, zostanie ona natychmiast przekazana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. W celu zarejestrowania zainteresowania wieloma tematami klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, w którym używane są znaki wieloznaczne. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest rozłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

# Tworzenie aplikacji Microsoft Windows Communication Foundation przy użyciu języka IBM MQ

Kanał niestandardowy produktu Microsoft Windows Communication Foundation (WCF) dla produktu IBM MQ wysyła i odbiera komunikaty między klientami i usługami WCF.

**Pojęcia pokrewne**

"Wprowadzenie do kanału niestandardowego produktu IBM MQ dla środowiska WCF z produktem .NET" na stronie 1300
Kanał niestandardowy dla produktu IBM MQ jest kanałem transportowym używającym zunifikowanego modelu programowania Microsoft Windows Communication Foundation (WCF).

"Korzystanie z kanałów niestandardowych produktu IBM MQ dla środowiska WCF" na stronie 1305
Przegląd informacji dostępnych dla programistów korzystających z kanałów niestandardowych systemu IBM MQ dla produktu Windows Communication Foundation (WCF).

"Korzystanie z przykładów WCF" na stronie 1325
Przykłady produktu Windows Communication Foundation (WCF) zawierają kilka prostych przykładów użycia niestandardowego kanału IBM MQ .

FFST: Technologia obsługi pierwszego niepowodzenia WCF XMS

**Zadania pokrewne**

Śledzenie kanału niestandardowego WCF dla produktu IBM MQ

Rozwiązywanie problemów z niestandardowym kanałem WCF w produkcie IBM MQ

## Wprowadzenie do kanału niestandardowego produktu IBM MQ dla środowiska WCF z produktem .NET

Kanał niestandardowy dla produktu IBM MQ jest kanałem transportowym używającym zunifikowanego modelu programowania Microsoft Windows Communication Foundation (WCF).

Środowisko Microsoft Windows Communication Foundation framework, wprowadzone w wersji Microsoft.NET 3, umożliwia tworzenie aplikacji i usług .NET niezależnie od transportu i protokołów używanych do ich łączenia, umożliwiając używanie alternatywnych transportów lub konfiguracji zgodnie ze środowiskiem, w którym wdrożono usługę lub aplikację.

Połączenia są zarządzane w czasie wykonywania przez narzędzie WCF przez budowanie stosu kanału zawierającego wymaganą kombinację następujących elementów:

- Elementy protokołu: opcjonalny zestaw elementów, w przypadku których nie można dodać jednego lub więcej elementów w celu obsługi protokołów, takich jak standardy WS-*.
- Program kodujący komunikaty: obowiązkowy element w stosie sterujący serializowaniem komunikatu do jego formatu łącznika.
- Kanał transportowy: obowiązkowy element w stosie odpowiedzialny za transport komunikatu przekształconego do postaci szeregowej do jego punktu końcowego.

Kanał niestandardowy dla produktu IBM MQ jest kanałem transportowym i dlatego musi być sparowany z koderem komunikatów i opcjonalnymi protokołami zgodnie z wymaganiami aplikacji korzystającej z niestandardowego powiązania WCF. W ten sposób aplikacje, które zostały zaprojektowane pod kątem używania produktu WCF, mogą używać kanału niestandardowego produktu IBM MQ do wysyłania i odbierania danych w taki sam sposób, w jaki korzystają z wbudowanych transportów udostępnianych przez produkt Microsoft, co umożliwia prostą integrację z asynchronicznymi, skalowalnymi i niezawodnymi funkcjami przesyłania komunikatów produktu IBM MQ. Pełną listę obsługiwanych funkcji zawiera sekcja "Niestandardowe funkcje i możliwości kanału WCF" na stronie 1305.

## Kiedy i dlaczego używany jest niestandardowy kanał IBM MQ dla WCF?

Kanału niestandardowego produktu IBM MQ można używać do wysyłania i odbierania komunikatów między klientami i usługami WCF w taki sam sposób, jak wbudowanych transportów udostępnianych przez produkt Microsoft, co umożliwia aplikacjom uzyskiwanie dostępu do funkcji produktu IBM MQ w ujednoliconym modelu programistycznym WCF.

Typowe scenariusze wzorców użycia dla kanału niestandardowego produktu IBM MQ dla systemu WCF to interfejs inny niż SOAP na potrzeby transmisji rodzimych komunikatów produktu IBM MQ .

### Komunikaty przenoszone przy użyciu formatu komunikatu innego niż SOAP/innego niżJMS (Pure MQMessage)

Jeśli kanał niestandardowy IBM MQ dla produktu WCF jest używany jako interfejs inny niż SOAP na potrzeby transmisji rodzimych komunikatów produktu IBM MQ , komunikaty są przenoszone przy użyciu formatu komunikatu innego niż SOAP/Non-JMS (Pure MQMessage) produktu IBM MQ.

Użytkownicy środowiska WCF mogą uruchamiać usługę, czyli mogą wysyłać komunikaty do kolejki produktu IBM MQ za pomocą komunikatów MQMessage. Aplikacje mogą pobrać i ustawić pola i ładunek MQMD. Jeśli komunikat jest dostępny w kolejkach IBM MQ , może zostać przetworzony przez dowolną usługę WCF lub aplikacje inne niż WCF, takie jak aplikacje C lub Java działające w systemie AIX, Linux, Windowslub z/OS.

## Wymagania programowe dotyczące kanału niestandardowego produktu IBM MQ dla środowiska WCF

W tej sekcji przedstawiono wymagania programowe dla niestandardowego kanału produktu IBM MQ dla środowiska WCF. Kanał niestandardowy produktu IBM MQ dla systemu WCF może łączyć się tylko z menedżerami kolejek w wersji IBM WebSphere MQ 7.0 lub nowszej.

### Wymagania środowiska wykonawczego

- Na komputerze hosta musi być zainstalowane środowisko Microsoft.NET Framework v4.7.2 lub nowsze.
- *Przesyłanie komunikatów Java i .NET oraz usługi Web Services* są instalowane domyślnie jako część instalatora produktu IBM MQ . Ten komponent instaluje zespoły .NET wymagane dla kanału niestandardowego w globalnej pamięci podręcznej zespołu.

**Uwaga:** Jeśli produkt Microsoft .NET Framework V4.7.2 lub nowszej nie zostanie zainstalowany przed zainstalowaniem produktu IBM MQ, instalacja produktu IBM MQ będzie kontynuowana bez błędu, ale plik IBM MQ classes for .NET nie będzie dostępny. Jeśli produkt.NET Framework został zainstalowany po zainstalowaniu produktu IBM MQ, należy zarejestrować zespoły produktu IBM MQ.NET , uruchamiając skrypt *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` , gdzie *WMQInstallDir* jest katalogiem, w którym zainstalowano produkt IBM MQ . Ten skrypt instaluje wymagane zespoły w globalnej pamięci podręcznej zespołu (GAC). Zestaw plików `amqi*.log` , które rejestrują wykonywane działania, jest tworzony w katalogu %TEMP% . Nie jest konieczne ponowne uruchamianie skryptu `amqiRegisterdotNet.cmd` , jeśli produkt .NET został zaktualizowany do wersji V4.7.2 lub nowszej z wcześniejszej wersji, na przykład z produktu .NET V3.5.

### Wymagania dotyczące środowiska programistycznego

- Microsoft Visual Studio 2015 lub Windows Software Development Kit for .NET 4.7.2 lub nowszy.
- Aby można było zbudować przykładowe pliki rozwiązania, na komputerze hosta musi być zainstalowane środowisko Microsoft.NET Framework V4.7.2 lub nowszej.

## IBM MQ custom channel for WCF: Co jest zainstalowane?

Kanał niestandardowy dla produktu IBM MQ jest kanałem transportowym używającym zunifikowanego modelu programowania Microsoft Windows Communication Foundation (WCF). Kanał niestandardowy jest instalowany domyślnie w ramach instalacji.

### Kanał niestandardowy IBM MQ dla WCF

Kanał niestandardowy i jego zależności są zawarte w komponencie `Java and .NET Messaging and Web Services` , który jest instalowany domyślnie. Podczas aktualizowania produktu IBM MQ z wcześniejszej wersji niżIBM MQ 8.0aktualizacja domyślnie instaluje niestandardowy kanał IBM MQ dla WCF, jeśli komponent `Java and .NET Messaging and Web Services` został wcześniej zainstalowany we wcześniejszej instalacji.

Komponent `.NET Messaging and Web Services` zawiera plik `IBM.XMS.WCF.dll` i plik `IBM.WMQ.WCF.dll` , a te pliki są głównym zespołem kanału niestandardowego, który zawiera klasy interfejsu WCF. Te pliki są instalowane w globalnej pamięci podręcznej zespołu poprawek (Global Assembly Cache-GAC) i są również dostępne w następującym katalogu: *MQ_INSTALLATION_PATH* `\bin` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem, w którym zainstalowano produkt IBM MQ .

Poniższa tabela zawiera podsumowanie klas kluczy, które są wymagane do korzystania z kanału niestandardowego.

*Tabela 189. Klasy kluczy wymagane do korzystania z kanału niestandardowego*

| | **Interfejs SOAP/JMS (istniejący)** | **Interfejs inny niż SOAP/inny niżJMS (z IBM MQ 8.0 )** |
|---|---|---|
| Zespół kanału niestandardowego | IBM.XMS.WCF.dll | IBM.WMQ.WCF.dll |
| Nazwa powiązania transportu | IBM.XMS.WCF.SoapJmsIbmTransportBindingElement | IBM.WMQ.WCF.WmqIbmTransportBindingElement |
| Program importujący powiązania transportu | IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter | IBM.WMQ.WCF.WmqIbmTransportBindingElementImporter |
| Konfiguracja powiązania transportu | IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig | IBM.WMQ.WCF.WmqIbmTransportBindingElementConfig |
| Przykłady (jednokierunkowe) | SimpleOneWay_Client, SimpleOneWay_Service | MQMessaging_OneWay_Client, MQMessaging_OneWay_Service |
| Przykłady (RequestReply) | SimpleRequestReply_Client, SimpleRequestReply_Service | MQMessaging_RequestReply_Client, MQMessaging_RequestReply_Service |

IBM.WMQ.WCF.dll obsługuje zarówno interfejsy SOAP/JMS , jak i Non-SOAP/Non-JMS . Do korzystania z IBM.WMQ.WCF , ponieważ obsługuje oba interfejsy.

### Wysyłanie sformatowanych komunikatów MQSTR

Jeśli komunikat żądania jest typu MQSTR, można wysłać komunikat odpowiedzi w formacie MQSTR.

Aby zmienić format komunikatu odpowiedzi, należy użyć dodatkowego parametru identyfikatora URI **replyMessageFormat** . Obsługiwane są następujące wartości:

**""**

" jest wartością domyślną.

Komunikat odpowiedzi jest w formacie bajtowym (MQMFT_NONE). Na przykład:

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t= "
```

**MQSTR**

Komunikat odpowiedzi jest w formacie MQSTR (MQMFT_STRING). Na przykład:

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t=MQSTR"
```

**Uwagi:**

1. W wartości **replyMessageFormat** nie jest rozróżniana wielkość liter.
2. Użycie dowolnej wartości innej niż "" lub *MQSTR*powoduje wystąpienie wyjątku dotyczącego niepoprawnej wartości parametru.

## Przykłady kanału niestandardowego IBM MQ

Przykłady zawierają kilka prostych przykładów użycia niestandardowego kanału IBM MQ dla WCF. Przykłady i powiązane z nimi pliki znajdują się w katalogu *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym produktu IBM MQ. Więcej informacji na temat przykładowych kanałów niestandardowych produktu IBM MQ zawiera sekcja “Korzystanie z przykładów WCF” na stronie 1325.

## svcutil.exe.config

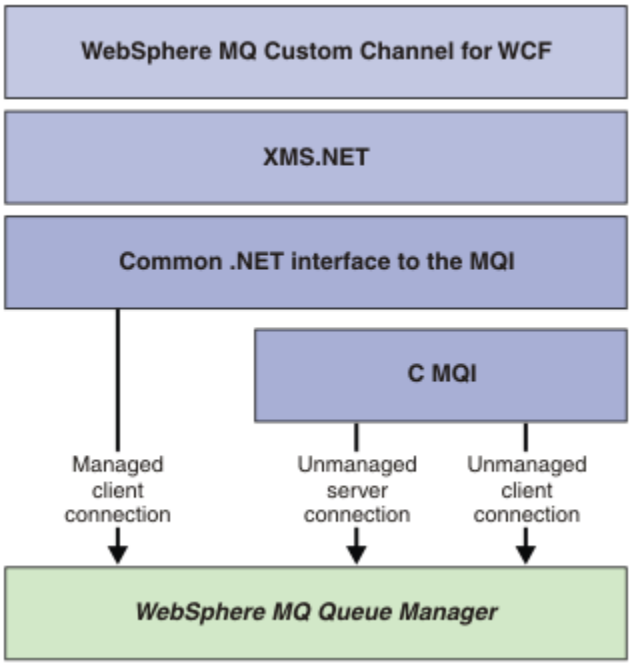
`svcutil.exe.config` jest przykładem ustawień konfiguracyjnych wymaganych do włączenia narzędzia do generowania proxy klienta Microsoft WCF `svcutil` w celu rozpoznania kanału niestandardowego. Plik `svcutil.exe.config` znajduje się w katalogu *MQ_INSTALLATION_PATH* `\tools\wcf\docs\examples\` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym IBM MQ. Więcej informacji na temat korzystania z `svcutil.exe.config`zawiera sekcja “Generowanie plików konfiguracyjnych aplikacji i proxy klienta WCF przy użyciu narzędzia svcutil z metadanymi z działającej usługi” na stronie 1322.

# Architektura WCF

Niestandardowy kanał IBM MQ dla WCF jest zintegrowany z interfejsem API `IBM Message Service Client for .NET (XMS .NET)` .

## Interfejs SOAP/JMS

Architektura WCF jest przedstawiona na poniższym diagramie:




*Rysunek 149. Architektura WCF dla interfejsu SOAP/JMS*

Wszystkie wymagane komponenty są instalowane domyślnie wraz z instalacją produktu.

Trzy połączenia są następujące:

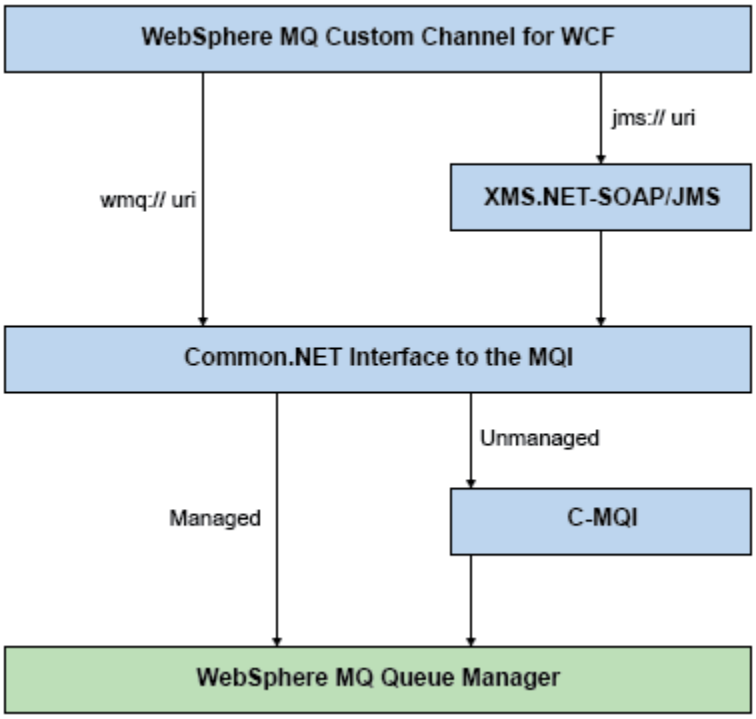
- Zarządzane połączenia klienta
- Niezarządzane połączenia serwera
- Niezarządzane połączenia klienckie

Więcej informacji na temat tych połączeń zawiera sekcja “Opcje połączenia WCF” na stronie 1311.

## Interfejs inny niż SOAP/inny niżJMS

Niestandardowy kanał IBM MQ dla WCF obsługuje zarówno interfejs SOAP/JMS (dostępny z serwisu IBM WebSphere MQ 7.0.1 ) i interfejsu innego niż SOAP/Non-JMS .

Architektura WCF jest przedstawiona na poniższym diagramie:




*Rysunek 150. Architektura WCF dla interfejsu innego niż SOAP/Non-JMS*

# Korzystanie z kanałów niestandardowych produktu IBM MQ dla środowiska WCF

Przegląd informacji dostępnych dla programistów korzystających z kanałów niestandardowych systemu IBM MQ dla produktu Windows Communication Foundation (WCF).

Produkt Microsoft Windows Communication Foundation stanowi podstawę obsługi usług Web Service i przesyłania komunikatów w środowisku Microsoft.NET Framework 3. Produkt IBM MQ może być używany jako kanał niestandardowy w środowisku WCF w środowisku .NET Framework 3 w taki sam sposób, jak wbudowane kanały oferowane przez produkt Microsoft.

Komunikaty transportowane przez kanał niestandardowy są formatowane zgodnie z implementacją protokołu SOAP korzystającego z protokołu JMS produktu IBM MQ. Aplikacje mogą następnie komunikować się z usługami udostępnianymi przez usługę WCF lub przez infrastrukturę usług WebSphere SOAP over JMS .

## Niestandardowe funkcje i możliwości kanału WCF

Poniższe tematy zawierają informacje dotyczące funkcji i możliwości kanału niestandardowego WCF.

### *Niestandardowe kształty kanału WCF*

Przegląd niestandardowych kształtów kanałów, których produkt IBM MQ może używać w kanałach niestandardowych produktu Microsoft Windows Communication Foundation (WCF).

Niestandardowy kanał IBM MQ dla WCF obsługuje dwa kształty kanału:

- Jednokierunkowa
- Żądanie-odpowiedź

WCF automatycznie wybiera kształt kanału zgodnie z udostępnianą umową o świadczenie usług.

Kontrakty zawierające metody, które korzystają tylko z parametru **IsOneWay** , są obsługiwane przez jednokierunkowy kształt kanału, na przykład:

```
[OperationContract(IsOneWay = true)]
void printString(String text);
```

Kontrakty, które zawierają zarówno metodę jednokierunkową, jak i metodę żądanie-odpowiedź, lub wszystkie metody żądanie-odpowiedź, są obsługiwane przez kształt kanału żądanie-odpowiedź. Na przykład:

```
    [OperationContract]
    int subtract(int a, int b);

    [OperationContract(IsOneWay = true)]
    void printString(string text);
```

**Uwaga:** W przypadku mieszania metod jednokierunkowych i żądanie-odpowiedź w tym samym kontrakcie należy upewnić się, że zachowanie jest zgodne z zamierzonym, szczególnie w przypadku pracy w środowisku mieszanym, ponieważ metody jednokierunkowe oczekują na odpowiedź o wartości NULL od usługi.

## Kanał jednokierunkowy

Jednokierunkowy kanał niestandardowy IBM MQ dla WCF jest używany na przykład do wysyłania komunikatów z klienta WCF przy użyciu jednokierunkowego kształtu kanału. Kanał może wysyłać komunikaty tylko w jednym kierunku, na przykład z menedżera kolejek klienta do kolejki w usłudze WCF.

## Kanał żądanie-odpowiedź

Kanał niestandardowy typu żądanie-odpowiedź IBM MQ dla usługi WCF jest używany na przykład do asynchronicznego wysyłania komunikatów w dwóch kierunkach. Ta sama instancja klienta musi być używana na potrzeby asynchronicznego przesyłania komunikatów. Kanał może wysyłać komunikaty w jednym kierunku, na przykład z menedżera kolejek klienta do kolejki w usłudze WCF, a następnie wysyłać komunikat odpowiedzi z narzędzia WCF do kolejki w menedżerze kolejek klienta.

### *Nazwy i wartości parametrów identyfikatorów URI WCF*

Nazwy i wartości parametrów URI dla interfejsu SOAP/JMS i interfejsu Non-SOAP/Non JMS .

## Interfejs SOAP/JMS

**connectionFactory**
Parametr connectionFactory jest wymagany.

**InitialContextFactory**
Parametr fabryki initialContextjest wymagany i musi być ustawiony na wartośćcom.ibm.mq.jms.Nojndi, aby zapewnić kompatybilność z produktem WebSphere Application Server i innymi produktami.

## Interfejs inny niż SOAP/inny niż JMS

Format identyfikatora URI jest taki sam, jak w przypadku specyfikacji MA93 . Więcej informacji na temat specyfikacji IRI IBM MQ zawiera publikacja SupportPac - MA93 .

**Składnia identyfikatora URI IBM MQ**

```
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 Przykład IRI**

W poniższym przykładzie IRI informuje requester usług, że może użyć połączenia powiązania klienta TCP IBM MQ z komputerem nazywanym example.com na porcie 1414 i umieścić komunikaty żądań trwałych w kolejce o nazwie SampleQ w menedżerze kolejek QM1. IIRI określa, że dostawca usług umieści odpowiedzi w kolejce o nazwie SampleReplyQ.

```
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
```

**Dla połączeń z włączoną obsługą protokołu TLS**

Aby nawiązywać połączenia TLS (Secured) przy użyciu klienta/usługi WCF, należy ustawić następujące właściwości z odpowiednimi wartościami w identyfikatorze URI. Wszystkie właściwości z przedrostkiem "*" są obowiązkowe do nawiązania bezpiecznego połączenia.

- **sslKeyRepository**: *SYSTEM lub *USER
- * **sslCipherSpec**: poprawna CipherSpec, na przykład TLS_RSA_WITH_AES_128_CBC_SHA256.
- **sslCertRevocationCheck**: true lub false.
- **sslKeyResetCount**: wartość większa niż 32kb.
- **sslPeerName:** nazwa wyróżniająca certyfikatu serwera

Na przykład:

```
"wmq://localhost:1414/msg/queue/SampleQ?
connectQueueManager=QM1&sslkeyrepository=*SYSTEM&sslcipherspec=
TLS_RSA_WITH_AES_128_CBC_SHA&sslcertrevocationcheck=true&"sslpe
ername=" + "" + "CN=ibmwebspheremqqmm&sslkeyresetcount=45000"
```

## *Gwarantowane dostarczenie kanału niestandardowego WCF*

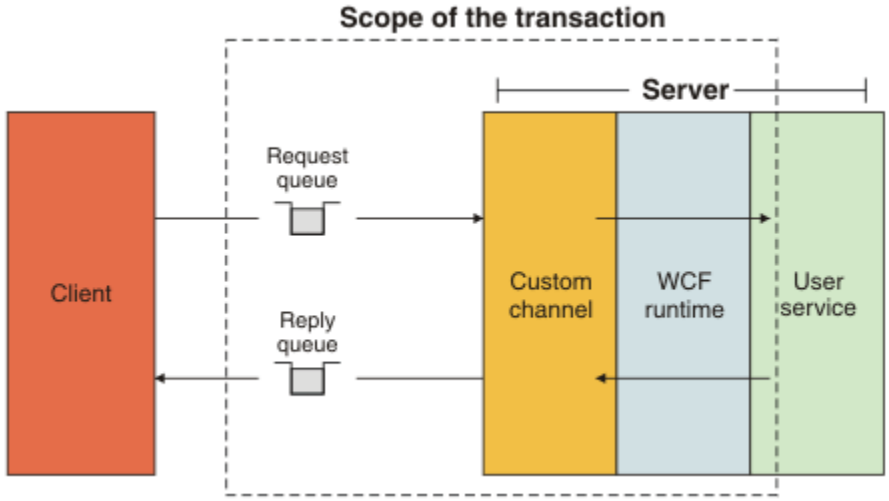
Gwarantowana dostawa gwarantuje, że zlecenie usługi lub odpowiedź zostanie wykonane i nie zostanie utracona.

Komunikat żądania jest odbierany, a każdy komunikat odpowiedzi jest wysyłany w ramach lokalnego punktu synchronizacji transakcji, który może zostać wycofany w przypadku niepowodzenia środowiska wykonawczego. Przykładami tych niepowodzeń są: nieobsłużony wyjątek zgłoszony przez usługę, niepowodzenie rozesłania komunikatu do usługi lub niepowodzenie dostarczenia komunikatu odpowiedzi.

`AssuredDelivery` to atrybut gwarantowanej dostawy, który można określić w umowie o świadczenie usług w celu zagwarantowania, że wszystkie komunikaty żądań odebrane przez usługę i wszystkie komunikaty odpowiedzi wysłane z usługi nie zostaną utracone w przypadku awarii środowiska wykonawczego.

Aby zapewnić zachowanie komunikatów w przypadku awarii systemu lub przerwy w zasilaniu, komunikaty muszą być wysyłane jako trwałe. Aby można było używać trwałych komunikatów, aplikacja kliencka musi mieć tę opcję określoną w swoim identyfikatorze URI punktu końcowego.

Transakcje rozproszone nie są obsługiwane, a zasięg transakcji nie wykracza poza przetwarzanie komunikatu żądania i odpowiedzi wykonywane przez produkt IBM MQ. Każda praca wykonana w ramach usługi może zostać ponownie uruchomiona w wyniku niepowodzenia, które powoduje ponowne odebranie komunikatu. Poniższy diagram przedstawia zasięg transakcji:




Gwarantowane dostarczenie jest włączane przez zastosowanie atrybutu `AssuredDelivery` do klasy usługi, jak pokazano w poniższym przykładzie:

```
[AssuredDelivery]
class TestCalculatorService : IWMQSampleCalculatorContract
{
    public int add(int a, int b)
    {
        int ans = a + b;
        return ans;
    }
}
```

Jeśli używany jest atrybut `AssuredDelivery` , należy pamiętać o następujących kwestiach:

- Jeśli kanał określi, że niepowodzenie może się powtórzyć, jeśli komunikat został wycofany i odebrany ponownie, komunikat jest traktowany jako komunikat nieprzetwarzalny i nie jest zwracany do kolejki żądań w celu ponownego przetworzenia. Na przykład: jeśli odebrany komunikat nie jest poprawnie sformatowany lub nie może zostać rozesłany do usługi. Nieobsłużone wyjątki zgłaszane przez operację usługi są zawsze ponownie zgłaszane, dopóki komunikat nie zostanie ponownie dostarczony przez maksymalną liczbę razy określoną we właściwości progu wycofania w kolejce żądań. Więcej informacji na ten temat można znaleźć pod adresem: “Niestandardowe komunikaty nieprzetwarzalne kanału WCF” na stronie 1309
- Kanał wykonuje odczyt, przetwarzanie i odpowiadanie na każdy komunikat żądania jako operację niepodzielną przy użyciu pojedynczego wątku wykonywania w celu wymuszenia integralności transakcji. Aby umożliwić współbieżne działanie operacji usługi, kanał umożliwia środowisku WCF tworzenie wielu instancji kanału. Liczba instancji kanału dostępnych na potrzeby przetwarzania żądań jest kontrolowana przez właściwość powiązania `MaxConcurrentCalls`. Więcej informacji na ten temat można znaleźć pod adresem: “Opcje konfiguracyjne powiązania WCF” na stronie 1317
- Funkcja gwarantowanego dostarczania używa zarówno punktu rozszerzenia WCF IOperationInvoker , jak i IErrorHandler . Jeśli te punkty rozszerzalności są używane zewnętrznie przez aplikację, aplikacja musi zapewnić, że wszystkie wcześniej zarejestrowane punkty rozszerzalności są wywoływane. Jeśli procedura obsługi IErrorHandler nie będzie tego robić, może to spowodować, że błędy nie będą zgłaszane. Jeśli ta czynność nie powiedzie się dla elementu IOperationInvoker , może to spowodować, że system WCF przestanie odpowiadać.

### *Zabezpieczenia kanału niestandardowego WCF*

Kanał niestandardowy produktu IBM MQ dla systemu WCF obsługuje użycie protokołu TLS tylko w przypadku niezarządzanych połączeń klienckich z menedżerem kolejek.

Określ TLS, używając pozycji w tabeli definicji kanału klienta (CCDT). Więcej informacji na temat tabel CCDT zawiera sekcja Tabela definicji kanału klienta.

### *Tabele definicji kanału klienta WCF (CCDT)*

Niestandardowy kanał IBM MQ dla WCF obsługuje użycie tabel definicji kanału klienta (CCDT) w celu skonfigurowania informacji o połączeniu dla połączeń klienckich.

CDC są kontrolowane przez następujące dwie zmienne środowiskowe:

- *MQCHLLIB* określa katalog, w którym znajduje się tabela.
- *MQCHLTAB* określa nazwę pliku tabeli.

Jeśli te zmienne środowiskowe są zdefiniowane, mają one wyższy priorytet niż wszystkie szczegóły połączenia klienta określone w identyfikatorze URI.

Więcej informacji na temat tabel definicji kanału klienta zawiera sekcja Tabela definicji kanału klienta.

### *Niestandardowe komunikaty nieprzetwarzalne kanału WCF*

Jeśli usługa nie może przetworzyć komunikatu żądania lub nie może dostarczyć komunikatu odpowiedzi do kolejki odpowiedzi, komunikat jest traktowany jako komunikat nieprzetwarzalny.

## Komunikaty żądań nieprzetwarzalnych

Jeśli nie można przetworzyć komunikatu żądania, jest on traktowany jako komunikat nieprzetwarzalny. To działanie zapobiega ponownemu odebraniu przez usługę tego samego komunikatu, który nie może być przetwarzany. Aby nieprzetwarzalny komunikat żądania był traktowany jako komunikat nieprzetwarzalny, musi być spełniony jeden z następujących warunków:

- Liczba wycofanych komunikatów przekroczyła próg wycofania określony w kolejce żądań, który występuje tylko wtedy, gdy dla usługi określono gwarantowane dostarczenie. Więcej informacji na temat gwarantowanego dostarczania można znaleźć pod adresem: “Gwarantowane dostarczenie kanału niestandardowego WCF” na stronie 1307
- Komunikat nie został poprawnie sformatowany i nie mógł zostać zinterpretowany jako komunikat protokołu SOAP korzystającego z protokołu JMS .

## Nieprzetwarzalne komunikaty odpowiedzi

Jeśli usługa nie może dostarczyć komunikatu odpowiedzi do kolejki odpowiedzi, komunikat odpowiedzi jest traktowany jako komunikat nieprzetwarzalny. W przypadku komunikatów odpowiedzi to działanie umożliwia późniejsze pobranie komunikatów odpowiedzi w celu określenia problemu.

## Obsługa komunikatów nieprzetwarzalnych

Działanie podjęte dla komunikatu nieprzetwarzalnego zależy od konfiguracji menedżera kolejek i wartości ustawionych w opcjach raportu komunikatu. W przypadku protokołu SOAP korzystającego z protokołu JMSnastępujące opcje raportu są domyślnie ustawione dla komunikatów żądania i nie można ich konfigurować:

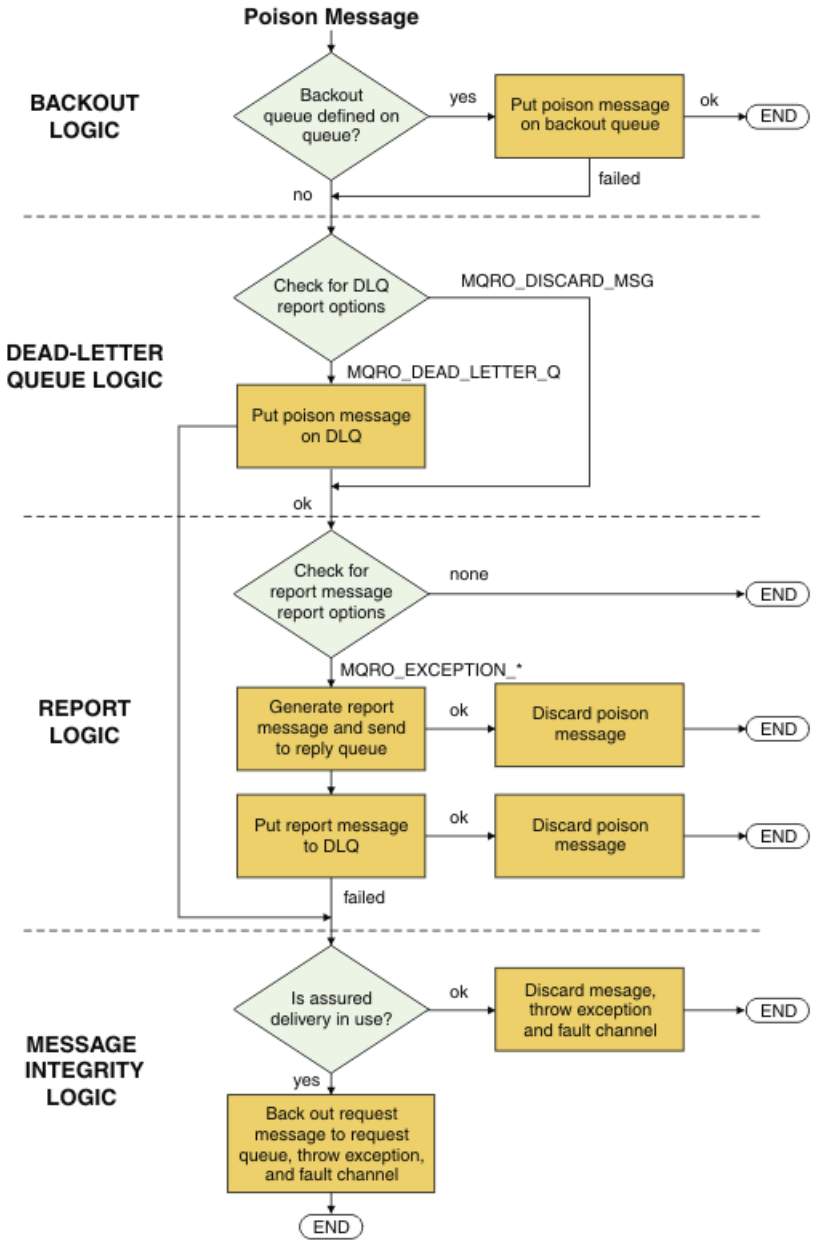
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

W przypadku protokołu SOAP korzystającego z protokołu JMSnastępująca opcja raportu jest domyślnie ustawiona dla komunikatów odpowiedzi i nie można jej konfigurować:

- MQRO_DEAD_LETTER_Q

Jeśli komunikaty pochodzą ze źródła innego niż WCF, należy zapoznać się z dokumentacją tego źródła.

Poniższy diagram przedstawia możliwe działania i kroki wykonywane w przypadku niepowodzenia obsługi komunikatów nieprzetwarzalnych:




## *Możliwości komunikatów IBM MQ dla aplikacji WCF*

Non-SOAP/Non-JMS (to znaczy IBM MQ ) możliwości komunikatów dla aplikacji WCF.

W przypadku interfejsu innego niż SOAP/Non-JMS możliwości komunikatów IBM MQ dla aplikacji WCF są następujące:

- Aplikacje WCF mogą wysyłać i odbierać podstawowe komunikaty IBM MQ , które mogą być przetwarzane przez dowolną aplikację IBM MQ .
- Aplikacje WCF mają pełną kontrolę nad aktualizowaniem deskryptora MQMD i ładunku.
- Klient WCF może wysyłać komunikaty IBM MQ , które mogą być używane przez dowolne klienty IBM MQ , na przykład klienty C, Java, JMSi .NET .

Interfejs WCF for Non-SOAP/Non-JMS musi używać następujących klas do ustawiania ładunku komunikatu i deskryptora MQMD dla komunikatu:

- WmqStringKomunikat dla ładunku typu String
- WmqBytesKomunikat dla ładunku typu Bytes
- WmqXmlKomunikat dla ładunku typu XML

Aby ustawić ładunek komunikatu, należy użyć właściwości **Data** dla komunikatu WmqString, komunikatu WmqByteslub klasy komunikatu WmqXml(w zależności od typu ładunku). Na przykład użyj następującego kodu, aby ustawić ładunek typu String:

```
WmqStringMessage strMsg = new WmqStringMessage();
//Setting the Message PayLoad
strMsg.Data = "Hello World";
//MQMD property
strMsg.Format = WmqMessageFormat.MQFMT_STRING;
```

## Opcje połączenia WCF

Istnieją trzy tryby łączenia niestandardowego kanału produktu IBM MQ dla produktu WCF z menedżerem kolejek. Należy rozważyć, który typ połączenia najlepiej odpowiada wymaganiom.

Więcej informacji na temat opcji połączenia zawiera sekcja “Różnice w połączeniach” na stronie 598

Więcej informacji na temat architektury WCF można znaleźć pod adresem: “Architektura WCF” na stronie 1303

### Niezarządzane połączenie klienta

Połączenie nawiązane w tym trybie łączy się jako klient IBM MQ z serwerem IBM MQ działającym na komputerze lokalnym lub zdalnym.

Aby użyć niestandardowego kanału IBM MQ dla produktu WCF jako klienta IBM MQ , można go zainstalować razem z produktem IBM MQ MQI clientna serwerze IBM MQ lub na osobnym komputerze.

### Niezarządzane połączenie z serwerem

W przypadku użycia w trybie powiązań serwera niestandardowy kanał produktu IBM MQ dla produktu WCF używa interfejsu API menedżera kolejek zamiast komunikacji sieciowej. Korzystanie z połączeń powiązań zapewnia lepszą wydajność aplikacji IBM MQ niż korzystanie z połączeń sieciowych.

Aby użyć połączenia powiązań, należy na serwerze IBM MQ zainstalować niestandardowy kanał IBM MQ dla produktu WCF.

### Zarządzane połączenie klienta

Połączenie nawiązane w tym trybie łączy się jako klient IBM MQ z serwerem IBM MQ działającym na komputerze lokalnym lub zdalnym.

Klasy kanału niestandardowego IBM MQ dla systemu .NET 3 łączącego się w tym trybie pozostają w kodzie zarządzanym .NET i nie wywołują usług rodzimych. Więcej informacji na temat kodu zarządzanego zawiera dokumentacja systemu Microsoft .

Korzystanie z klienta zarządzanego podlega pewnym ograniczeniom. Więcej informacji na temat tych ograniczeń zawiera sekcja “Zarządzane połączenia klienta” na stronie 598.

## Tworzenie i konfigurowanie niestandardowego kanału IBM MQ dla produktu WCF

Kanały niestandardowe produktu IBM MQ dla produktu WCF działają w taki sam sposób, jak kanały transportowe produktu WCF oferowane przez produkt Microsoft. Niestandardowy kanał IBM MQ dla WCF można utworzyć na jeden z dwóch sposobów.

### O tym zadaniu

Kanał niestandardowy produktu IBM MQ jest zintegrowany z produktem WCF jako kanał transportowy WCF i jako taki musi być sparowany z koderem komunikatów i opcjonalnymi kanałami protokołu, dzięki czemu może utworzyć pełny stos kanałów, który może być używany przez aplikację. Do pomyślnego utworzenia pełnego stosu kanału wymagane są dwa elementy:

1. Definicja powiązania: określa, które elementy są wymagane do zbudowania stosu kanału aplikacji, w tym kanał transportowy, koder komunikatów i wszystkie protokoły, a także wszystkie ogólne ustawienia konfiguracyjne. W przypadku kanału niestandardowego należy utworzyć definicję powiązania w postaci powiązania niestandardowego WCF.
2. Definicja punktu końcowego: łączy kontrakt usługi z definicją powiązania, a także udostępnia rzeczywisty identyfikator URI połączenia, który opisuje miejsce, z którym aplikacja może nawiązać połączenie. W przypadku kanału niestandardowego identyfikator URI ma postać identyfikatora URI protokołu SOAP korzystającego z protokołu JMS .

Definicje te można utworzyć na jeden z dwóch sposobów:

- Administracyjnie. Definicje są tworzone przez podanie szczegółów w pliku konfiguracyjnym aplikacji (na przykład: `app.config` ).
- Programowo. Definicje są tworzone bezpośrednio na podstawie kodu aplikacji.

Decyzja o tym, która metoda ma być używana do tworzenia definicji, musi być oparta na wymaganiach aplikacji w następujący sposób:

- Metoda administracyjna konfiguracji zapewnia elastyczność umożliwiającą zmianę szczegółów usługi i klienta po wdrożeniu bez konieczności odbudowywania aplikacji.
- Metoda programowa konfiguracji zapewnia większą ochronę przed błędami konfiguracji oraz możliwość dynamicznego generowania konfiguracji w czasie wykonywania.

### *Administracyjne tworzenie kanału niestandardowego WCF przez podanie informacji o powiązaniu i punkcie końcowym w pliku konfiguracyjnym aplikacji*

Kanał niestandardowy produktu IBM MQ dla produktu WCF jest kanałem WCF na poziomie transportu. Aby używać kanału niestandardowego, należy zdefiniować punkt końcowy i powiązanie. Definicje te można wykonać, podając informacje o powiązaniu i punkcie końcowym w pliku konfiguracyjnym aplikacji.

Aby skonfigurować i używać kanału niestandardowego IBM MQ dla produktu WCF, który jest kanałem WCF na poziomie transportu, należy zdefiniować powiązanie i definicję punktu końcowego. Powiązanie przechowuje informacje o konfiguracji kanału, a definicja punktu końcowego zawiera szczegóły połączenia. Te definicje można utworzyć na dwa sposoby:

- Programowo bezpośrednio z kodu aplikacji, zgodnie z następującym opisem: “Tworzenie niestandardowego kanału WCF przez programowe przekazywanie informacji o powiązaniu i punkcie końcowym” na stronie 1314
- Administracyjnie, poprzez podanie szczegółów w pliku konfiguracyjnym aplikacji, zgodnie z opisem w poniższej procedurze.

Plik konfiguracyjny aplikacji klienckiej lub usługowej ma zwykle nazwę *yourappname*`.exe.config` , gdzie *nazwa_środowiska* jest nazwą aplikacji. Plik konfiguracyjny aplikacji można łatwo zmodyfikować przy użyciu narzędzia edytora konfiguracji usługi Microsoft o nazwie `SvcConfigEditor.exe` w następujący sposób:

- Uruchom narzędzie edytora konfiguracji `SvcConfigEditor.exe` . Domyślnym miejscem instalacji narzędzia jest: *`Drive:`*`\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcConfigEditor.exe` , gdzie *Drive:* jest nazwą napędu instalacyjnego.

### Krok 1: Dodaj rozszerzenie elementu powiązania, aby umożliwić WCF znalezienie kanału niestandardowego

1. Kliknij prawym przyciskiem myszy opcję **Zaawansowane** > **Rozszerzenie** > **element powiązania** , aby otworzyć menu, a następnie wybierz opcję **Nowe** .
2. Wypełnij pola zgodnie z opisem w poniższej tabeli:

| *Tabela 190. Pola nowego elementu powiązania* | |
|---|---|
| **Pole** | **Wartość** |
| **Nazwa** | `IBM.XMS.WCF.SoapJmsIbmTransportChannel` |
| **Typ** | Przejdź do katalogu `IBM.XMS.WCF.dll` w globalnej pamięci podręcznej zespołu poprawek (Global Assembly Cache-GAC) i wybierz opcję `IBM.XMS.WCFSoapJmsIbmTransportBindingElementConfig` . |

### Krok 2: tworzenie niestandardowej definicji powiązania, która tworzy parę kanału niestandardowego z koderem komunikatów WCF

1. Kliknij prawym przyciskiem myszy opcję **Powiązania** , aby otworzyć menu, a następnie wybierz opcję **Nowa konfiguracja powiązania** .
2. Wypełnij pola zgodnie z opisem w poniższej tabeli:

| *Tabela 191. Nowe pola konfiguracji powiązania* | |
|---|---|
| **Pole** | **Wartość** |
| **Nazwa** | `CustomBinding_WMQ` |
| **BindingElement 1** | `textMessageEncoding (MessageVersion: Soap11)` |
| **BindingElement 2** | `IBM.XMS.WCF.SoapJmsIbmTransportChannel` |

### Krok 3: Określanie właściwości powiązania

1. Wybierz *IBM.XMS.WCF.SoapJmsIbmTransportChannel* z powiązania utworzonego w: “Krok 2: tworzenie niestandardowej definicji powiązania, która tworzy parę kanału niestandardowego z koderem komunikatów WCF” na stronie 1313
2. Wprowadź wymagane zmiany w wartościach domyślnych właściwości zgodnie z opisem w sekcji “Opcje konfiguracyjne powiązania WCF” na stronie 1317 .

### Krok 4: tworzenie definicji punktu końcowego

Utwórz definicję punktu końcowego, która odwołuje się do powiązania niestandardowego utworzonego w sekcji “Krok 2: tworzenie niestandardowej definicji powiązania, która tworzy parę kanału niestandardowego z koderem komunikatów WCF” na stronie 1313 i udostępnia szczegóły połączenia usługi. Sposób określania tych informacji zależy od tego, czy definicja dotyczy aplikacji klienckiej, czy aplikacji usługowej.

W przypadku aplikacji klienckiej dodaj definicję punktu końcowego do sekcji klienta w następujący sposób:

1. Kliknij prawym przyciskiem myszy opcję **Klient** > **Punkty końcowe** , aby otworzyć menu, a następnie wybierz opcję **Nowy punkt końcowy klienta** .
2. Wypełnij pola zgodnie z opisem w poniższej tabeli:

| *Tabela 192. Pola nowego punktu końcowego klienta* | |
|---|---|
| **Pole** | **Wartość** |
| **Nazwa** | `Endpoint_WMQ` |

| *Tabela 192. Pola nowego punktu końcowego klienta (kontynuacja)* | |
|---|---|
| **Pole** | **Wartość** |
| **Adres** | *Identyfikator URI SOAP/JMS opisujący szczegóły połączenia WMQ wymagane do uzyskania dostępu do usługi.* Więcej informacji na ten temat można znaleźć pod adresem: "Niestandardowy kanał IBM MQ dla formatu adresu URI punktu końcowego WCF" na stronie 1316 |
| **Łączy** | `customBinding` |
| **BindingConfiguration** | `CustomBinding_WMQ` |
| **Contract (kontrakt)** | *Nazwa interfejsu umowy o świadczenie usług* |

W przypadku aplikacji usługi dodaj definicję usługi do sekcji usług w następujący sposób:

1. Kliknij prawym przyciskiem myszy opcję **Usługi** , aby otworzyć menu, a następnie wybierz opcję **Nowa usługa**, a następnie wybierz klasę usługi, która ma być udostępniana.
2. Dodaj definicję punktu końcowego do sekcji **Punkty końcowe** dla nowej usługi i wypełnij pola w sposób przedstawiony w poniższej tabeli:

| *Tabela 193. Pola nowego punktu końcowego usługi* | |
|---|---|
| **Pole** | **Wartość** |
| **Nazwa** | `Endpoint_WMQ` |
| **Adres** | *Identyfikator URI SOAP/JMS opisujący szczegóły połączenia WMQ wymagane do uzyskania dostępu do usługi.* Więcej informacji na ten temat można znaleźć pod adresem: "Niestandardowy kanał IBM MQ dla formatu adresu URI punktu końcowego WCF" na stronie 1316 |
| **Łączy** | `customBinding` |
| **BindingConfiguration** | `CustomBinding_WMQ` |
| **Contract (kontrakt)** | *Nazwa klasy implementacji usługi* |

### *Tworzenie niestandardowego kanału WCF przez programowe przekazywanie informacji o powiązaniu i punkcie końcowym*

Kanał niestandardowy produktu IBM MQ dla produktu WCF jest kanałem WCF na poziomie transportu. Punkt końcowy i powiązanie muszą być zdefiniowane w celu użycia kanału niestandardowego, a te definicje można wykonać programowo bezpośrednio z poziomu kodu aplikacji.

Aby skonfigurować i używać kanału niestandardowego IBM MQ dla produktu WCF, który jest kanałem WCF na poziomie transportu, należy zdefiniować powiązanie i definicję punktu końcowego. Powiązanie przechowuje informacje o konfiguracji kanału, a definicja punktu końcowego zawiera szczegóły połączenia. Więcej informacji na ten temat zawiera sekcja "Korzystanie z przykładów WCF" na stronie 1325.

Te definicje można utworzyć na dwa sposoby:

- Administracyjnie, podając szczegółowe informacje w pliku konfiguracyjnym aplikacji, zgodnie z opisem w sekcji "Administracyjne tworzenie kanału niestandardowego WCF przez podanie informacji o powiązaniu i punkcie końcowym w pliku konfiguracyjnym aplikacji" na stronie 1312.
- Programowo bezpośrednio z kodu aplikacji, zgodnie z opisem w poniższych podtematach.

*Programowe definiowanie informacji o powiązaniu i punkcie końcowym: interfejs SOAP/JMS*
W przypadku interfejsu SOAP/JMS można zdefiniować punkt końcowy i powiązanie programowo bezpośrednio z kodu aplikacji.

## O tym zadaniu

Aby programowo dostarczyć informacje o powiązaniu i punkcie końcowym, dodaj wymagany kod do aplikacji, wykonując następujące kroki.

## Procedura

1. Utwórz instancję elementu powiązania transportu kanału, dodając do aplikacji następujący kod:

   ```
   SoapJmsIbmTransportBindingElement transportBindingElement = new
   SoapJmsIbmTransportBindingElement();
   ```

2. Ustaw wszystkie wymagane właściwości powiązania, na przykład dodając do aplikacji następujący kod w celu ustawienia właściwości `ClientConnectionMode`:

   ```
   transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
   ```

3. Utwórz powiązanie niestandardowe, które łączy kanał transportowy z koderem komunikatów, dodając do aplikacji następujący kod:

   ```
   Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
   transportBindingElement);
   ```

4. Utwórz identyfikator URI SOAP/JMS .

   Identyfikator URI SOAP/JMS , który opisuje szczegóły połączenia IBM MQ wymagane w celu uzyskania dostępu do usługi, musi być podany jako adres punktu końcowego. Podany adres zależy od tego, czy kanał jest używany dla aplikacji usługowej, czy dla aplikacji klienckiej.

   - W przypadku aplikacji klienckich identyfikator URI SOAP/JMS musi zostać utworzony jako `EndpointAddress` w następujący sposób:

     ```
     EndpointAddress address = new EndpointAddress("jms:/queue?
     destination=SampleQ@QM1&connectionFactory
     =connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
     ```

   - W przypadku aplikacji usług identyfikator URI SOAP/JMS musi zostać utworzony jako identyfikator URI w następujący sposób:

     ```
     Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=
     connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
     ```

     Więcej informacji na temat adresów punktów końcowych zawiera sekcja “Niestandardowy kanał IBM MQ dla formatu adresu URI punktu końcowego WCF” na stronie 1316.

*Programowe definiowanie informacji o powiązaniu i punkcie końcowym: interfejs inny niż SOAP/inny niżJMS*
W przypadku interfejsu innego niż SOAP/Non-JMS można zdefiniować punkt końcowy i powiązanie programowo bezpośrednio z kodu aplikacji.

## O tym zadaniu

Aby programowo dostarczyć informacje o powiązaniu i punkcie końcowym, dodaj wymagany kod do aplikacji, wykonując następujące kroki.

## Procedura

1. Utwórz powiązanie WmqBinding , dodając do aplikacji następujący kod:

   ```
   WmqBinding binding = new WmqBinding();
   ```

   Ten kod tworzy powiązanie, które tworzy parę elementów WmqMsgEncodingElement i WmqIbmTransportBindingwymaganych dla interfejsu innego niż SOAP/Non-JMS .
2. Podaj identyfikator URI wmq: // opisujący szczegóły połączenia IBM MQ wymagane do uzyskania dostępu do usługi.

   Sposób podawania identyfikatora URI wmq: // zależy od tego, czy kanał jest używany na potrzeby aplikacji usługi, czy aplikacji klienckiej.

   - W przypadku aplikacji klienckich identyfikator URI wmq: // musi zostać utworzony jako EndpointAddress w następujący sposób:

     ```
     EndpointAddress address = new EndpointAddress
     ("wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
     ```

   - W przypadku aplikacji usług identyfikator URI wmq: // musi zostać utworzony jako identyfikator URI w następujący sposób:

     ```
     Uri sampleAddress = new Uri(
     "wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
     ```

### *Niestandardowy kanał IBM MQ dla formatu adresu URI punktu końcowego WCF*

Usługa Web Service jest określana przy użyciu identyfikatora URI (Universal Resource Identifier), który udostępnia szczegóły położenia i połączenia. Format identyfikatora URI zależy od tego, czy używany jest interfejs SOAP/JMS , czy interfejs inny niż SOAP/inny niżJMS .

## Interfejs SOAP/JMS

Format identyfikatora URI, który jest obsługiwany w transporcie IBM MQ dla protokołu SOAP, umożliwia pełną kontrolę nad parametrami i opcjami specyficznymi dla protokołu SOAP/ IBM MQ podczas uzyskiwania dostępu do usług docelowych. Ten format jest zgodny z produktem WebSphere Application Server i produktem CICS, co ułatwia integrację produktu IBM MQ z obydwoma tymi produktami.

Składnia identyfikatora URI jest następująca:

```
jms:/queue? name=value&name=value...
```

gdzie nazwa jest nazwą parametru, a *wartość* jest odpowiednią wartością, a element nazwa = *wartość* może być powtarzany dowolną liczbę razy, przy czym drugie i kolejne wystąpienia są poprzedzone znakiem ampersand (&).

W nazwach parametrów rozróżniana jest wielkość liter, podobnie jak w nazwach obiektów IBM MQ . Jeśli dowolny parametr zostanie określony więcej niż raz, końcowe wystąpienie parametru zostanie zastosowane, co oznacza, że aplikacje klienckie mogą przesłonić wartości parametrów, dołączając je do identyfikatora URI. Jeśli zostaną dołączone dodatkowe nierozpoznane parametry, zostaną one zignorowane.

Jeśli identyfikator URI jest przechowywany w łańcuchu XML, znak ampersand musi być reprezentowany jako "&". Podobnie, jeśli identyfikator URI jest zakodowany w skrypcie, należy zachować ostrożność podczas zmiany znaczenia znaków, takich jak **&** , które w przeciwnym razie byłyby interpretowane przez powłokę.

Oto przykład prostego identyfikatora URI dla usługi Axis:

```
jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Poniżej przedstawiono przykład prostego identyfikatora URI dla usługi .NET :

```
jms:/queue?destination=myQ&connectionFactory=()&targetService=MyService.asmx
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Podawane są tylko wymagane parametry (parametr `targetService` jest wymagany tylko dla usług systemu .NET ), a parametr `connectionFactory` nie ma opcji.

W tym przykładzie na osi `connectionFactory` znajduje się kilka opcji:

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

W tym przykładzie środowiska Axis podano również opcję `sslPeerName` w fabryce `connectionFactory` . Wartość nazwy sslPeerzawiera pary nazwa-wartość i znaczące odstępy wewnętrzne:

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
sslPeerName(CN=MQ Test 1,O=IBM,S=Hampshire,C=GB)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

## Interfejs NON-SOAP/Non-JMS

Format identyfikatora URI dla interfejsu NON-SOAP/Non-JMS umożliwia pełną kontrolę nad parametrami i opcjami specyficznymi dla produktu IBM MQ podczas uzyskiwania dostępu do usług docelowych.

Składnia identyfikatora URI jest następująca:

```
wmq://example.com:1415/msg/queue/INS.QUOTE.REQUEST@MOTOR.INS ?ReplyTo=msg/queue/
INS.QUOTE.REPLY@BRANCH452&persistence=MQPER_NOT_PERSISTENT
```

Ta poprawka IRI informuje requester usług, że może on użyć połączenia powiązania klienta TCP IBM MQ TCP z komputerem o nazwie example.com na porcie 1415 i umieścić komunikaty żądań trwałych w kolejce o nazwie INS.QUOTE.REQUEST w menedżerze kolejek MOTOR.INS. IIRI określa, że dostawca usług umieszcza odpowiedzi w kolejce o nazwie INS.QUOTE.REPLY w menedżerze kolejek BRANCH452. Format identyfikatora URI jest taki sam, jak określony dla pakietu SupportPac MA93. Więcej informacji na temat specyfikacji IRI systemu IBM MQ zawiera publikacja SupportPac MA93: IBM MQ -Service Definition .

### *Opcje konfiguracyjne powiązania WCF*

Istnieją dwa sposoby stosowania opcji konfiguracyjnych do informacji o powiązaniach kanałów niestandardowych. Właściwości można ustawić administracyjnie lub programowo.

Opcje konfiguracji powiązania można ustawić na jeden z dwóch sposobów:

1. Administracyjnie: ustawienia właściwości powiązania muszą być określone w sekcji transportu definicji powiązania niestandardowego w pliku konfiguracyjnym aplikacji, na przykład: `app.config`.
2. Programowo: kod aplikacji musi zostać zmodyfikowany w celu określenia właściwości podczas inicjowania powiązania niestandardowego.

## Administracyjne ustawianie właściwości powiązania

Ustawienia właściwości powiązania można określić w pliku konfiguracyjnym aplikacji, na przykład: `app.config`. Plik konfiguracyjny jest generowany przez **`svcutil`**, jak pokazano w poniższych przykładach.

**Interfejs SOAP/JMS**

```
<customBinding>
...
      <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>
...
</customBinding>
```

**Interfejs inny niż SOAP/inny niżJMS**

```
<customBinding>
      <IBM.WMQ.WCF.WmqMsgEncodingElement/>
      <IBM.WMQ.WCF.WmqIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="65536" clientConnectionMode="managedclient"/>
</customBinding>
```

## Programowe ustawianie właściwości powiązania

Aby dodać właściwość powiązania WCF w celu określenia trybu połączenia klienta, należy zmodyfikować kod usługi w celu określenia właściwości podczas inicjowania powiązania niestandardowego.

Aby określić tryb połączenia niezarządzanego klienta, należy użyć następującego przykładu:

```
SoapJmsIbmTransportBindingElement
transportBindingElement = new SoapJmsIbmTransportBindingElement();
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.CLIENT_UNMANAGED;

Binding sampleBinding = new CustomBinding(new TextMessageEncodingBindingElement(),
                                                    transportBindingElement);
```

## Właściwości powiązania WCF

*Tabela 194. Wartości właściwości powiązania podczas ustawiania administracyjnego lub programowego*

| Nazwa właściwości | Aplikacja klienta lub usługi | Wartość administracyjna | Wartość programowa | Opis |
|---|---|---|---|---|
| maxBufferPoolSize | Obie | Od 0 do 64 bitowej liczby całkowitej ze znakiem | Od 0 do 64 bitowej liczby całkowitej ze znakiem | Określa maksymalną wielkość pamięci, która może być używana do przechowywania buforów komunikatów WCF dla instancji kanału. |
| maxMessageWielkość | Obie | Od 1 do 32 bitowej liczby całkowitej ze znakiem | Od 1 do 32 bitowej liczby całkowitej ze znakiem | Określa maksymalną ilość pamięci, która może być używana dla pojedynczego komunikatu WCF. |

*Tabela 194. Wartości właściwości powiązania podczas ustawiania administracyjnego lub programowego (kontynuacja)*

| Nazwa właściwości | Aplikacja klienta lub usługi | Wartość administracyjna | Wartość programowa | Opis |
|---|---|---|---|---|
| Tryb clientConnection | Obie | 0 (wartość domyślna)<br>1 | AS_URI (wartość domyślna)<br>NIEZARZĄDZANY_KLIENT | Określa tryb połączenia klienta kanału transportowego.<br>0 oznacza, że tryb połączenia klienta jest określony w identyfikatorze URI. Używany tylko wtedy, gdy używane jest połączenie klienta. Określa, że tryb połączenia klienta jest określony w identyfikatorze URI. Jeśli nie ustawiono trybu połączenia klienta, wartością domyślną jest 0.<br>1 oznacza, że tryb połączenia klienta jest klientem niezarządzanym. Używany tylko wtedy, gdy używane jest połączenie klienta. |
| MaxConcurrentwywołań | Klient | Zakres obejmuje wartości od 0 do 2 147 483 647<br>16 jest wartością domyślną | Zakres obejmuje wartości od 0 do 2 147 483 647<br>16 jest wartością domyślną | Ta właściwość definiuje maksymalną liczbę współbieżnych operacji, które mogą być wykonywane jednocześnie na pojedynczym serwerze proxy klienta. Jeśli uruchomiono więcej operacji, są one umieszczane w kolejce do momentu zakończenia lub upłynięcia limitu czasu trwającej operacji. To ustawienie może być używane do sterowania maksymalną liczbą wątków i zasobów, które mogą być używane przez pojedynczy serwer proxy.<br>0 usuwa ten limit, umożliwiając współbieżne wykonywanie wszystkich operacji. |

*Tabela 194. Wartości właściwości powiązania podczas ustawiania administracyjnego lub programowego (kontynuacja)*

| Nazwa właściwości | Aplikacja klienta lub usługi | Wartość administracyjna | Wartość programowa | Opis |
|---|---|---|---|---|
| MaxConcurrentwywołań | Usługa | Zakres obejmuje liczby od 1 do 2 147 483 647<br>16 jest wartością domyślną | Zakres obejmuje liczby od 1 do 2 147 483 647<br>16 jest wartością domyślną | Ta właściwość jest używana tylko wtedy, gdy włączona jest funkcja gwarantowanego dostarczania (więcej informacji na temat gwarantowanego dostarczania zawiera sekcja “Gwarantowane dostarczenie kanału niestandardowego WCF” na stronie 1307 ). Określa ona maksymalną liczbę współbieżnych operacji, które mogą być w tym samym czasie wykonywane dla danego punktu końcowego.<br>Podczas zmiany tego ustawienia należy zachować ostrożność. Każda współbieżna operacja wymaga dodatkowych zasobów, w szczególności nowej instancji kanału niestandardowego i powiązanych wątków z puli wątków w celu wykonania działań na żądaniach. Nadmierna alokacja może być szkodliwym działaniem i mieć poważny wpływ na wydajność. Aby ta właściwość była obsługiwana, należy odpowiednio skonfigurować pulę wątków. |

## Usługi budowlane i hostingowe dla WCF

Przegląd usług Microsoft Windows Communication Foundation (WCF) z wyjaśnieniem sposobu tworzenia i konfigurowania usług WCF.

Kanał niestandardowy produktu IBM MQ dla usług WCF i usług WCF, które z niego korzystają, może być udostępniany za pomocą następujących metod:

- Self-hosting
- Usługa Windows

Kanał niestandardowy IBM MQ dla WCF nie może być udostępniany w usłudze Windows Process Activation Service.

Poniższe tematy zawierają kilka prostych przykładów samoobsługowych, które ilustrują wykonywane kroki. Dokumentacja elektroniczna produktu Microsoft WCF, która zawiera dalsze informacje i najnowsze informacje, jest dostępna w serwisie WWW Microsoft MSDN pod adresem https://msdn.microsoft.com.

### *Budowanie aplikacji usług WCF przy użyciu metody 1: Samoobsługa administracyjna przy użyciu pliku konfiguracyjnego aplikacji*

Po utworzeniu pliku konfiguracyjnego aplikacji otwórz instancję usługi i dodaj określony kod do aplikacji.

## Zanim rozpoczniesz

Utwórz lub zmodyfikuj plik konfiguracyjny aplikacji dla usługi zgodnie z opisem w sekcji “Administracyjne tworzenie kanału niestandardowego WCF przez podanie informacji o powiązaniu i punkcie końcowym w pliku konfiguracyjnym aplikacji” na stronie 1312 .

## O tym zadaniu

1. Utwórz i otwórz instancję usługi w hoście usługi. Typ usługi musi być taki sam, jak typ usługi określony w pliku konfiguracyjnym usługi.
2. Dodaj do aplikacji następujący kod:

```
ServiceHost service = new ServiceHost(typeof(MyService));
service.Open();
...
service.Close();
```

### *Budowanie aplikacji usługowych WCF przy użyciu metody 2: Samoobsługa programowa bezpośrednio z aplikacji*

Dodaj właściwości powiązania, utwórz host usługi z instancją wymaganej klasy usługi i otwórz usługę.

## Zanim rozpoczniesz

1. Dodaj do projektu odwołanie do pliku `IBM.XMS.WCF.dll` kanału niestandardowego. Katalog `IBM.XMS.WCF.dll` znajduje się w katalogu *`WMQInstallDir`*`\bin` , gdzie *WMQInstallDir* jest katalogiem, w którym zainstalowano produkt IBM MQ .
2. Dodaj instrukcję *using* do przestrzeni nazw `IBM.XMS.WCF` , na przykład: `using IBM.XMS.WCF`
3. Utwórz instancję elementu powiązania kanałów i punktu końcowego w sposób opisany w sekcji “Tworzenie niestandardowego kanału WCF przez programowe przekazywanie informacji o powiązaniu i punkcie końcowym” na stronie 1314

## O tym zadaniu

Jeśli wymagane są zmiany właściwości powiązania kanału, wykonaj następujące kroki:

1. Dodaj właściwości powiązania do pliku `transportBindingElement` , jak pokazano w poniższym przykładzie:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=

connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. Utwórz host usługi z instancją wymaganej klasy usługi:

```
ServiceHost service = new ServiceHost(typeof(MyService));
```

3. Otwórz usługę:

```
service.AddServiceEndpoint(typeof(IMyServiceContract), binding, address);
service.Open();
...
service.Close();
```

### *Ujawnianie metadanych przy użyciu punktu końcowego HTTP*

Instrukcje dotyczące ujawniania metadanych usługi, która jest skonfigurowana do używania niestandardowego kanału IBM MQ dla WCF.

### O tym zadaniu

Jeśli metadane usług muszą zostać ujawnione (aby narzędzia, takie jak `svcutil` , mogły uzyskać do nich dostęp bezpośrednio z działającej usługi, a nie na przykład z pliku WSDL w trybie bez połączenia), należy to zrobić, ujawniając metadane usług za pomocą punktu końcowego HTTP . Aby dodać ten dodatkowy punkt końcowy, można wykonać następujące kroki.

1. Dodaj adres bazowy określający miejsce, w którym metadane muszą być ujawniane dla hosta ServiceHost, na przykład:

```
ServiceHost service = new ServiceHost(typeof(TestService),
                      new Uri("http://localhost:8000/MyService"));
```

2. Dodaj następujący kod do ServiceHost przed otwarciem usługi:

```
ServiceMetadataBehavior metadataBehavior = new ServiceMetadataBehavior();
  metadataBehavior.HttpGetEnabled = true;
  service.Description.Behaviors.Add(metadataBehavior);
  service.AddServiceEndpoint(typeof(IMetadataExchange),
     MetadataExchangeBindings.CreateMexHttpBinding(), "mex");
```

### Wyniki

Metadane są teraz dostępne pod następującym adresem: `http://localhost:8000/MyService`

## Budowanie aplikacji klienckich dla środowiska WCF

Przegląd generowania i budowania aplikacji klienckich Microsoft Windows Communication Foundation (WCF).

Aplikację kliencką można utworzyć dla usługi WCF. Aplikacje klienckie są zwykle generowane za pomocą narzędzia narzędziowego metadanych Microsoft ServiceModel (Svcutil.exe) w celu utworzenia wymaganych plików konfiguracyjnych i plików proxy, które mogą być używane bezpośrednio przez aplikację.

### *Generowanie plików konfiguracyjnych aplikacji i proxy klienta WCF przy użyciu narzędzia svcutil z metadanymi z działającej usługi*

Instrukcje dotyczące korzystania z narzędzia Microsoft svcutil.exe w celu wygenerowania klienta dla usługi, która jest skonfigurowana do używania niestandardowego kanału IBM MQ dla WCF.

### Zanim rozpoczniesz

Istnieją trzy wymagania wstępne dotyczące używania narzędzia svcutil do tworzenia wymaganych plików konfiguracyjnych i plików proxy, które mogą być używane bezpośrednio przez aplikację:

- Usługa WCF musi być uruchomiona przed uruchomieniem narzędzia svcutil.
- Usługa WCF musi ujawniać swoje metadane przy użyciu portu HTTP oprócz odwołań do niestandardowych punktów końcowych kanału produktu IBM MQ w celu wygenerowania klienta bezpośrednio z działającej usługi.
- Kanał niestandardowy musi być zarejestrowany w danych konfiguracyjnych dla svcutil.

### O tym zadaniu

W poniższych krokach wyjaśniono sposób generowania klienta dla usługi, która jest skonfigurowana do korzystania z kanału niestandardowego IBM MQ , ale także przedstawiono jej metadane w czasie wykonywania za pośrednictwem oddzielnego portu HTTP :

1. Uruchom usługę WCF (usługa musi być uruchomiona przed uruchomieniem narzędzia svcutil).
2. Dodaj szczegóły z pliku konfiguracyjnego `svcutil.exe` z katalogu głównego instalacji do aktywnego pliku konfiguracyjnego svcutil, zwykle `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config` , aby program svcutil rozpoznał kanał niestandardowy IBM MQ .

3. Uruchom svcutil z wiersza komend, na przykład:

```
svcutil /language:C# /r: installlocation\bin\IBM.XMS.WCF.dll
/config:app.config http://localhost:8000/IBM.XMS.WCF/samples
```

4. Skopiuj wygenerowane pliki `app.config` i `YourService.cs` do projektu klienta Microsoft Visual Studio.

## Co dalej

Jeśli nie można bezpośrednio pobrać metadanych usług, można użyć narzędzia svcutil do wygenerowania plików klienta z pliku WSDL. Więcej informacji na ten temat można znaleźć pod adresem: “Generowanie plików konfiguracyjnych aplikacji i proxy klienta WCF przy użyciu narzędzia svcutil z plikiem WSDL” na stronie 1323

### *Generowanie plików konfiguracyjnych aplikacji i proxy klienta WCF przy użyciu narzędzia svcutil z plikiem WSDL*

Instrukcje generowania klientów WCF z pliku WSDL, jeśli metadane usługi są niedostępne.

Jeśli nie można bezpośrednio pobrać metadanych usługi w celu wygenerowania klienta na podstawie metadanych działającej usługi, można użyć narzędzia svcutil do wygenerowania plików klienta z pliku WSDL. W pliku WSDL należy wprowadzić następujące modyfikacje, aby określić, że ma być używany kanał niestandardowy IBM MQ :

1. Dodaj następujące definicje przestrzeni nazw i informacje o strategii:

```
<wsdl:definitions
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">

        <wsp:Policy wsu:Id="CustomBinding_IWMQSampleContract_policy">
            <wsp:ExtactlyOne>
                <wsp:All>
                    <xms:xms xmlns:xms="http://sample.schemas.ibm.com/policy/xms" />
                </wsp:All>
            </wsp:ExactlyOne>
        </wsp:Policy>

...

</wsdl:definitions>
```

2. Zmodyfikuj sekcję powiązań, aby odwoływał się do nowej sekcji strategii, i usuń wszystkie definicje `transport` z bazowego elementu powiązania:

```
<wsdl:definitions ...>

        <wsdl:binding ...>
                <wsp:PolicyReference URI="#CustomerBinding_IWMQSampleContract_policy" />
                <[soap]:binding ... transport="" />
                ...
        </wsdl:binding>
</wsdl:definitions>
```

3. Uruchom svcutil z wiersza komend, na przykład:

```
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
```

### *Budowanie aplikacji klienckich WCF przy użyciu serwera proxy klienta z plikiem konfiguracyjnym aplikacji*

## Zanim rozpoczniesz

Utwórz lub zmodyfikuj plik konfiguracyjny aplikacji dla klienta zgodnie z opisem w sekcji “Administracyjne tworzenie kanału niestandardowego WCF przez podanie informacji o powiązaniu i punkcie końcowym w pliku konfiguracyjnym aplikacji” na stronie 1312 .

## O tym zadaniu

Utwórz i otwórz instancję serwera proxy klienta. Parametr przekazywany do wygenerowanego serwera proxy musi być taki sam jak nazwa punktu końcowego określona w pliku konfiguracyjnym klienta, na przykład `Endpoint_WMQ`:

```
MyClientProxy myClient = new MyClientProxy("Endpoint_WMQ");
            try {
                   myClient.myMethod("HelloWorld!");
                   myClient.Close();
            }
            catch (TimeoutException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (CommunicationException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (Exception e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
```

# *Budowanie aplikacji klienckich WCF przy użyciu proxy klienta z konfiguracją programową*

## Zanim rozpoczniesz

1. Dodaj do projektu odwołanie do pliku `IBM.XMS.WCF.dll` kanału niestandardowego. Katalog `IBM.XMS.WCF.dll` znajduje się w katalogu *`WMQInstallDir`*`\bin` , gdzie *WMQInstallDir* jest katalogiem, w którym zainstalowano produkt IBM MQ .
2. Dodaj instrukcję *using* do przestrzeni nazw `IBM.XMS.WCF` , na przykład: `using IBM.XMS.WCF`
3. Utwórz instancję elementu powiązania i punktu końcowego kanału w sposób opisany w sekcji “Tworzenie niestandardowego kanału WCF przez programowe przekazywanie informacji o powiązaniu i punkcie końcowym” na stronie 1314

## O tym zadaniu

Jeśli wymagane są zmiany właściwości powiązania kanału, wykonaj następujące kroki.

1. Dodaj właściwości powiązania do pliku `transportBindingElement` , jak pokazano na poniższym rysunku:

```
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. Utwórz proxy klienta, jak pokazano na poniższym rysunku, gdzie *powiązanie* i *adres punktu końcowego* to powiązanie i adres punktu końcowego skonfigurowane w kroku 1 i przekazane:

```
 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();
            }
```

# Korzystanie z przykładów WCF

Przykłady produktu Windows Communication Foundation (WCF) zawierają kilka prostych przykładów użycia niestandardowego kanału IBM MQ .

Do budowania przykładowych projektów wymagany jest pakiet Microsoft.NET 3.5 SDK lub Microsoft Visual Studio 2008.

## Przykład prostego jednokierunkowego klienta i serwera WCF

W tym przykładzie przedstawiono kanał niestandardowy IBM MQ używany do uruchamiania usługi Windows Communication Foundation (WCF) z klienta WCF przy użyciu jednokierunkowego kształtu kanału.

### O tym zadaniu

Usługa implementuje pojedynczą metodę, która wyprowadza łańcuch do konsoli. Klient został wygenerowany przy użyciu narzędzia `svcutil` w celu pobrania metadanych usługi z oddzielnie ujawnionego punktu końcowego HTTP zgodnie z opisem w sekcji "Generowanie plików konfiguracyjnych aplikacji i proxy klienta WCF przy użyciu narzędzia svcutil z metadanymi z działającej usługi" na stronie 1322

Przykład został skonfigurowany przy użyciu konkretnych nazw zasobów zgodnie z opisem w poniższej procedurze. Jeśli konieczna jest zmiana nazw zasobów, należy również zmienić odpowiednią wartość w aplikacji klienckiej w pliku *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\app.config` oraz w aplikacji serwisowej w pliku *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\TestServices.cs` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym produktu IBM MQ. Więcej informacji na temat formatowania identyfikatora URI punktu końcowego JMS zawiera sekcja *IBM MQ Transport dla SOAP* w dokumentacji produktu IBM MQ . Jeśli konieczne jest zmodyfikowanie przykładowego rozwiązania i źródła, potrzebne jest środowisko IDE, na przykład Microsoft Visual Studio 8 lub nowsze.

### Procedura

1. Utwórz menedżer kolejek o nazwie *QM1* .
2. Utwórz miejsce docelowe kolejki o nazwie *SampleQ* .
3. Uruchom usługę, aby nasłuchiwanie oczekiwało na komunikaty: uruchom plik *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\bin\Release\TestService.exe` , gdzie *MQ_INSTALLATION_PATH* to katalog instalacyjny IBM MQ.
4. Uruchom klienta raz: uruchom plik *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\bin\Release\TestClient.exe` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym IBM MQ.

   Aplikacja kliencka zapętla się pięć razy, wysyłając pięć komunikatów do serwera *SampleQ* .

### Wyniki

Aplikacja usługowa pobiera komunikaty z komendy *SampleQ* i wyświetla na ekranie komunikat `Hello World` pięć razy.

### Co dalej

## Prosty przykład WCF klienta i serwera żądanie-odpowiedź

Ten przykład przedstawia kanał niestandardowy IBM MQ używany do uruchamiania usługi Windows Communication Foundation (WCF) z klienta WCF przy użyciu kształtu kanału żądanie-odpowiedź.

### O tym zadaniu

Ta usługa udostępnia kilka prostych metod kalkulatora, które dodają i odejmują dwie liczby, a następnie zwracają wynik. Klient został wygenerowany przy użyciu narzędzia `svcutil` w celu pobrania metadanych usługi z oddzielnie ujawnionego punktu końcowego HTTP zgodnie z opisem w sekcji "Generowanie plików konfiguracyjnych aplikacji i proxy klienta WCF przy użyciu narzędzia svcutil z metadanymi z działającej usługi" na stronie 1322

Przykład został skonfigurowany przy użyciu konkretnych nazw zasobów, tak jak opisano w poniższej procedurze. Jeśli konieczna jest zmiana nazw zasobów, należy również zmienić odpowiednią wartość w aplikacji klienckiej w pliku *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\client\app.config` oraz w aplikacji serwisowej w pliku *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\service\RequestReplyService.cs` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym produktu IBM MQ. Więcej informacji na temat formatowania identyfikatora URI punktu końcowego JMS zawiera sekcja *IBM MQ Transport dla SOAP* w dokumentacji produktu IBM MQ . Jeśli konieczne jest zmodyfikowanie przykładowego rozwiązania i źródła, potrzebne jest środowisko IDE, na przykład Microsoft Visual Studio 8 lub nowsze.

### Procedura

1. Utwórz menedżer kolejek o nazwie *QM1* .
2. Utwórz miejsce docelowe kolejki o nazwie *SampleQ* .
3. Utwórz miejsce docelowe kolejki o nazwie *SampleReplyQ* .
4. Uruchom usługę, aby nasłuchiwanie oczekiwało na komunikaty: uruchom plik *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\service\bin\Release\SimpleRequestReply_Service.exe` , gdzie *MQ_INSTALLATION_PATH* to katalog instalacyjny IBM MQ.
5. Uruchom klienta raz: uruchom plik *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\client\bin\Release\SimpleRequestReply_Client.exe` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym IBM MQ.

### Wyniki

Po uruchomieniu klienta uruchamiany jest następujący proces, który jest powtarzany cztery razy, co powoduje wysłanie łącznie pięciu komunikatów w każdą stronę:

1. Klient umieszcza komunikat żądania w *SampleQ* i czeka na odpowiedź.
2. Usługa pobiera komunikat żądania z *SampleQ*.
3. Usługa dodaje i odejmuje niektóre wartości przy użyciu treści komunikatu.
4. Następnie usługa umieszcza wyniki w komunikacie w kolejce *SampleReplyQ*i oczekuje na umieszczenie nowego komunikatu przez klienta.
5. Klient otrzyma komunikat z kolejki *SampleReplyQ* i wyświetli wyniki na ekranie.

### Co dalej

## Klient WCF do usługi systemu .NET udostępnianej przez przykład IBM MQ

Przykładowe aplikacje klienckie i przykładowe aplikacje proxy usług są dostarczane zarówno dla .NET , jak i dla Java. Przykłady są oparte na usłudze Kursy akcji, która pobiera żądanie wyceny akcji, a następnie udostępnia wycenę akcji.

### Zanim rozpoczniesz

W tym przykładzie wymagane jest, aby środowisko usług serwerowych .NET SOAP over JMS było poprawnie zainstalowane i skonfigurowane w produkcie IBM MQ i było dostępne z poziomu lokalnego menedżera kolejek.

Jeśli środowisko usług serwerowych .NET SOAP over JMS jest poprawnie zainstalowane i skonfigurowane w produkcie IBM MQ i jest dostępne z poziomu lokalnego menedżera kolejek, należy wykonać dodatkowe kroki konfiguracyjne.

1. Ustaw zmienną środowiskową **WMQSOAP_HOME** na katalog instalacyjny IBM MQ , na przykład: `C:\Program Files\IBM\MQ`
2. Upewnij się, że Java kompilator `javac` jest dostępny i znajduje się w zmiennej PATH.
3. Skopiuj plik `axis.jar` z katalogu `prereqs/axis` obrazu instalacyjnego do katalogu produkcyjnego IBM MQ , na przykład: `C:\Program Files\IBM\MQ\java\lib\soap`
4. Dodaj do zmiennej PATH: *MQ_INSTALLATION_PATH*`\Java\lib` , gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog, w którym zainstalowano produkt IBM MQ , na przykład: `C:\Program Files\IBM\MQ`
5. Upewnij się, że położenie pliku .NET zostało poprawnie określone w pliku *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` , gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog, w którym zainstalowano produkt IBM MQ , na przykład: `C:\Program Files\IBM\MQ`. Położenie pliku .NET można określić na przykład: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

Po wykonaniu powyższych kroków przetestuj i uruchom usługę:

1. Przejdź do katalogu roboczego protokołu SOAP korzystającego z protokołu JMS .
2. Wprowadź jedną z następujących komend, aby uruchomić test weryfikacyjny i pozostawić uruchomione nasłuchiwanie usługi:
   - W systemie .NET: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt dotnet hold` , gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog, w którym zainstalowano produkt IBM MQ .
   - W przypadku kamery AXIS: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold` , gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog, w którym zainstalowano produkt IBM MQ .

Argument `hold` powoduje, że obiekty nasłuchiwania pozostają uruchomione po zakończeniu testu.

Jeśli podczas tej konfiguracji zostaną zgłoszone błędy, można usunąć wszystkie zmiany, aby procedura mogła zostać zrestartowana w następujący sposób:

1. Usuń wygenerowany katalog SOAP over JMS .
2. Usuń menedżer kolejek.

### O tym zadaniu

W tym przykładzie zademonstrowano połączenie klienta WCF z przykładową usługą .NET SOAP over JMS udostępnioną w produkcie IBM MQ przy użyciu jednokierunkowego kształtu kanału. Usługa implementuje prosty przykład StockQuote , który wyprowadza łańcuch tekstowy na konsolę.

Klient został wygenerowany przy użyciu pliku WSDL w celu wygenerowania plików klienta zgodnie z opisem w sekcji “Generowanie plików konfiguracyjnych aplikacji i proxy klienta WCF przy użyciu narzędzia svcutil z plikiem WSDL” na stronie 1323

Przykład został skonfigurowany przy użyciu konkretnych nazw zasobów zgodnie z opisem w poniższej procedurze. Jeśli konieczna jest zmiana nazw zasobów, należy również zmienić odpowiednią wartość

w aplikacji klienckiej w pliku *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQNET\default\client\app.config` oraz w aplikacji serwisowej w pliku *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQNET\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl` , gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog instalacyjny produktu IBM MQ. Więcej informacji na temat formatowania identyfikatora URI punktu końcowego JMS zawiera sekcja *IBM MQ Transport dla SOAP* w dokumentacji produktu IBM MQ .

### Procedura

Uruchom klienta raz: uruchom plik *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQNET\default\client\bin\Release\TestClient.exe` , gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog instalacyjny IBM MQ.

Aplikacja kliencka zapętla się pięć razy, wysyłając pięć komunikatów do kolejki przykładów.

### Wyniki

Aplikacja usługi pobiera komunikaty z przykładowej kolejki i wyświetla na ekranie komunikat `Hello World` pięć razy.

## Klient WCF do usługi Java środowiska Axis obsługiwanej przez przykład IBM MQ

Przykładowe aplikacje klienckie i przykładowe aplikacje proxy usług są dostarczane zarówno dla Java , jak i dla .NET. Przykłady są oparte na usłudze Kursy akcji, która pobiera żądanie wyceny akcji, a następnie udostępnia wycenę akcji.

### Zanim rozpoczniesz

Ten przykład wymaga, aby środowisko usług serwerowych .NET SOAP over JMS było poprawnie zainstalowane i skonfigurowane w produkcie IBM MQ oraz aby było dostępne z poziomu lokalnego menedżera kolejek.

Jeśli środowisko usług serwerowych .NET SOAP over JMS jest poprawnie zainstalowane i skonfigurowane w produkcie IBM MQ i jest dostępne z poziomu lokalnego menedżera kolejek, należy wykonać dodatkowe kroki konfiguracyjne.

1. Ustaw zmienną środowiskową **WMQSOAP_HOME** na katalog instalacyjny IBM MQ , na przykład: `C:\Program Files\IBM\MQ`
2. Upewnij się, że Java kompilator `javac` jest dostępny i znajduje się w zmiennej PATH.
3. Skopiuj plik `axis.jar` z katalogu `prereqs/axis` obrazu instalacyjnego do katalogu instalacyjnego IBM MQ .
4. Dodaj do zmiennej PATH: *MQ_INSTALLATION_PATH*`\Java\lib` , gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog, w którym zainstalowano produkt IBM MQ , na przykład: `C:\Program Files\IBM\MQ`
5. Upewnij się, że położenie pliku .NET zostało poprawnie określone w pliku *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` , gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog, w którym zainstalowano produkt IBM MQ , na przykład: `C:\Program Files\IBM\MQ`. Położenie pliku .NET można określić na przykład: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

Po wykonaniu powyższych kroków przetestuj i uruchom usługę:

1. Przejdź do katalogu roboczego protokołu SOAP korzystającego z protokołu JMS .
2. Wprowadź jedną z następujących komend, aby uruchomić test weryfikacyjny i pozostawić uruchomione nasłuchiwanie usługi:
   - W systemie .NET: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt dotnet hold` , gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog, w którym zainstalowano produkt IBM MQ .

- W przypadku kamery AXIS: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt Dotnet2AxisClient hold , gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog, w którym zainstalowano produkt IBM MQ .

Argument hold powoduje, że obiekty nasłuchiwania pozostają uruchomione po zakończeniu testu.

Jeśli podczas tej konfiguracji zostaną zgłoszone błędy, można usunąć wszystkie zmiany, aby procedura została zrestartowana w następujący sposób:

1. Usuń wygenerowany katalog SOAP over JMS .
2. Usuń menedżer kolejek.

### O tym zadaniu

Przykład demonstruje połączenie klienta WCF z przykładową usługą Axis Java SOAP over JMS udostępnianą w produkcie IBM MQ przy użyciu jednokierunkowego kształtu kanału. Usługa implementuje prosty przykład StockQuote , który wyprowadza łańcuch tekstowy do pliku zapisanego w katalogu bieżącym.

Klient został wygenerowany przy użyciu pliku WSDL w celu wygenerowania plików klienta zgodnie z opisem w sekcji "Generowanie plików konfiguracyjnych aplikacji i proxy klienta WCF przy użyciu narzędzia svcutil z plikiem WSDL" na stronie 1323

Przykład został skonfigurowany przy użyciu konkretnych nazw zasobów zgodnie z opisem w tym akapicie. Jeśli konieczna jest zmiana nazw zasobów, należy również zmienić odpowiednią wartość w aplikacji klienckiej w pliku *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\client\app.config oraz w aplikacji serwisowej w pliku *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl , gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog instalacyjny produktu IBM MQ.

### Procedura

Uruchom klienta raz: uruchom plik *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\client\bin\Release\TestClient.exe , gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog instalacyjny IBM MQ.

Aplikacja kliencka zapętla się pięć razy, wysyłając pięć komunikatów do kolejki przykładów.

### Wyniki

Aplikacja usługi pobiera komunikaty z przykładowej kolejki i pięciokrotnie dodaje łańcuch Hello World do pliku w katalogu bieżącym.

## Klient WCF do usługi Java udostępnianej przez przykład WebSphere Application Server

Przykładowe aplikacje klienckie i przykładowe aplikacje proxy usług są dostarczane dla produktu WebSphere Application Server 6. Udostępniana jest także usługa żądanie-odpowiedź.

### Zanim rozpoczniesz

W tym przykładzie wymagana jest następująca konfiguracja IBM MQ :

*Tabela 195. IBM MQ wymagana konfiguracja*

| **Obiekt** | **Wymagana nazwa** |
|---|---|
| Menedżer kolejek | QM1 |
| Kolejka lokalna | HelloWorld |
| Kolejka lokalna | HelloWorldOdpowiedz |

Ten przykład wymaga również poprawnego zainstalowania i skonfigurowania środowiska usług serwerowych WebSphere Application Server 6 . Produkt WebSphere Application Server 6 domyślnie używa połączenia w trybie powiązań do nawiązania połączenia z produktem IBM MQ . Dlatego produkt WebSphere Application Server 6 musi być zainstalowany na tym samym komputerze, co menedżer kolejek.

Po skonfigurowaniu środowiska WAS należy wykonać następujące dodatkowe kroki konfiguracyjne:

1. Utwórz następujące obiekty JNDI w repozytorium JNDI serwera WebSphere Application Server :
   a. Miejsce docelowe kolejki produktu JMS o nazwie `HelloWorld`
      - Ustaw nazwę JNDI na `jms/HelloWorld`
      - Ustaw nazwę kolejki na `HelloWorld`
   b. Fabryka połączeń kolejki produktu JMS o nazwie `HelloWorldQCF`
      - Ustaw nazwę JNDI na `jms/HelloWorldQCF`
      - Ustaw nazwę menedżera kolejek na `QM1`
   c. Fabryka połączeń kolejki produktu JMS o nazwie `WebServicesReplyQCF`
      - Ustaw nazwę JNDI na `jms/WebServicesReplyQCF`
      - Ustaw nazwę menedżera kolejek na `QM1`
2. Utwórz port nasłuchiwania komunikatów o nazwie `HelloWorldPort` w pliku WebSphere Application Server z następującą konfiguracją:
   - Ustaw nazwę JNDI fabryki połączeń na `jms/HelloWorldQCF`
   - Ustaw nazwę JNDI miejsca docelowego na `jms/HelloWorld`
3. Zainstaluj aplikację `HelloWorldEJBEAR.ear` usługi Web Service na serwerze WebSphere Application Server w następujący sposób:
   a. Kliknij opcję **Aplikacje** > **Nowa aplikacja** > **Nowa aplikacja korporacyjna**.
   b. Przejdź do katalogu *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym IBM MQ.
   c. Nie zmieniaj żadnej z opcji domyślnych w kreatorze i zrestartuj serwer aplikacji po zainstalowaniu aplikacji.

Po zakończeniu konfiguracji serwera WAS przetestuj usługę, uruchamiając ją raz:

1. Przejdź do katalogu roboczego Soap over JMS .
2. Wprowadź następującą komendę, aby uruchomić przykład: *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\TestClient.exe` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym IBM MQ.

## O tym zadaniu

Przykład demonstruje połączenie klienta WCF z przykładową usługą WebSphere Application Server SOAP over JMS udostępnianą w przykładach WCF dołączonych do produktu IBM MQprzy użyciu kształtu kanału żądanie-odpowiedź. Komunikaty przepływają między systemem WCF a produktem WebSphere Application Server przy użyciu kolejek systemu IBM MQ . Usługa implementuje metodę `HelloWorld(...)` , która pobiera łańcuch i zwraca powitanie do klienta.

Klient został wygenerowany przy użyciu narzędzia svcutil w celu pobrania metadanych usługi z oddzielnie ujawnionego punktu końcowego HTTP zgodnie z opisem w sekcji “Generowanie plików konfiguracyjnych aplikacji i proxy klienta WCF przy użyciu narzędzia svcutil z metadanymi z działającej usługi” na stronie 1322

Przykład został skonfigurowany przy użyciu konkretnych nazw zasobów zgodnie z opisem w poniższej procedurze. Jeśli konieczna jest zmiana nazw zasobów, należy również zmienić odpowiednią wartość w aplikacji klienckiej w pliku *MQ_INSTALLATION_PATH*

`\tools\wcf\samples\WAS\default\client\app.config` oraz w aplikacji usługowej w *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym IBM MQ.

Usługa i klient są oparte na usłudze i kliencie, które zostały opisane w IBM Developer artykule *Budowanie usługi WWW JMS przy użyciu protokołu SOAP korzystającego z produktów JMS i WebSphere Studio*. Więcej informacji na temat tworzenia usług Web Service protokołu SOAP korzystającego z protokołu JMS , które są zgodne z kanałem niestandardowym produktu IBM MQ WCF, zawiera sekcja https://www.ibm.com/developerworks/websphere/library/techarticles/0402_du/0402_du.html.

## Procedura

Uruchom klienta raz: uruchom plik *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\default\client\bin\Release\TestClient.exe` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym IBM MQ.

Aplikacja kliencka uruchamia obie metody usługi w tym samym czasie, wysyłając dwa komunikaty do przykładowej kolejki.

## Wyniki

Aplikacja usługi pobiera komunikaty z kolejki przykładowej i udostępnia odpowiedź na wywołanie metody `HelloWorld(...)` , które aplikacja kliencka wysyła do konsoli.

# Uwagi

Niniejsza publikacja została opracowana z myślą o produktach i usługach oferowanych w Stanach Zjednoczonych.

IBM może nie oferować w innych krajach produktów, usług lub opcji omawianych w tej publikacji. Informacje o produktach i usługach dostępnych w danym kraju można uzyskać od lokalnego przedstawiciela IBM. Odwołanie do produktu, programu lub usługi IBM nie oznacza, że można użyć wyłącznie tego produktu, programu lub usługi IBM. Zamiast nich można zastosować ich odpowiednik funkcjonalny pod warunkiem, że nie narusza to praw własności intelektualnej firmy IBM. Jednakże cała odpowiedzialność za ocenę przydatności i sprawdzenie działania produktu, programu lub usługi pochodzących od producenta innego niż IBM spoczywa na użytkowniku.

IBM może posiadać patenty lub złożone wnioski patentowe na towary i usługi, o których mowa w niniejszej publikacji. Przedstawienie niniejszej publikacji nie daje żadnych uprawnień licencyjnych do tychże patentów. Pisemne zapytania w sprawie licencji można przesyłać na adres:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Zapytania w sprawie licencji dotyczących informacji kodowanych przy użyciu dwubajtowych zestawów znaków (DBCS) należy kierować do lokalnych działów IBM Intellectual Property Department lub zgłaszać na piśmie pod adresem:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

**Poniższy akapit nie obowiązuje w Wielkiej Brytanii, a także w innych krajach, w których jego treść pozostaje w sprzeczności z przepisami prawa miejscowego:** INTERNATIONAL BUSINESS MACHINES CORPORATION DOSTARCZA TĘ PUBLIKACJĘ W STANIE, W JAKIM SIĘ ZNAJDUJE ("AS IS"), BEZ JAKICHKOLWIEK GWARANCJI (RĘKOJMIĘ RÓWNIEŻ WYŁĄCZA SIĘ), WYRAŹNYCH LUB DOMNIEMANYCH, A W SZCZEGÓLNOŚCI DOMNIEMANYCH GWARANCJI PRZYDATNOŚCI HANDLOWEJ, PRZYDATNOŚCI DO OKREŚLONEGO CELU ORAZ GWARANCJI, ŻE PUBLIKACJA TA NIE NARUSZA PRAW OSÓB TRZECICH. Ustawodawstwa niektórych krajów nie dopuszczają zastrzeżeń dotyczących gwarancji wyraźnych lub domniemanych w odniesieniu do pewnych transakcji; w takiej sytuacji powyższe zdanie nie ma zastosowania.

Informacje zawarte w niniejszej publikacji mogą zawierać nieścisłości techniczne lub błędy typograficzne. Informacje te są okresowo aktualizowane, a zmiany te zostaną uwzględnione w kolejnych wydaniach tej publikacji. IBM zastrzega sobie prawo do wprowadzania ulepszeń i/lub zmian w produktach i/lub programach opisanych w tej publikacji w dowolnym czasie, bez wcześniejszego powiadomienia.

Wszelkie wzmianki w tej publikacji na temat stron internetowych innych podmiotów zostały wprowadzone wyłącznie dla wygody użytkowników i w żadnym wypadku nie stanowią zachęty do ich odwiedzania. Materiały dostępne na tych stronach nie są częścią materiałów opracowanych dla tego produktu IBM, a użytkownik korzysta z nich na własną odpowiedzialność.

IBM ma prawo do używania i rozpowszechniania informacji przysłanych przez użytkownika w dowolny sposób, jaki uzna za właściwy, bez żadnych zobowiązań wobec ich autora.

Licencjobiorcy tego programu, którzy chcieliby uzyskać informacje na temat programu w celu: (i) wdrożenia wymiany informacji między niezależnie utworzonymi programami i innymi programami (łącznie

z tym opisywanym) oraz (ii) wspólnego wykorzystywania wymienianych informacji, powinni skontaktować się z:

IBM Corporation
Koordynator współdziałania oprogramowania, dział 49XA
3605 Autostrada 52 N
Rochester, MN 55901
U.S.A.

Informacje takie mogą być udostępnione, o ile spełnione zostaną odpowiednie warunki, w tym, w niektórych przypadkach, zostanie uiszczona stosowna opłata.

Licencjonowany program opisany w niniejszej publikacji oraz wszystkie inne licencjonowane materiały dostępne dla tego programu są dostarczane przez IBM na warunkach określonych w Umowie IBM z Klientem, Międzynarodowej Umowie Licencyjnej IBM na Program lub w innych podobnych umowach zawartych między IBM i użytkownikami.

Wszelkie dane dotyczące wydajności zostały zebrane w kontrolowanym środowisku. W związku z tym rezultaty uzyskane w innych środowiskach operacyjnych mogą się znacząco różnić. Niektóre pomiary mogły być dokonywane na systemach będących w fazie rozwoju i nie ma gwarancji, że pomiary wykonane na ogólnie dostępnych systemach dadzą takie same wyniki. Niektóre z pomiarów mogły być estymowane przez ekstrapolację. Rzeczywiste wyniki mogą być inne. Użytkownicy powinni we własnym zakresie sprawdzić odpowiednie dane dla ich środowiska.

Informacje dotyczące produktów innych niż produkty IBM pochodzą od dostawców tych produktów, z opublikowanych przez nich zapowiedzi lub innych powszechnie dostępnych źródeł. Firma IBM nie testowała tych produktów i nie może potwierdzić dokładności pomiarów wydajności, kompatybilności ani żadnych innych danych związanych z tymi produktami. Pytania dotyczące możliwości produktów innych podmiotów należy kierować do dostawców tych produktów.

Wszelkie stwierdzenia dotyczące przyszłych kierunków rozwoju i zamierzeń IBM mogą zostać zmienione lub wycofane bez powiadomienia.

Publikacja ta zawiera przykładowe dane i raporty używane w codziennych operacjach działalności gospodarczej. W celu kompleksowego ich zilustrowania, podane przykłady zawierają nazwiska osób prywatnych, nazwy przedsiębiorstw oraz nazwy produktów. Wszystkie te nazwy/nazwiska są fikcyjne i jakiekolwiek podobieństwo do istniejących nazw/nazwisk i adresów jest całkowicie przypadkowe.

LICENCJA W ZAKRESIE PRAW AUTORSKICH:

Niniejsza publikacja zawiera przykładowe aplikacje w kodzie źródłowym, ilustrujące techniki programowania w różnych systemach operacyjnych. Użytkownik może kopiować, modyfikować i dystrybuować te programy przykładowe w dowolnej formie bez uiszczania opłat na rzecz IBM, w celu projektowania, używania, sprzedaży lub dystrybucji aplikacji zgodnych z aplikacyjnym interfejsem programistycznym dla tego systemu operacyjnego, dla którego napisane zostały programy przykładowe. Programy przykładowe nie zostały gruntownie przetestowane. IBM nie może zatem gwarantować ani sugerować niezawodności, użyteczności i funkcjonalności tych programów.

W przypadku przeglądania niniejszych informacji w formie elektronicznej, zdjęcia i kolorowe ilustracje mogą nie być wyświetlane.

## Informacje dotyczące interfejsu programistycznego

Informacje o interfejsie programistycznym, jeśli są dostępne, mają na celu pomóc w tworzeniu aplikacji do użycia z tym programem.

Podręcznik ten zawiera informacje na temat interfejsów programistycznych, które umożliwiają klientom pisanie programów w celu uzyskania dostępu do usług produktu WebSphere MQ.

Informacje te mogą również zawierać informacje na temat diagnostyki, modyfikacji i strojenia. Tego typu informacje są udostępniane jako pomoc przy debugowaniu aplikacji.

**Ważne:** Informacji o diagnostyce, modyfikacji i strojeniu nie należy używać jako interfejsu programistycznego, ponieważ mogą one ulec zmianie.

## Znaki towarowe

IBM, logo IBM , ibm.comsą znakami towarowymi IBM Corporation zarejestrowanymi w wielu systemach prawnych na całym świecie. Aktualna lista znaków towarowych IBM dostępna jest w serwisie WWW IBM, w sekcji "Copyright and trademark information"(Informacje o prawach autorskich i znakach towarowych), pod adresemwww.ibm.com/legal/copytrade.shtml. Nazwy innych produktów lub usług mogą być znakami towarowymi IBM lub innych podmiotów.

Microsoft oraz Windows są znakami towarowymi firmy Microsoft Corporation w Stanach Zjednoczonych i/lub innych krajach.

UNIX jest zastrzeżonym znakiem towarowym The Open Group w Stanach Zjednoczonych i/lub w innych krajach.

Linux jest zastrzeżonym znakiem towarowym Linusa Torvaldsa w Stanach Zjednoczonych i/lub w innych krajach.

Ten produkt zawiera oprogramowanie opracowane przez Eclipse Project (https://www.eclipse.org/).

Java oraz wszystkie znaki towarowe i logo dotyczące języka Java są znakami towarowymi lub zastrzeżonymi znakami towarowymi Oracle i/lub przedsiębiorstw afiliowanych Oracle.

Numer pozycji:

(1P) P/N: